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!', 7: '#DIV/0!', 15: '#VALUE!', 23: '#REF!', 29: '#NAME?', 36: '#NUM!', 42: '#N/A'}

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.

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.

---

exception **`xlrd.compdoc.CompDocError`**

---

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. `'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. `'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: `"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 symbol	Kind number	Value representation
oBOOL	3	integer: 0 => False; 1 => True
oERR	4	None, or an int error code (same as XL_CELL_ERROR in the Cell class).
oMSNG	5	Used by Excel as a placeholder for a missing (not supplied) function argument. Should *not* appear as a final formula result. Value is None.
oNUM	2	A float. Note that there is no way of distinguishing dates.

oREF	-1	The value is either None or a non-empty list of absolute Ref3D instances.
oREL	-2	The value is None or a non-empty list of fully or partially relative Ref3D instances.
oSTRG	1	A Unicode string.
oUNK	0	The kind is unknown or ambiguous. The value is None

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.MSODrawing**

class **xlrd.sheet.MSObj**

class **xlrd.sheet.MSTxo**

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.

Type symbol	Type number	Python value
XL_CELL_EMPTY	0	empty string ''
XL_CELL_TEXT	1	a Unicode string
XL_CELL_NUMBER	2	float
XL_CELL_DATE	3	float
XL_CELL_BOOLEAN	4	int; 1 means TRUE, 0 means FALSE
XL_CELL_ERROR	5	int representing internal Excel codes; for a text representation, refer to the supplied dictionary error_text_from_code
XL_CELL_BLANK	6	empty string ''. Note: this type will appear only when open_workbook(..., formatting_info=True) is used.

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.

[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.

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)

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

exception **xlrd.xldate.XLDateBadTuple**

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.