xlrd

Release 1.1.0

Sep 11, 2017

Contents

xlrd is a library for reading data and formatting information from Excel files, whether they are .xls or .xlsx files.

CHAPTER 1

Handling of Unicode

This package presents all text strings as Python unicode objects. From Excel 97 onwards, text in Excel spreadsheets has been stored as UTF-16LE (a 16-bit Unicode Transformation Format). Older files (Excel 95 and earlier) don't keep strings in Unicode; a `CODEPAGE` record provides a codepage number (for example, 1252) which is used by xlrd to derive the encoding (for same example: "cp1252") which is used to translate to Unicode.

If the `CODEPAGE` record is missing (possible if the file was created by third-party software), `xlrd` will assume that the encoding is ascii, and keep going. If the actual encoding is not ascii, a `UnicodeDecodeError` exception will be raised and you will need to determine the encoding yourself, and tell xlrd:

```
book = xlrd.open_workbook(..., encoding_override="cp1252")
```

If the `CODEPAGE` record exists but is wrong (for example, the codepage number is 1251, but the strings are actually encoded in koi8_r), it can be overridden using the same mechanism.

The supplied `runxlrd.py` has a corresponding command-line argument, which may be used for experimentation:

```
runxlrd.py -e koi8_r 3rows myfile.xls
```

The first place to look for an encoding, the "codec name", is the Python documentation.

CHAPTER 2

Dates in Excel spreadsheets

In reality, there are no such things. What you have are floating point numbers and pious hope. There are several problems with Excel dates:

1. Dates are not stored as a separate data type; they are stored as floating point numbers and you have to rely on:

   - the "number format" applied to them in Excel and/or
   - knowing which cells are supposed to have dates in them.

   This module helps with the former by inspecting the format that has been applied to each number cell; if it appears to be a date format, the cell is classified as a date rather than a number.

   Feedback on this feature, especially from non-English-speaking locales, would be appreciated.

2. Excel for Windows stores dates by default as the number of days (or fraction thereof) since `1899-12-31T00:00:00`. Excel for Macintosh uses a default start date of `1904-01-01T00:00:00`.

   The date system can be changed in Excel on a per-workbook basis (for example: Tools -> Options -> Calculation, tick the "1904 date system" box). This is of course a bad idea if there are already dates in the workbook. There is no good reason to change it even if there are no dates in the workbook.

   Which date system is in use is recorded in the workbook. A workbook transported from Windows to Macintosh (or vice versa) will work correctly with the host Excel.

   When using this package's *`xldate_as_tuple()`* function to convert numbers from a workbook, you must use the `datemode` attribute of the `Book` object. If you guess, or make a judgement depending on where you believe the workbook was created, you run the risk of being 1462 days out of kilter.

   Reference: https://support.microsoft.com/en-us/help/180162/xl-the-1900-date-system-vs.-the-1904-date-system

3. The Excel implementation of the Windows-default 1900-based date system works on the incorrect premise that 1900 was a leap year. It interprets the number 60 as meaning `1900-02-29`, which is not a valid date.

   Consequently, any number less than 61 is ambiguous. For example, is 59 the result of `1900-02-28` entered directly, or is it `1900-03-01` minus 2 days?

   The OpenOffice.org Calc program "corrects" the Microsoft problem; entering `1900-02-27` causes the number 59 to be stored. Save as an XLS file, then open the file with Excel and you'll see `1900-02-28` displayed.

Reference: https://support.microsoft.com/en-us/help/214326/excel-incorrectly-assumes-that-the-year-1900-is-a-leap-year

4. The Macintosh-default 1904-based date system counts `1904-01-02` as day 1 and `1904-01-01` as day zero. Thus any number such that `(0.0 <= number < 1.0)` is ambiguous. Is 0.625 a time of day (`15:00:00`), independent of the calendar, or should it be interpreted as an instant on a particular day (`1904-01-01T15:00:00`)?

   The functions in *xldate* take the view that such a number is a calendar-independent time of day (like Python's `datetime.time` type) for both date systems. This is consistent with more recent Microsoft documentation. For example, the help file for Excel 2002, which says that the first day in the 1904 date system is `1904-01-02`.

5. Usage of the Excel `DATE()` function may leave strange dates in a spreadsheet. Quoting the help file in respect of the 1900 date system:

```
If year is between 0 (zero) and 1899 (inclusive),
Excel adds that value to 1900 to calculate the year.
For example, DATE(108,1,2) returns January 2, 2008 (1900+108).
```

   This gimmick, semi-defensible only for arguments up to 99 and only in the pre-Y2K-awareness era, means that `DATE(1899, 12, 31)` is interpreted as `3799-12-31`.

   For further information, please refer to the documentation for the functions in *xldate*.

CHAPTER 3

Named references, constants, formulas, and macros

A name is used to refer to a cell, a group of cells, a constant value, a formula, or a macro. Usually the scope of a name is global across the whole workbook. However it can be local to a worksheet. For example, if the sales figures are in different cells in different sheets, the user may define the name "Sales" in each sheet. There are built-in names, like "Print_Area" and "Print_Titles"; these two are naturally local to a sheet.

To inspect the names with a user interface like MS Excel, OOo Calc, or Gnumeric, click on Insert -> Names -> Define. This will show the global names, plus those local to the currently selected sheet.

A *Book* object provides two dictionaries (*Book.name_map* and *Book.name_and_scope_map*) and a list (*Book.name_obj_list*) which allow various ways of accessing the *Name* objects. There is one *Name* object for each *NAME* record found in the workbook. *Name* objects have many attributes, several of which are relevant only when `obj.macro` is `1`.

In the examples directory you will find `namesdemo.xls` which showcases the many different ways that names can be used, and `xlrdnamesAPIdemo.py` which offers 3 different queries for inspecting the names in your files, and shows how to extract whatever a name is referring to. There is currently one "convenience method", *Name.cell()*, which extracts the value in the case where the name refers to a single cell. The source code for *Name.cell()* is an extra source of information on how the *Name* attributes hang together.

Note: Name information is *not* extracted from files older than Excel 5.0 (`Book.biff_version < 50`).

CHAPTER 4

Formatting information in Excel Spreadsheets

Introduction

This collection of features, new in xlrd version 0.6.1, is intended to provide the information needed to:

- display/render spreadsheet contents (say) on a screen or in a PDF file
- copy spreadsheet data to another file without losing the ability to display/render it.

The Palette; Colour Indexes

A colour is represented in Excel as a `(red, green, blue)` ("RGB") tuple with each component in `range(256)`. However it is not possible to access an unlimited number of colours; each spreadsheet is limited to a palette of 64 different colours (24 in Excel 3.0 and 4.0, 8 in Excel 2.0). Colours are referenced by an index ("colour index") into this palette.

Colour indexes 0 to 7 represent 8 fixed built-in colours: black, white, red, green, blue, yellow, magenta, and cyan.

The remaining colours in the palette (8 to 63 in Excel 5.0 and later) can be changed by the user. In the Excel 2003 UI, Tools -> Options -> Color presents a palette of 7 rows of 8 colours. The last two rows are reserved for use in charts.

The correspondence between this grid and the assigned colour indexes is NOT left-to-right top-to-bottom.

Indexes 8 to 15 correspond to changeable parallels of the 8 fixed colours – for example, index 7 is forever cyan; index 15 starts off being cyan but can be changed by the user.

The default colour for each index depends on the file version; tables of the defaults are available in the source code. If the user changes one or more colours, a `PALETTE` record appears in the XLS file – it gives the RGB values for *all* changeable indexes.

Note that colours can be used in "number formats": `[CYAN]...` and `[COLOR8]...` refer to colour index 7; `[COLOR16]...` will produce cyan unless the user changes colour index 15 to something else.

In addition, there are several "magic" colour indexes used by Excel:

0x18 (BIFF3-BIFF4), 0x40 (BIFF5-BIFF8): System window text colour for border lines (used in XF, CF, and WINDOW2 records)

0x19 (BIFF3-BIFF4), 0x41 (BIFF5-BIFF8): System window background colour for pattern background (used in XF and CF records)

0x43: System face colour (dialogue background colour)

0x4D: System window text colour for chart border lines

0x4E: System window background colour for chart areas

0x4F: Automatic colour for chart border lines (seems to be always Black)

0x50: System ToolTip background colour (used in note objects)

0x51: System ToolTip text colour (used in note objects)

0x7FFF: System window text colour for fonts (used in FONT and CF records).

Note: 0x7FFF appears to be the *default* colour index. It appears quite often in FONT records.

Default Formatting

Default formatting is applied to all empty cells (those not described by a cell record):

- Firstly, row default information (ROW record, *Rowinfo* class) is used if available.
- Failing that, column default information (COLINFO record, *Colinfo* class) is used if available.
- As a last resort the worksheet/workbook default cell format will be used; this should always be present in an Excel file, described by the XF record with the fixed index 15 (0-based). By default, it uses the worksheet/workbook default cell style, described by the very first XF record (index 0).

Formatting features not included in xlrd

- Asian phonetic text (known as "ruby"), used for Japanese furigana. See OOo docs s3.4.2 (p15)
- Conditional formatting. See OOo docs s5.12, s6.21 (CONDFMT record), s6.16 (CF record)
- Miscellaneous sheet-level and book-level items, e.g. printing layout, screen panes.
- Modern Excel file versions don't keep most of the built-in "number formats" in the file; Excel loads formats according to the user's locale. Currently, xlrd's emulation of this is limited to a hard-wired table that applies to the US English locale. This may mean that currency symbols, date order, thousands separator, decimals separator, etc are inappropriate.

Note: This does not affect users who are copying XLS files, only those who are visually rendering cells.

CHAPTER 5

Loading worksheets on demand

This feature, new in version 0.7.1, is governed by the `on_demand` argument to the *`open_workbook()`* function and allows saving memory and time by loading only those sheets that the caller is interested in, and releasing sheets when no longer required.

`on_demand=False` (default): No change. *`open_workbook()`* loads global data and all sheets, releases resources no longer required (principally the `str` or `mmap.mmap` object containing the Workbook stream), and returns.

`on_demand=True` and BIFF version < 5.0: A warning message is emitted, `on_demand` is recorded as `False`, and the old process is followed.

`on_demand=True` and BIFF version >= 5.0: *`open_workbook()`* loads global data and returns without releasing resources. At this stage, the only information available about sheets is *`Book.nsheets`* and *`Book.sheet_names()`*.

`Book.sheet_by_name()` and *`Book.sheet_by_index()`* will load the requested sheet if it is not already loaded.

`Book.sheets()` will load all unloaded sheets.

The caller may save memory by calling *`Book.unload_sheet()`* when finished with the sheet. This applies irrespective of the state of `on_demand`.

The caller may re-load an unloaded sheet by calling *`Book.sheet_by_name()`* or *`Book.sheet_by_index()`*, except if the required resources have been released (which will have happened automatically when `on_demand` is false). This is the only case where an exception will be raised.

The caller may query the state of a sheet using *`Book.sheet_loaded()`*.

`Book.release_resources()` may used to save memory and close any memory-mapped file before proceeding to examine already-loaded sheets. Once resources are released, no further sheets can be loaded.

When using on-demand, it is advisable to ensure that *`Book.release_resources()`* is always called, even if an exception is raised in your own code; otherwise if the input file has been memory-mapped, the `mmap.mmap` object will not be closed and you will not be able to access the physical file until your Python process terminates. This can be done by calling *`Book.release_resources()`* explicitly in the finally part of a try/finally block.

The Book object is also a context manager, so you can wrap your code in a `with` statement that will make sure underlying resources are closed.

CHAPTER 6

XML vulnerabilities and Excel files

If your code ingests `.xlsx` files that come from sources in which you do not have absolute trust, please be aware that `.xlsx` files are made up of XML and, as such, are susceptible to the vulnerabilities of XML.

xlrd uses ElementTree to parse XML, but as you'll find if you look into it, there are many different ElementTree implementations. A good summary of vulnerabilities you should worry can be found here: XML vulnerabilities.

For clarity, xlrd will try and import ElementTree from the following sources. The list is in priority order, with those earlier in the list being preferred to those later in the list:

1. xml.etree.cElementTree
2. cElementTree
3. lxml.etree
4. xml.etree.ElementTree
5. elementtree.ElementTree

To guard against these problems, you should consider the defusedxml project which can be used as follows:

```
import defusedxml
from defusedxml.common import EntitiesForbidden
from xlrd import open_workbook
defusedxml.defuse_stdlib()


def secure_open_workbook(**kwargs):
    try:
        return open_workbook(**kwargs)
    except EntitiesForbidden:
        raise ValueError('Please use a xlsx file without XEE')
```

CHAPTER 7

API Reference

xlrd

xlrd.**open_workbook**(*filename=None*, *logfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *verbosity=0*, *use_mmap=1*, *file_contents=None*, *encoding_override=None*, *formatting_info=False*, *on_demand=False*, *ragged_rows=False*)

Open a spreadsheet file for data extraction.

Parameters

- **filename** – The path to the spreadsheet file to be opened.
- **logfile** – An open file to which messages and diagnostics are written.
- **verbosity** – Increases the volume of trace material written to the logfile.
- **use_mmap** – Whether to use the mmap module is determined heuristically. Use this arg to override the result.

  Current heuristic: mmap is used if it exists.
- **file_contents** – A string or an `mmap.mmap` object or some other behave-alike object. If `file_contents` is supplied, `filename` will not be used, except (possibly) in messages.
- **encoding_override** – Used to overcome missing or bad codepage information in older-version files. See *Handling of Unicode*.
- **formatting_info** – The default is `False`, which saves memory. In this case, "Blank" cells, which are those with their own formatting information but no data, are treated as empty by ignoring the file's `BLANK` and `MULBLANK` records. This cuts off any bottom or right "margin" of rows of empty or blank cells. Only *`cell_value()`* and *`cell_type()`* are available.

  When `True`, formatting information will be read from the spreadsheet file. This provides all cells, including empty and blank cells. Formatting information is available for each cell.

Note that this will raise a NotImplementedError when used with an xlsx file.

- **on_demand** – Governs whether sheets are all loaded initially or when demanded by the caller. See *Loading worksheets on demand*.
- **ragged_rows** – The default of `False` means all rows are padded out with empty cells so that all rows have the same size as found in *ncols*.

  `True` means that there are no empty cells at the ends of rows. This can result in substantial memory savings if rows are of widely varying sizes. See also the *row_len()* method.

Returns An instance of the *Book* class.

xlrd.**dump**(*filename*, *outfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *unnumbered=False*)

For debugging: dump an XLS file's BIFF records in char & hex.

Parameters

- **filename** – The path to the file to be dumped.
- **outfile** – An open file, to which the dump is written.
- **unnumbered** – If true, omit offsets (for meaningful diffs).

xlrd.**count_records**(*filename*, *outfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

For debugging and analysis: summarise the file's BIFF records. ie: produce a sorted file of `(record_name, count)`.

Parameters

- **filename** – The path to the file to be summarised.
- **outfile** – An open file, to which the summary is written.

xlrd.biffh

exception xlrd.biffh.**XLRDError**

An exception indicating problems reading data from an Excel file.

class xlrd.biffh.**BaseObject**

Parent of almost all other classes in the package. Defines a common *dump()* method for debugging.

dump(*f=None*, *header=None*, *footer=None*, *indent=0*)

Parameters

- **f** – open file object, to which the dump is written
- **header** – text to write before the dump
- **footer** – text to write after the dump
- **indent** – number of leading spaces (for recursive calls)

xlrd.biffh.**error_text_from_code** = {0: '#NULL!', 36: '#NUM!', 23: '#REF!', 42: '#N/A', 7: '#DIV/0!', 29: '#NAME?

This dictionary can be used to produce a text version of the internal codes that Excel uses for error cells.

xlrd.biffh.**unpack_unicode**(*data*, *pos*, *lenlen=2*)

Return unicode_strg

xlrd.biffh.**unpack_unicode_update_pos**(*data*, *pos*, *lenlen=2*, *known_len=None*)

Return (unicode_strg, updated value of pos)

xlrd.book

class `xlrd.book.`**Name**
Information relating to a named reference, formula, macro, etc.

Note: Name information is **not** extracted from files older than Excel 5.0 (`Book.biff_version < 50`)

---

hidden = 0
0 = Visible; 1 = Hidden

func = 0
0 = Command macro; 1 = Function macro. Relevant only if macro == 1

vbasic = 0
0 = Sheet macro; 1 = VisualBasic macro. Relevant only if macro == 1

macro = 0
0 = Standard name; 1 = Macro name

complex = 0
0 = Simple formula; 1 = Complex formula (array formula or user defined).

Note: No examples have been sighted.

---

builtin = 0
0 = User-defined name; 1 = Built-in name

Common examples: `Print_Area`, `Print_Titles`; see OOo docs for full list

funcgroup = 0
Function group. Relevant only if macro == 1; see OOo docs for values.

binary = 0
0 = Formula definition; 1 = Binary data

Note: No examples have been sighted.

---

name_index = 0
The index of this object in book.name_obj_list

raw_formula = b'‘
An 8-bit string.

scope = -1

-1: The name is global (visible in all calculation sheets).

-2: The name belongs to a macro sheet or VBA sheet.

-3: The name is invalid.

0 <= scope < book.nsheets: The name is local to the sheet whose index is scope.

result = None
The result of evaluating the formula, if any. If no formula, or evaluation of the formula encountered problems, the result is `None`. Otherwise the result is a single instance of the *Operand* class.

cell()
This is a convenience method for the frequent use case where the name refers to a single cell.

Returns An instance of the *Cell* class.

Raises ***xlrd.biffh.XLRDError*** – The name is not a constant absolute reference to a single cell.

area2d(*clipped=True*)
This is a convenience method for the use case where the name refers to one rectangular area in one worksheet.

Parameters **clipped** – If `True`, the default, the returned rectangle is clipped to fit in `(0, sheet.nrows, 0, sheet.ncols)`. it is guaranteed that `0 <= rowxlo <= rowxhi <= sheet.nrows` and that the number of usable rows in the area (which may be zero) is `rowxhi - rowxlo`; likewise for columns.

Returns a tuple `(sheet_object, rowxlo, rowxhi, colxlo, colxhi)`.

Raises ***xlrd.biffh.XLRDError*** – The name is not a constant absolute reference to a single area in a single sheet.

class `xlrd.book.`**Book**
Contents of a "workbook".

> **Warning:** You should not instantiate this class yourself. You use the *Book* object that was returned when you called *open_workbook()*.

datemode = 0
Which date system was in force when this file was last saved.

0: 1900 system (the Excel for Windows default).

1: 1904 system (the Excel for Macintosh default).

Defaults to 0 in case it's not specified in the file.

biff_version = 0
Version of BIFF (Binary Interchange File Format) used to create the file. Latest is 8.0 (represented here as 80), introduced with Excel 97. Earliest supported by this module: 2.0 (represented as 20).

codepage = None
An integer denoting the character set used for strings in this file. For BIFF 8 and later, this will be 1200, meaning Unicode; more precisely, UTF_16_LE. For earlier versions, this is used to derive the appropriate Python encoding to be used to convert to Unicode. Examples: `1252 -> 'cp1252'`, `10000 -> 'mac_roman'`

encoding = None
The encoding that was derived from the codepage.

countries = (0, 0)
A tuple containing the telephone country code for:

[0]: the user-interface setting when the file was created.

[1]: the regional settings.

Example: `(1, 61)` meaning `(USA, Australia)`.

This information may give a clue to the correct encoding for an unknown codepage. For a long list of observed values, refer to the OpenOffice.org documentation for the `COUNTRY` record.

user_name = ‘’
What (if anything) is recorded as the name of the last user to save the file.

font_list = []
A list of *Font* class instances, each corresponding to a FONT record.

New in version 0.6.1.

format_list = []
A list of *Format* objects, each corresponding to a FORMAT record, in the order that they appear in the input file. It does *not* contain builtin formats.

If you are creating an output file using (for example) xlwt, use this list.

The collection to be used for all visual rendering purposes is *format_map*.

New in version 0.6.1.

format_map = {}
The mapping from *format_key* to *Format* object.

New in version 0.6.1.

load_time_stage_1 = -1.0
Time in seconds to extract the XLS image as a contiguous string (or mmap equivalent).

load_time_stage_2 = -1.0
Time in seconds to parse the data from the contiguous string (or mmap equivalent).

sheets()

Returns A list of all sheets in the book.

All sheets not already loaded will be loaded.

sheet_by_index(*sheetx*)

Parameters **sheetx** – Sheet index in range(nsheets)

Returns A *Sheet*.

sheet_by_name(*sheet_name*)

Parameters **sheet_name** – Name of the sheet required.

Returns A *Sheet*.

sheet_names()

Returns A list of the names of all the worksheets in the workbook file. This information is available even when no sheets have yet been loaded.

sheet_loaded(*sheet_name_or_index*)

Parameters **sheet_name_or_index** – Name or index of sheet enquired upon

Returns True if sheet is loaded, False otherwise.

New in version 0.7.1.

unload_sheet(*sheet_name_or_index*)

Parameters **sheet_name_or_index** – Name or index of sheet to be unloaded.

New in version 0.7.1.

release_resources()
This method has a dual purpose. You can call it to release memory-consuming objects and (possibly) a memory-mapped file (*mmap.mmap* object) when you have finished loading sheets in *on_demand* mode, but still require the *Book* object to examine the loaded sheets. It is also called automatically (a) when *open_workbook()* raises an exception and (b) if you are using a *with* statement, when the *with* block is exited. Calling this method multiple times on the same object has no ill effect.

name_and_scope_map = {}

A mapping from (lower_case_name, scope) to a single *Name* object.

New in version 0.6.0.

name_map = {}
A mapping from *lower_case_name* to a list of *Name* objects. The list is sorted in scope order. Typically there will be one item (of global scope) in the list.

New in version 0.6.0.

nsheets = 0
The number of worksheets present in the workbook file. This information is available even when no sheets have yet been loaded.

name_obj_list = []
List containing a *Name* object for each *NAME* record in the workbook.

New in version 0.6.0.

colour_map = {}
This provides definitions for colour indexes. Please refer to *The Palette; Colour Indexes* for an explanation of how colours are represented in Excel.

Colour indexes into the palette map into *(red, green, blue)* tuples. "Magic" indexes e.g. *0x7FFF* map to *None*.

colour_map is what you need if you want to render cells on screen or in a PDF file. If you are writing an output XLS file, use *palette_record*.

Note: Extracted only if *open_workbook(..., formatting_info=True)*

New in version 0.6.1.

palette_record = []
If the user has changed any of the colours in the standard palette, the XLS file will contain a *PALETTE* record with 56 (16 for Excel 4.0 and earlier) RGB values in it, and this list will be e.g. *[(r0, b0, g0), ..., (r55, b55, g55)]*. Otherwise this list will be empty. This is what you need if you are writing an output XLS file. If you want to render cells on screen or in a PDF file, use *colour_map*.

Note: Extracted only if *open_workbook(..., formatting_info=True)*

New in version 0.6.1.

xf_list = []
A list of *XF* class instances, each corresponding to an *XF* record.

New in version 0.6.1.

style_name_map = {}
This provides access via name to the extended format information for both built-in styles and user-defined styles.

It maps `name` to `(built_in, xf_index)`, where `name` is either the name of a user-defined style, or the name of one of the built-in styles. Known built-in names are Normal, RowLevel_1 to RowLevel_7, ColLevel_1 to ColLevel_7, Comma, Currency, Percent, "Comma [0]", "Currency [0]", Hyperlink, and "Followed Hyperlink".

`built_in` has the following meanings

1: built-in style

0: user-defined

`xf_index` is an index into *Book.xf_list*.

References: OOo docs s6.99 (`STYLE` record); Excel UI Format/Style

New in version 0.6.1.

Extracted only if `open_workbook(..., formatting_info=True)`

New in version 0.7.4.

`xlrd.book.`**unpack_SST_table**(*datatab*, *nstrings*)
Return list of strings

xlrd.compdoc

Implements the minimal functionality required to extract a "Workbook" or "Book" stream (as one big string) from an OLE2 Compound Document file.

`xlrd.compdoc.`**SIGNATURE = b'\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1'**
Magic cookie that should appear in the first 8 bytes of the file.

class `xlrd.compdoc.`**CompDoc**(*mem*, *logfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *DEBUG=0*)
Compound document handler.

Parameters **mem** – The raw contents of the file, as a string, or as an `mmap.mmap` object. The only operation it needs to support is slicing.

get_named_stream(*qname*)
Interrogate the compound document's directory; return the stream as a string if found, otherwise return `None`.

Parameters **qname** – Name of the desired stream e.g. `u'Workbook'`. Should be in Unicode or convertible thereto.

locate_named_stream(*qname*)
Interrogate the compound document's directory.

If the named stream is not found, `(None, 0, 0)` will be returned.

If the named stream is found and is contiguous within the original byte sequence (`mem`) used when the document was opened, then `(mem, offset_to_start_of_stream, length_of_stream)` is returned.

Otherwise a new string is built from the fragments and `(new_string, 0, length_of_stream)` is returned.

Parameters **qname** – Name of the desired stream e.g. `u'Workbook'`. Should be in Unicode or convertible thereto.

xlrd.formatting

Module for formatting information.

`xlrd.formatting.`**nearest_colour_index**(*colour_map*, *rgb*, *debug=0*)
General purpose function. Uses Euclidean distance. So far used only for pre-BIFF8 `WINDOW2` record. Doesn't have to be fast. Doesn't have to be fancy.

class `xlrd.formatting.`**EqNeAttrs**
This mixin class exists solely so that *Format*, *Font*, and *XF* objects can be compared by value of their attributes.

class `xlrd.formatting.`**Font**
An Excel "font" contains the details of not only what is normally considered a font, but also several other display attributes. Items correspond to those in the Excel UI's Format -> Cells -> Font tab.

New in version 0.6.1.

bold = 0
1 = Characters are bold. Redundant; see "weight" attribute.

character_set = 0
Values:

```
0 = ANSI Latin
1 = System default
2 = Symbol,
77 = Apple Roman,
128 = ANSI Japanese Shift-JIS,
129 = ANSI Korean (Hangul),
130 = ANSI Korean (Johab),
134 = ANSI Chinese Simplified GBK,
136 = ANSI Chinese Traditional BIG5,
161 = ANSI Greek,
162 = ANSI Turkish,
163 = ANSI Vietnamese,
177 = ANSI Hebrew,
178 = ANSI Arabic,
186 = ANSI Baltic,
204 = ANSI Cyrillic,
222 = ANSI Thai,
238 = ANSI Latin II (Central European),
255 = OEM Latin I
```

colour_index = 0
An explanation of "colour index" is given in *The Palette; Colour Indexes*.

escapement = 0
1 = Superscript, 2 = Subscript.

family = 0
Values:

```
0 = None (unknown or don't care)
1 = Roman (variable width, serifed)
```

```
2 = Swiss (variable width, sans-serifed)
3 = Modern (fixed width, serifed or sans-serifed)
4 = Script (cursive)
5 = Decorative (specialised, for example Old English, Fraktur)
```

font_index = 0
The 0-based index used to refer to this Font() instance. Note that index 4 is never used; xlrd supplies a dummy place-holder.

height = 0
Height of the font (in twips). A twip = 1/20 of a point.

italic = 0
1 = Characters are italic.

name = ''
The name of the font. Example: `u"Arial"`.

struck_out = 0
1 = Characters are struck out.

underline_type = 0
Values:

```
0 = None
1 = Single;  0x21 (33) = Single accounting
2 = Double;  0x22 (34) = Double accounting
```

underlined = 0
1 = Characters are underlined. Redundant; see *underline_type* attribute.

weight = 400
Font weight (100-1000). Standard values are 400 for normal text and 700 for bold text.

outline = 0
1 = Font is outline style (Macintosh only)

shadow = 0
1 = Font is shadow style (Macintosh only)

class xlrd.formatting.**Format**(*format_key*, *ty*, *format_str*)
"Number format" information from a FORMAT record.

New in version 0.6.1.

format_key = 0
The key into *format_map*

type = 0
A classification that has been inferred from the format string. Currently, this is used only to distinguish between numbers and dates. Values:

```
FUN = 0 # unknown
FDT = 1 # date
FNU = 2 # number
FGE = 3 # general
FTX = 4 # text
```

format_str = ''
The format string

xlrd.formatting.**fmt_bracketed_sub**()
Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

class xlrd.formatting.**XFBorder**
A collection of the border-related attributes of an XF record. Items correspond to those in the Excel UI's Format -> Cells -> Border tab.

An explanations of "colour index" is given in *The Palette; Colour Indexes*.

There are five line style attributes; possible values and the associated meanings are:

```
0 = No line,
1 = Thin,
2 = Medium,
3 = Dashed,
4 = Dotted,
5 = Thick,
6 = Double,
7 = Hair,
8 = Medium dashed,
9 = Thin dash-dotted,
10 = Medium dash-dotted,
11 = Thin dash-dot-dotted,
12 = Medium dash-dot-dotted,
13 = Slanted medium dash-dotted.
```

The line styles 8 to 13 appear in BIFF8 files (Excel 97 and later) only. For pictures of the line styles, refer to OOo docs s3.10 (p22) "Line Styles for Cell Borders (BIFF3-BIFF8)".</p>

New in version 0.6.1.

top_colour_index = 0
The colour index for the cell's top line

bottom_colour_index = 0
The colour index for the cell's bottom line

left_colour_index = 0
The colour index for the cell's left line

right_colour_index = 0
The colour index for the cell's right line

diag_colour_index = 0
The colour index for the cell's diagonal lines, if any

top_line_style = 0
The line style for the cell's top line

bottom_line_style = 0
The line style for the cell's bottom line

left_line_style = 0
The line style for the cell's left line

right_line_style = 0
The line style for the cell's right line

diag_line_style = 0
The line style for the cell's diagonal lines, if any

diag_down = 0
1 = draw a diagonal from top left to bottom right

diag_up = 0
1 = draw a diagonal from bottom left to top right

class xlrd.formatting.**XFBackground**
A collection of the background-related attributes of an XF record. Items correspond to those in the Excel UI's Format -> Cells -> Patterns tab.

An explanations of "colour index" is given in *The Palette; Colour Indexes*.

New in version 0.6.1.

fill_pattern = 0
See section 3.11 of the OOo docs.

background_colour_index = 0
See section 3.11 of the OOo docs.

pattern_colour_index = 0
See section 3.11 of the OOo docs.

class xlrd.formatting.**XFAlignment**
A collection of the alignment and similar attributes of an XF record. Items correspond to those in the Excel UI's Format -> Cells -> Alignment tab.

New in version 0.6.1.

hor_align = 0
Values: section 6.115 (p 214) of OOo docs

vert_align = 0
Values: section 6.115 (p 215) of OOo docs

rotation = 0
Values: section 6.115 (p 215) of OOo docs.

Note: file versions BIFF7 and earlier use the documented orientation attribute; this will be mapped (without loss) into *rotation*.

---

text_wrapped = 0
1 = text is wrapped at right margin

indent_level = 0
A number in range(15).

shrink_to_fit = 0
1 = shrink font size to fit text into cell.

text_direction = 0
0 = according to context; 1 = left-to-right; 2 = right-to-left

class xlrd.formatting.**XFProtection**
A collection of the protection-related attributes of an XF record. Items correspond to those in the Excel UI's Format -> Cells -> Protection tab. Note the OOo docs include the "cell or style" bit in this bundle of attributes. This is incorrect; the bit is used in determining which bundles to use.

New in version 0.6.1.

cell_locked = 0
1 = Cell is prevented from being changed, moved, resized, or deleted (only if the sheet is protected).

formula_hidden = 0
1 = Hide formula so that it doesn't appear in the formula bar when the cell is selected (only if the sheet is protected).

class xlrd.formatting.**XF**
eXtended Formatting information for cells, rows, columns and styles.

Each of the 6 flags below describes the validity of a specific group of attributes.

In cell XFs:

- `flag==0` means the attributes of the parent style `XF` are used, (but only if the attributes are valid there);
- `flag==1` means the attributes of this `XF` are used.

In style XFs:

- `flag==0` means the attribute setting is valid;
- `flag==1` means the attribute should be ignored.

Note: the API provides both "raw" XFs and "computed" XFs. In the latter case, cell XFs have had the above inheritance mechanism applied.

New in version 0.6.1.

is_style = 0
0 = cell XF, 1 = style XF

parent_style_index = 0
cell XF: Index into Book.xf_list of this XF's style XF

style XF: 0xFFF

xf_index = 0
Index into *xf_list*

font_index = 0
Index into *font_list*

format_key = 0
Key into *format_map*

> **Warning:** OOo docs on the XF record call this "Index to FORMAT record". It is not an index in the Python sense. It is a key to a map. It is true *only* for Excel 4.0 and earlier files that the key into format_map from an XF instance is the same as the index into format_list, and *only* if the index is less than 164.

protection = None
An instance of an *XFProtection* object.

background = None
An instance of an *XFBackground* object.

alignment = None
An instance of an *XFAlignment* object.

border = None
An instance of an *XFBorder* object.

xlrd.formula

Module for parsing/evaluating Microsoft Excel formulas.

class xlrd.formula.**Operand**(*akind=None*, *avalue=None*, *arank=0*, *atext='?'*)
Used in evaluating formulas. The following table describes the kinds and how their values are represented.

kind = 0
oUNK means that the kind of operand is not known unambiguously.

value = None
None means that the actual value of the operand is a variable (depends on cell data), not a constant.

text = '?'
The reconstituted text of the original formula. Function names will be in English irrespective of the original language, which doesn't seem to be recorded anywhere. The separator is ",", not ";" or whatever else might be more appropriate for the end-user's locale; patches welcome.

class xlrd.formula.**Ref3D**(*atuple*)
Represents an absolute or relative 3-dimensional reference to a box of one or more cells.

The coords attribute is a tuple of the form:

```
(shtxlo, shtxhi, rowxlo, rowxhi, colxlo, colxhi)
```

where 0 <= thingxlo <= thingx < thingxhi.

Note: It is quite possible to have thingx > nthings; for example Print_Titles could have colxhi == 256 and/or rowxhi == 65536 irrespective of how many columns/rows are actually used in the worksheet. The caller will need to decide how to handle this situation. Keyword: IndexError :-)

The components of the coords attribute are also available as individual attributes: shtxlo, shtxhi, rowxlo, rowxhi, colxlo, and colxhi.

The relflags attribute is a 6-tuple of flags which indicate whether the corresponding (sheet|row|col)(lo|hi) is relative (1) or absolute (0).

Note: There is necessarily no information available as to what cell(s) the reference could possibly be relative to. The caller must decide what if any use to make of oREL operands.

New in version 0.6.0.

xlrd.formula.**cellname**(*rowx*, *colx*)
Utility function: (5, 7) => 'H6'

xlrd.formula.**cellnameabs**(*rowx*, *colx*, *r1c1=0*)
Utility function: (5, 7) => '$H$6'

xlrd.formula.**colname**(*colx*)
Utility function: 7 => 'H', 27 => 'AB'

xlrd.formula.**rangename3d**(*book*, *ref3d*)
Utility function: Ref3D(1, 4, 5, 20, 7, 10) => 'Sheet2:Sheet3!$H$6:$J$20' (assuming Excel's default sheetnames)

xlrd.formula.**rangename3drel**(*book*, *ref3d*, *browx=None*, *bcolx=None*, *r1c1=0*)
Utility function: Ref3D(coords=(0, 1, -32, -22, -13, 13), relflags=(0, 0, 1, 1, 1, 1))

In R1C1 mode => `'Sheet1!R[-32]C[-13]:R[-23]C[12]'`

In A1 mode => depends on base cell `(browx, bcolx)`

xlrd.sheet

class `xlrd.sheet.`**Sheet**(*book*, *position*, *name*, *number*)
Contains the data for one worksheet.

In the cell access functions, `rowx` is a row index, counting from zero, and `colx` is a column index, counting from zero. Negative values for row/column indexes and slice positions are supported in the expected fashion.

For information about cell types and cell values, refer to the documentation of the *Cell* class.

Warning: You don't instantiate this class yourself. You access *Sheet* objects via the *Book* object that was returned when you called *xlrd.open_workbook()*.

col(*colx*)
Returns a sequence of the *Cell* objects in the given column.

gcw
A 256-element tuple corresponding to the contents of the GCW record for this sheet. If no such record, treat as all bits zero. Applies to BIFF4-7 only. See docs of the *Colinfo* class for discussion.

vert_split_pos = 0
Number of columns in left pane (frozen panes; for split panes, see comments in code)

horz_split_pos = 0
Number of rows in top pane (frozen panes; for split panes, see comments in code)

horz_split_first_visible = 0
Index of first visible row in bottom frozen/split pane

vert_split_first_visible = 0
Index of first visible column in right frozen/split pane

split_active_pane = 0
Frozen panes: ignore it. Split panes: explanation and diagrams in OOo docs.

has_pane_record = 0
Boolean specifying if a `PANE` record was present, ignore unless you're `xlutils.copy`

book = None
A reference to the *Book* object to which this sheet belongs.

Example usage: `some_sheet.book.datemode`

name = ''
Name of sheet.

nrows = 0
Number of rows in sheet. A row index is in `range(thesheet.nrows)`.

ncols = 0
Nominal number of columns in sheet. It is one more than the maximum column index found, ignoring trailing empty cells. See also the `ragged_rows` parameter to *open_workbook()* and *row_len()*.

defcolwidth = None
Default column width from `DEFCOLWIDTH` record, else `None`. From the OOo docs:

Column width in characters, using the width of the zero character from default font (first FONT record in the file). Excel adds some extra space to the default width, depending on the default font and default font size. The algorithm how to exactly calculate the resulting column width is not known. Example: The default width of 8 set in this record results in a column width of 8.43 using Arial font with a size of 10 points.

For the default hierarchy, refer to the *Colinfo* class.

New in version 0.6.1.

standardwidth = None
Default column width from STANDARDWIDTH record, else None.

From the OOo docs:

Default width of the columns in 1/256 of the width of the zero character, using default font (first FONT record in the file).

For the default hierarchy, refer to the *Colinfo* class.

New in version 0.6.1.

default_row_height = None
Default value to be used for a row if there is no ROW record for that row. From the *optional* DEFAULTROWHEIGHT record.

default_row_height_mismatch = None
Default value to be used for a row if there is no ROW record for that row. From the *optional* DEFAULTROWHEIGHT record.

default_row_hidden = None
Default value to be used for a row if there is no ROW record for that row. From the *optional* DEFAULTROWHEIGHT record.

default_additional_space_above = None
Default value to be used for a row if there is no ROW record for that row. From the *optional* DEFAULTROWHEIGHT record.

default_additional_space_below = None
Default value to be used for a row if there is no ROW record for that row. From the *optional* DEFAULTROWHEIGHT record.

colinfo_map = {}
The map from a column index to a *Colinfo* object. Often there is an entry in COLINFO records for all column indexes in range(257).

Note: xlrd ignores the entry for the non-existent 257th column.

On the other hand, there may be no entry for unused columns.

New in version 0.6.1.

Populated only if open_workbook(..., formatting_info=True)

rowinfo_map = {}
The map from a row index to a *Rowinfo* object.

..note:: It is possible to have missing entries – at least one source of XLS files doesn't bother writing ROW records.

New in version 0.6.1.

Populated only if `open_workbook(..., formatting_info=True)`

col_label_ranges = []
List of address ranges of cells containing column labels. These are set up in Excel by Insert > Name > Labels > Columns.

New in version 0.6.0.

How to deconstruct the list:

```
for crange in thesheet.col_label_ranges:
    rlo, rhi, clo, chi = crange
    for rx in xrange(rlo, rhi):
        for cx in xrange(clo, chi):
            print "Column label at (rowx=%d, colx=%d) is %r" \
                (rx, cx, thesheet.cell_value(rx, cx))
```

row_label_ranges = []
List of address ranges of cells containing row labels. For more details, see *col_label_ranges*.

New in version 0.6.0.

merged_cells = []
List of address ranges of cells which have been merged. These are set up in Excel by Format > Cells > Alignment, then ticking the "Merge cells" box.

Note: The upper limits are exclusive: i.e. `[2, 3, 7, 9]` only spans two cells.

Note: Extracted only if `open_workbook(..., formatting_info=True)`

New in version 0.6.1.

How to deconstruct the list:

```
for crange in thesheet.merged_cells:
    rlo, rhi, clo, chi = crange
    for rowx in xrange(rlo, rhi):
        for colx in xrange(clo, chi):
            # cell (rlo, clo) (the top left one) will carry the data
            # and formatting info; the remainder will be recorded as
            # blank cells, but a renderer will apply the formatting info
            # for the top left cell (e.g. border, pattern) to all cells in
            # the range.
```

rich_text_runlist_map = {}
Mapping of `(rowx, colx)` to list of `(offset, font_index)` tuples. The offset defines where in the string the font begins to be used. Offsets are expected to be in ascending order. If the first offset is not zero, the meaning is that the cell's `XF`'s font should be used from offset 0.

This is a sparse mapping. There is no entry for cells that are not formatted with rich text.

How to use:

```
runlist = thesheet.rich_text_runlist_map.get((rowx, colx))
if runlist:
    for offset, font_index in runlist:
        # do work here.
        pass
```

New in version 0.7.2.

Populated only if `open_workbook(..., formatting_info=True)`

horizontal_page_breaks = []
A list of the horizontal page breaks in this sheet. Breaks are tuples in the form `(index of row after break, start col index, end col index)`.

Populated only if `open_workbook(..., formatting_info=True)`

New in version 0.7.2.

vertical_page_breaks = []
A list of the vertical page breaks in this sheet. Breaks are tuples in the form `(index of col after break, start row index, end row index)`.

Populated only if `open_workbook(..., formatting_info=True)`

New in version 0.7.2.

visibility = 0
Visibility of the sheet:

```
0 = visible
1 = hidden (can be unhidden by user -- Format -> Sheet -> Unhide)
2 = "very hidden" (can be unhidden only by VBA macro).
```

hyperlink_list = []
A list of *Hyperlink* objects corresponding to `HLINK` records found in the worksheet.

New in version 0.7.2.

hyperlink_map = {}
A sparse mapping from `(rowx, colx)` to an item in *hyperlink_list*. Cells not covered by a hyperlink are not mapped. It is possible using the Excel UI to set up a hyperlink that covers a larger-than-1x1 rectangle of cells. Hyperlink rectangles may overlap (Excel doesn't check). When a multiply-covered cell is clicked on, the hyperlink that is activated (and the one that is mapped here) is the last in *hyperlink_list*.

New in version 0.7.2.

cell_note_map = {}
A sparse mapping from `(rowx, colx)` to a *Note* object. Cells not containing a note ("comment") are not mapped.

New in version 0.7.2.

cell(*rowx*, *colx*)
Cell object in the given row and column.

cell_value(*rowx*, *colx*)
Value of the cell in the given row and column.

cell_type(*rowx*, *colx*)
Type of the cell in the given row and column.

Refer to the documentation of the *Cell* class.

cell_xf_index(*rowx*, *colx*)
XF index of the cell in the given row and column. This is an index into *xf_list*.

New in version 0.6.1.

row_len(*rowx*)
Returns the effective number of cells in the given row. For use with `open_workbook(ragged_rows=True)` which is likely to produce rows with fewer than *ncols* cells.

New in version 0.7.2.

row(*rowx*)
Returns a sequence of the *Cell* objects in the given row.

get_rows()
Returns a generator for iterating through each row.

row_types(*rowx*, *start_colx=0*, *end_colx=None*)
Returns a slice of the types of the cells in the given row.

row_values(*rowx*, *start_colx=0*, *end_colx=None*)
Returns a slice of the values of the cells in the given row.

row_slice(*rowx*, *start_colx=0*, *end_colx=None*)
Returns a slice of the *Cell* objects in the given row.

col_slice(*colx*, *start_rowx=0*, *end_rowx=None*)
Returns a slice of the *Cell* objects in the given column.

col_values(*colx*, *start_rowx=0*, *end_rowx=None*)
Returns a slice of the values of the cells in the given column.

col_types(*colx*, *start_rowx=0*, *end_rowx=None*)
Returns a slice of the types of the cells in the given column.

computed_column_width(*colx*)
Determine column display width.

Parameters **colx** – Index of the queried column, range 0 to 255. Note that it is possible to find out the width that will be used to display columns with no cell information e.g. column IV (colx=255).

Returns The column width that will be used for displaying the given column by Excel, in units of 1/256th of the width of a standard character (the digit zero in the first font).

New in version 0.6.1.

class xlrd.sheet.**Note**
Represents a user "comment" or "note". Note objects are accessible through *Sheet.cell_note_map*.

New in version 0.7.2.

author = ''
Author of note

col_hidden = 0
`True` if the containing column is hidden

colx = 0
Column index

rich_text_runlist = None
List of `(offset_in_string, font_index)` tuples. Unlike *Sheet.rich_text_runlist_map*, the first offset should always be 0.

row_hidden = 0
True if the containing row is hidden

rowx = 0
Row index

show = 0
True if note is always shown

text = ''
Text of the note

class xlrd.sheet.**Hyperlink**
Contains the attributes of a hyperlink. Hyperlink objects are accessible through *Sheet.hyperlink_list* and *Sheet.hyperlink_map*.

New in version 0.7.2.

frowx = None
Index of first row

lrowx = None
Index of last row

fcolx = None
Index of first column

lcolx = None
Index of last column

type = None
Type of hyperlink. Unicode string, one of 'url', 'unc', 'local file', 'workbook', 'unknown'

url_or_path = None
The URL or file-path, depending in the type. Unicode string, except in the rare case of a local but non-existent file with non-ASCII characters in the name, in which case only the "8.3" filename is available, as a `bytes` (3.x) or `str` (2.x) string, *with unknown encoding.*

desc = None
Description. This is displayed in the cell, and should be identical to the cell value. Unicode string, or `None`. It seems impossible NOT to have a description created by the Excel UI.

target = None
Target frame. Unicode string.

Note: No cases of this have been seen in the wild. It seems impossible to create one in the Excel UI.

---

textmark = None
The piece after the "#" in "http://docs.python.org/library#struct_module", or the `Sheet1!A1:Z99` part when type is "workbook".

quicktip = None
The text of the "quick tip" displayed when the cursor hovers over the hyperlink.

class xlrd.sheet.**Cell**(*ctype*, *value*, *xf_index=None*)
Contains the data for one cell.

Warning: You don't call this class yourself. You access *Cell* objects via methods of the *Sheet* object(s) that you found in the *Book* object that was returned when you called *open_workbook()*

Cell objects have three attributes: `ctype` is an int, `value` (which depends on `ctype`) and `xf_index`. If `formatting_info` is not enabled when the workbook is opened, `xf_index` will be `None`.

The following table describes the types of cells and how their values are represented in Python.

class xlrd.sheet.**Colinfo**
Width and default formatting information that applies to one or more columns in a sheet. Derived from COLINFO records.

Here is the default hierarchy for width, according to the OOo docs:

> In BIFF3, if a COLINFO record is missing for a column, the width specified in the record DEFCOLWIDTH is used instead.
>
> In BIFF4-BIFF7, the width set in this COLINFO record is only used, if the corresponding bit for this column is cleared in the GCW record, otherwise the column width set in the DEFCOLWIDTH record is used (the STANDARDWIDTH record is always ignored in this case[1]).
>
> In BIFF8, if a COLINFO record is missing for a column, the width specified in the record STANDARDWIDTH is used. If this STANDARDWIDTH record is also missing, the column width of the record DEFCOLWIDTH is used instead.

xlrd goes with the GCW version of the story. Reference to the source may be useful: see *Sheet.computed_column_width()*.

New in version 0.6.1.

width = 0
Width of the column in 1/256 of the width of the zero character, using default font (first FONT record in the file).

xf_index = -1
XF index to be used for formatting empty cells.

hidden = 0
1 = column is hidden

bit1_flag = 0
Value of a 1-bit flag whose purpose is unknown but is often seen set to 1

outline_level = 0
Outline level of the column, in range(7). (0 = no outline)

collapsed = 0
1 = column is collapsed

class xlrd.sheet.**Rowinfo**
Height and default formatting information that applies to a row in a sheet. Derived from ROW records.

New in version 0.6.1.

height
Height of the row, in twips. One twip == 1/20 of a point.

has_default_height
0 = Row has custom height; 1 = Row has default height.

outline_level
Outline level of the row (0 to 7)

[1] The docs on the GCW record say this:
If a bit is set, the corresponding column uses the width set in the STANDARDWIDTH record. If a bit is cleared, the corresponding column uses the width set in the COLINFO record for this column.
If a bit is set, and the worksheet does not contain the STANDARDWIDTH record, or if the bit is cleared, and the worksheet does not contain the COLINFO record, the DEFCOLWIDTH record of the worksheet will be used instead.

outline_group_starts_ends
1 = Outline group starts or ends here (depending on where the outline buttons are located, see `WSBOOL` record, which is not parsed by xlrd), *and* is collapsed.

hidden
1 = Row is hidden (manually, or by a filter or outline group)

height_mismatch
1 = Row height and default font height do not match.

has_default_xf_index
1 = the xf_index attribute is usable; 0 = ignore it.

xf_index
Index to default *XF* record for empty cells in this row. Don't use this if `has_default_xf_index == 0`.

additional_space_above
This flag is set if the upper border of at least one cell in this row or if the lower border of at least one cell in the row above is formatted with a thick line style. Thin and medium line styles are not taken into account.

additional_space_below
This flag is set if the lower border of at least one cell in this row or if the upper border of at least one cell in the row below is formatted with a medium or thick line style. Thin line styles are not taken into account.

xlrd.xldate

Tools for working with dates and times in Excel files.

The conversion from `days` to `(year, month, day)` starts with an integral "julian day number" aka JDN. FWIW:

- JDN 0 corresponds to noon on Monday November 24 in Gregorian year -4713.

More importantly:

- Noon on Gregorian 1900-03-01 (day 61 in the 1900-based system) is JDN 2415080.0
- Noon on Gregorian 1904-01-02 (day 1 in the 1904-based system) is JDN 2416482.0

exception `xlrd.xldate.`**XLDateError**
A base class for all datetime-related errors.

exception `xlrd.xldate.`**XLDateNegative**
`xldate < 0.00`

exception `xlrd.xldate.`**XLDateAmbiguous**
The 1900 leap-year problem `(datemode == 0 and 1.0 <= xldate < 61.0)`

exception `xlrd.xldate.`**XLDateTooLarge**
Gregorian year 10000 or later

exception `xlrd.xldate.`**XLDateBadDatemode**
`datemode` arg is neither 0 nor 1

`xlrd.xldate.`**xldate_as_tuple**(*xldate*, *datemode*)
Convert an Excel number (presumed to represent a date, a datetime or a time) into a tuple suitable for feeding to datetime or mx.DateTime constructors.

Parameters

- **xldate** – The Excel number

- **datemode** – 0: 1900-based, 1: 1904-based.

Raises

- ***xlrd.xldate.XLDateNegative*** –
- ***xlrd.xldate.XLDateAmbiguous*** –
- ***xlrd.xldate.XLDateTooLarge*** –
- ***xlrd.xldate.XLDateBadDatemode*** –
- ***xlrd.xldate.XLDateError*** –

Returns Gregorian `(year, month, day, hour, minute, nearest_second)`.

Warning: When using this function to interpret the contents of a workbook, you should pass in the *datemode* attribute of that workbook. Whether the workbook has ever been anywhere near a Macintosh is irrelevant.

Special case

If `0.0 <= xldate < 1.0`, it is assumed to represent a time; `(0, 0, 0, hour, minute, second)` will be returned.

Note: `1904-01-01` is not regarded as a valid date in the `datemode==1` system; its "serial number" is zero.

`xlrd.xldate.`**xldate_as_datetime**(*xldate*, *datemode*)

Convert an Excel date/time number into a `datetime.datetime` object.

Parameters

- **xldate** – The Excel number
- **datemode** – 0: 1900-based, 1: 1904-based.

Returns A `datetime.datetime` object.

`xlrd.xldate.`**xldate_from_date_tuple**(*date_tuple*, *datemode*)

Convert a date tuple (year, month, day) to an Excel date.

Parameters

- **year** – Gregorian year.
- **month** – `1 <= month <= 12`
- **day** – `1 <= day <= last day of that (year, month)`
- **datemode** – 0: 1900-based, 1: 1904-based.

Raises

- ***xlrd.xldate.XLDateAmbiguous*** –
- ***xlrd.xldate.XLDateBadDatemode*** –
- **xlrd.xldate.XLDateBadTuple** – `(year, month, day)` is too early/late or has invalid component(s)
- ***xlrd.xldate.XLDateError*** –

xlrd.xldate.**xldate_from_time_tuple**(*time_tuple*)
Convert a time tuple `(hour, minute, second)` to an Excel "date" value (fraction of a day).

Parameters

- **hour** – `0 <= hour < 24`
- **minute** – `0 <= minute < 60`
- **second** – `0 <= second < 60`

Raises **xlrd.xldate.XLDateBadTuple** – Out-of-range hour, minute, or second

xlrd.xldate.**xldate_from_datetime_tuple**(*datetime_tuple*, *datemode*)
Convert a datetime tuple `(year, month, day, hour, minute, second)` to an Excel date value. For more details, refer to other xldate_from_*_tuple functions.

Parameters

- **datetime_tuple** – `(year, month, day, hour, minute, second)`
- **datemode** – 0: 1900-based, 1: 1904-based.

You may also wish to consult the tutorial.

For details of how to install the package or get involved in its development, please see the sections below:

CHAPTER 8

Installation Instructions

If you want to experiment with xlrd, the easiest way to install it is to do the following in a virtualenv:

```
pip install xlrd
```

If your package uses setuptools and you decide to use xlrd, then you should add it as a requirement by adding an `install_requires` parameter in your call to `setup` as follows:

```
setup(
    # other stuff here
    install_requires=['xlrd'],
    )
```

CHAPTER 9

Development

This package is developed using continuous integration which can be found here:

https://travis-ci.org/python-excel/xlrd

If you wish to contribute to this project, then you should fork the repository found here:

https://github.com/python-excel/xlrd

Once that has been done and you have a checkout, you can follow these instructions to perform various development tasks:

Setting up a virtualenv

The recommended way to set up a development environment is to turn your checkout into a virtualenv and then install the package in editable form as follows:

```
$ virtualenv .
$ bin/pip install -Ur requirements.txt
$ bin/pip install -e .
```

Running the tests

Once you've set up a virtualenv, the tests can be run as follows:

```
$ bin/nosetests
```

To run tests on all the versions of Python that are supported, you can do:

```
$ bin/tox
```

If you change the supported python versions in `.travis.yml`, please remember to do the following to update `tox.ini`:

```
$ bin/panci --to=tox .travis.yml > tox.ini
```

Building the documentation

The Sphinx documentation is built by doing the following, having activated the virtualenv above, from the directory containing setup.py:

```
$ cd docs
$ make html
```

Making a release

To make a release, just update the version in `xlrd.info.__VERSION__`, update the change log, tag it, push to https://github.com/python-excel/xlrd and Travis CI should take care of the rest.

Once the above is done, make sure to go to https://readthedocs.org/projects/xlrd/versions/ and make sure the new release is marked as an Active Version.

CHAPTER 10

Changes

1.1.0 (22 August 2017)

- Fix for parsing of merged cells containing a single cell reference in xlsx files.
- Fix for "invalid literal for int() with base 10: 'true"' when reading some xlsx files.
- Make xldate_as_datetime available to import direct from xlrd.
- Build universal wheels.
- Sphinx documentation.
- Document the problem with XML vulnerabilities in xlsx files and mitigation measures.
- Fix `NameError` on `has_defaults is not defined`.
- Some whitespace and code style tweaks.
- Make example in README compatible with both Python 2 and 3.
- Add default value for cells containing errors that causeed parsing of some xlsx files to fail.
- Add Python 3.6 to the list of supported Python versions, drop 3.3 and 2.6.
- Use generator expressions to avoid unnecessary lists in memory.
- Document unicode encoding used in Excel files from Excel 97 onwards.
- Report hyperlink errors in R1C1 syntax.

Thanks to the following for their contributions to this release:

- icereval@gmail.com
- Daniel Rech
- Ville Skyttä
- Yegor Yefremov
- Maxime Lorant

- Alexandr N Zamaraev
- Zhaorong Ma
- Jon Dufresne
- Chris McIntyre
- coltleese@gmail.com
- Ivan Masá

1.0.0 (2 June 2016)

- Official support, such as it is, is now for 2.6, 2.7, 3.3+
- Fixes a bug in looking up non-lowercase sheet filenames by ensuring that the sheet targets are transformed the same way as the component_names dict keys.
- Fixes a bug for `ragged_rows=False` when merged cells increases the number of columns in the sheet. This requires all rows to be extended to ensure equal row lengths that match the number of columns in the sheet.
- Fixes to enable reading of SAP-generated .xls files.
- support BIFF4 files with missing FORMAT records.
- support files with missing WINDOW2 record.
- Empty cells are now always unicode strings, they were a bytestring on Python 2 and a unicode string on Python 3.
- Fix for `<cell> inlineStr` attribute without `<si>` child.
- Fix for a zoom of `None` causing problems on Python 3.
- Fix parsing of bad dimensions.
- Fix xlsx sheet to comments relationship.

Thanks to the following for their contributions to this release:

- Lars-Erik Hannelius
- Deshi Xiao
- Stratos Moro
- Volker Diels-Grabsch
- John McNamara
- Ville Skyttä
- Patrick Fuller
- Dragon Dave McKee
- Gunnlaugur Þór Briem

0.9.4 (14 July 2015)

- Automated tests are now run on Python 3.4

- Use `ElementTree.iter()` if available, instead of the deprecated `getiterator()` when parsing xlsx files.
- Fix #106 : Exception Value: unorderable types: Name() < Name()
- Create row generator expression with Sheet.get_rows()
- Fix for forward slash file separator and lowercase names within xlsx internals.

Thanks to the following for their contributions to this release:

- Corey Farwell
- Jonathan Kamens
- Deepak N
- Brandon R. Stoner
- John McNamara

0.9.3 (8 Apr 2014)

- Github issue #49
- Github issue #64 - skip meaningless chunk of 4 zero bytes between two otherwise-valid BIFF records
- Github issue #61 - fix updating of escapement attribute of Font objects read from workbooks.
- Implemented `Sheet.visibility` for xlsx files
- Ignore anchors (`$`) in cell references
- Dropped support for Python 2.5 and earlier, Python 2.6 is now the earliest Python release supported
- Read xlsx merged cell elements.
- Read cell comments in .xlsx files.
- Added xldate_as_datetime() function to convert from Excel serial date/time to datetime.datetime object.

Thanks to the following for their contributions to this release:

- John Machin
- Caleb Epstein
- Martin Panter
- John McNamara
- Gunnlaugur Þór Briem
- Stephen Lewis

0.9.2 (9 Apr 2013)

- Fix some packaging issues that meant docs and examples were missing from the tarball.
- Fixed a small but serious regression that caused problems opening .xlsx files.

0.9.1 (5 Apr 2013)

- Many fixes bugs in Python 3 support.
- Fix bug where ragged rows needed fixing when formatting info was being parsed.
- Improved handling of aberrant Excel 4.0 Worksheet files.
- Various bug fixes.
- Simplify a lot of the distribution packaging.
- Remove unused and duplicate imports.

Thanks to the following for their contributions to this release:

- Thomas Kluyver

0.9.0 (31 Jan 2013)

- Support for Python 3.2+
- Many new unit test added.
- Continuous integration tests are now run.
- Various bug fixes.

Special thanks to Thomas Kluyver and Martin Panter for their work on Python 3 compatibility.

Thanks to Manfred Moitzi for re-licensing his unit tests so we could include them.

Thanks to the following for their contributions to this release:

- “holm”
- Victor Safronovich
- Ross Jones

0.8.0 (22 Aug 2012)

- More work-arounds for broken source files.
- Support for reading .xlsx files.
- Drop support for Python 2.5 and older.

0.7.8 (7 June 2012)

- Ignore superfluous zero bytes at end of xls OBJECT record.
- Fix assertion error when reading file with xlwt-written bitmap.

0.7.7 (13 Apr 2012)

- More packaging changes, this time to support 2to3.

0.7.6 (3 Apr 2012)

- Fix more packaging issues.

0.7.5 (3 Apr 2012)

- Fix packaging issue that missed `version.txt` from the distributions.

0.7.4 (2 Apr 2012)

- More tolerance of out-of-spec files.
- Fix bugs reading long text formula results.

0.7.3 (28 Feb 2012)

- Packaging and documentation updates.

0.7.2 (21 Feb 2012)

- Tolerant handling of files with extra zero bytes at end of NUMBER record. Sample provided by Jan Kraus.
- Added access to cell notes/comments. Many cross-references added to Sheet class docs.
- Added code to extract hyperlink (HLINK) records. Based on a patch supplied by John Morrisey.
- Extraction of rich text formatting info based on code supplied by Nathan van Gheem.
- added handling of BIFF2 WINDOW2 record.
- Included modified version of page breaks patch from Sam Listopad.
- Added reading of the PANE record.
- Reading SCL record. New attribute `Sheet.scl_mag_factor`.
- Lots of bug fixes.
- Added `ragged_rows` functionality.

0.7.1 (31 May 2009)

- Backed out "slash'n'burn" of sheet resources in unload_sheet(). Fixed problem with STYLE records on some Mac Excel files.
- quieten warnings
- Integrated on_demand patch by Armando Serrano Lombillo

0.7.0 (11 March 2009)

- colname utility function now supports more than 256 columns.
- Fix bug where BIFF record type 0x806 was being regarded as a formula opcode.
- Ignore PALETTE record when formatting_info is false.
- Tolerate up to 4 bytes trailing junk on PALETTE record.
- Fixed bug in unused utility function xldate_from_date_tuple which affected some years after 2099.
- Added code for inspecting as-yet-unused record types: FILEPASS, TXO, NOTE.
- Added inspection code for add_in function calls.
- Added support for unnumbered biff_dump (better for doing diffs).
- ignore distutils cruft
- Avoid assertion error in compdoc when -1 used instead of -2 for first_SID of empty SCSS
- Make version numbers match up.
- Enhanced recovery from out-of-order/missing/wrong CODEPAGE record.
- Added Name.area2d convenience method.
- Avoided some checking of XF info when formatting_info is false.
- Minor changes in preparation for XLSX support.
- remove duplicate files that were out of date.
- Basic support for Excel 2.0
- Decouple Book init & load.
- runxlrd: minor fix for xfc.
- More Excel 2.x work.
- is_date_format() tweak.
- Better detection of IronPython.
- Better error message (including first 8 bytes of file) when file is not in a supported format.
- More BIFF2 formatting: ROW, COLWIDTH, and COLUMNDEFAULT records;
- finished stage 1 of XF records.
- More work on supporting BIFF2 (Excel 2.x) files.
- Added support for Excel 2.x (BIFF2) files. Data only, no formatting info. Alpha.
- Wasn't coping with EXTERNSHEET record followed by CONTINUE record(s).

- Allow for BIFF2/3-style FORMAT record in BIFF4/8 file
- Avoid crash when zero-length Unicode string missing options byte.
- Warning message if sector sizes are extremely large.
- Work around corrupt STYLE record
- Added missing entry for blank cell type to ctype_text
- Added "fonts" command to runxlrd script
- Warning: style XF whose parent XF index != 0xFFF
- Logfile arg wasn't being passed from open_workbook to compdoc.CompDoc.

0.6.1 (10 June 2007)

- Version number updated to 0.6.1
- Documented runxlrd.py commands in its usage message. Changed commands: dump to biff_dump, count_records to biff_count.

0.6.1a5

- Bug fixed: Missing "<" in a struct.unpack call means can't open files on bigendian platforms. Discovered by "Mihalis".
- Removed antique undocumented Book.get_name_dict method and experimental "trimming" facility.
- Meaningful exception instead of IndexError if a SAT (sector allocation table) is corrupted.
- If no CODEPAGE record in pre-8.0 file, assume ascii and keep going (instead of raising exception).

0.6.1a4

- At least one source of XLS files writes parent style XF records *after* the child cell XF records that refer to them, triggering IndexError in 0.5.2 and AssertionError in later versions. Reported with sample file by Todd O'Bryan. Fixed by changing to two-pass processing of XF records.
- Formatting info in pre-BIFF8 files: Ensured appropriate defaults and lossless conversions to make the info BIFF8-compatible. Fixed bug in extracting the "used" flags.
- Fixed problems discovered with opening test files from Planmaker 2006 (http://www.softmaker.com/english/ofwcomp_en.htm): (1) Four files have reduced size of PALETTE record (51 and 32 colours; Excel writes 56 always). xlrd now emits a NOTE to the logfile and continues. (2) FORMULA records use the Excel 2.x record code 0x0021 instead of 0x0221. xlrd now continues silently. (3) In two files, at the OLE2 compound document level, the internal directory says that the length of the Short-Stream Container Stream is 16384 bytes, but the actual contents are 11264 and 9728 bytes respectively. xlrd now emits a WARNING to the logfile and continues.
- After discussion with Daniel Rentz, the concept of two lists of XF (eXtended Format) objects (raw_xf_list and computed_xf_list) has been abandoned. There is now a single list, called xf_list

0.6.1a3

- Added Book.sheets ... for sheetx, sheet in enumerate(book.sheets):
- Formatting info: extraction of sheet-level flags from WINDOW2 record, and sheet.visibility from BOUNDSHEET record. Added Macintosh- only Font attributes "outline" and "shadow'.

0.6.1a2

- Added extraction of merged cells info.
- pyExcelerator uses "general" instead of "General" for the generic "number format". Worked around.
- Crystal Reports writes "WORKBOOK" in the OLE2 Compound Document directory instead of "Workbook". Changed to case-insensitive directory search. Reported by Vic Simkus.

0.6.1a1 (18 Dec 2006)

- Added formatting information for cells (font, "number format", background, border, alignment and protection) and rows/columns (height/width etc). To save memory and time for those who don't need it, this information is extracted only if formatting_info=1 is supplied to the open_workbook() function. The cell records BLANK and MULBLANKS which contain no data, only formatting information, will continue to be ignored in the default (no formatting info) case.
- Ralph Heimburger reported a problem with xlrd being intolerant about an Excel 4.0 file (created by "some web app") with a DIMENSIONS record that omitted Microsoft's usual padding with 2 unused bytes. Fixed.

0.6.0a4 (not released)

- Added extraction of human-readable formulas from NAME records.
- Worked around OOo Calc writing 9-byte BOOLERR records instead of 8. Reported by Rory Campbell-Lange.
- This history file converted to descending chronological order and HTML format.

0.6.0a3 (19 Sept 2006)

- Names: minor bugfixes; added script xlrdnameAPIdemo.py
- ROW records were being used as additional hints for sizing memory requirements. In some files the ROW records overstate the number of used columns, and/or there are ROW records for rows that have no data in them. This would cause xlrd to report sheet.ncols and/or sheet.nrows as larger than reasonably expected. Change: ROW records are ignored. The number of columns/rows is based solely on the highest column/row index seen in non-empty data records. Empty data records (types BLANK and MULBLANKS) which contain no data, only formatting information, have always been ignored, and this will continue. Consequence: trailing rows and columns which contain only empty cells will vanish.

0.6.0a2 (13 Sept 2006)

- Fixed a bug reported by Rory Campbell-Lange.: "open failed"; incorrect assumptions about the layout of array formulas which return strings.
- Further work on defined names, especially the API.

0.6.0a1 (8 Sept 2006)

- Sheet objects have two new convenience methods: col_values(colx, start_rowx=0, end_rowx=None) and the corresponding col_types. Suggested by Dennis O'Brien.
- BIFF 8 file missing its CODEPAGE record: xlrd will now assume utf_16_le encoding (the only possibility) and keep going.
- Older files missing a CODEPAGE record: an exception will be raised. Thanks to Sergey Krushinsky for a sample file. The open_workbook() function has a new argument (encoding_override) which can be used if the CODEPAGE record is missing or incorrect (for example, codepage=1251 but the data is actually encoded in koi8_r). The runxlrd.py script takes a corresponding -e argument, for example -e cp1251
- Further work done on parsing "number formats". Thanks to Chris Withers for the "`General_)`" example.
- Excel 97 introduced the concept of row and column labels, defined by Insert > Name > Labels. The ranges containing the labels are now exposed as the Sheet attributes row_label_ranges and col_label_ranges.
- The major effort in this 0.6.0 release has been the provision of access to named cell ranges and named constants (Excel: Insert/Name/Define). Juan C. Mendez provided very useful real-world sample files.

0.5.3a1 (24 May 2006)

- John Popplewell and Richard Sharp provided sample files which caused any reliance at all on DIMENSIONS records and ROW records to be abandoned.
- If the file size is not a whole number of OLE sectors, a warning message is logged. Previously this caused an exception to be raised.

0.5.2 (14 March 2006)

- public release
- Updated version numbers, README, HISTORY.

0.5.2a3 (13 March 2006)

- Gnumeric writes user-defined formats with format codes starting at 50 instead of 164; worked around.
- Thanks to Didrik Pinte for reporting the need for xlrd to be more tolerant of the idiosyncracies of other software, for supplying sample files, and for performing alpha testing.
- '_' character in a format should be treated like an escape character; fixed.
- An "empty" formula result means a zero-length string, not an empty cell! Fixed.

0.5.2a2 (9 March 2006)

- Found that Gnumeric writes all DIMENSIONS records with nrows and ncols each 1 less than they should be (except when it clamps ncols at 256!), and pyXLwriter doesn't write ROW records. Cell memory pre- allocation was generalised to use ROW records if available with fall- back to DIMENSIONS records.

0.5.2a1 (6 March 2006)

- pyXLwriter writes DIMENSIONS record with antique opcode 0x0000 instead of 0x0200; worked around
- A file written by Gnumeric had zeroes in DIMENSIONS record but data in cell A1; worked around

0.5.1 (18 Feb 2006)

- released to Journyx
- Python 2.1 mmap requires file to be opened for update access. Added fall-back to read-only access without mmap if 2.1 open fails because "permission denied".

0.5 (7 Feb 2006)

- released to Journyx
- Now works with Python 2.1. Backporting to Python 2.1 was partially funded by Journyx - provider of timesheet and project accounting solutions (http://journyx.com/)
- open_workbook() can be given the contents of a file instead of its name. Thanks to Remco Boerma for the suggestion.
- New module attribute __VERSION__ (as a string; for example "0.5")
- Minor enhancements to classification of formats as date or not-date.
- Added warnings about files with inconsistent OLE compound document structures. Thanks to Roman V. Kiseliov (author of pyExcelerator) for the tip-off.

0.4a1, (7 Sept 2005)

- released to Laurent T.
- Book and sheet objects can now be pickled and unpickled. Instead of reading a large spreadsheet multiple times, consider pickling it once and loading the saved pickle; can be much faster. Thanks to Laurent Thioudellet for the enhancement request.
- Using the mmap module can be turned off. But you would only do that for benchmarking purposes.
- Handling NUMBER records has been made faster

0.3a1 (15 May 2005)

- first public release

CHAPTER 11

Acknowledgements

Development of this package would not have been possible without the document OpenOffice.org's Documentation of the Microsoft Excel File Format" ("OOo docs" for short). The latest version is available from OpenOffice.org in PDF format and ODT format. Small portions of the OOo docs are reproduced in this document. A study of the OOo docs is recommended for those who wish a deeper understanding of the Excel file layout than the xlrd docs can provide.

Backporting to Python 2.1 was partially funded by Journyx - provider of timesheet and project accounting solutions.

Provision of formatting information in version 0.6.1 was funded by Simplistix Ltd.

CHAPTER 12

Licenses

There are two licenses associated with xlrd. This one relates to the bulk of the work done on the library:

```
Portions copyright © 2005-2009, Stephen John Machin, Lingfo Pty Ltd
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. None of the names of Stephen John Machin, Lingfo Pty Ltd and any
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.
```

This one covers some earlier work:

```
/*-
 * Copyright (c) 2001 David Giffin.
```

```
* All rights reserved.
*
* Based on the the Java version: Andrew Khan Copyright (c) 2000.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. All advertising materials mentioning features or use of this
*    software must display the following acknowledgment:
*    "This product includes software developed by
*     David Giffin <david@giffin.org>."
*
* 4. Redistributions of any form whatsoever must retain the following
*    acknowledgment:
*    "This product includes software developed by
*     David Giffin <david@giffin.org>."
*
* THIS SOFTWARE IS PROVIDED BY DAVID GIFFIN ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL DAVID GIFFIN OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

x

Index

R

S

T

U

V

W

X