pyExcelerator Documentation

Release 0.6.4.1

Rick van Hattem

Nov 05, 2017

Contents

CHAPTER 1

pyExcelerator Package

1.1 pyExcelerator Package

1.2 BIFFRecords Module

class pyExcelerator.BIFFRecords.**BackupRecord**(*backup*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains a Boolean value determining whether Excel makes a backup of the file while saving.

class pyExcelerator.BIFFRecords.**Biff8BOFRecord**(*rec_type*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Offset Size Contents 0 2 Version, contains 0600H for BIFF8 and BIFF8X 2 2 Type of the following data:

0005H = Workbook globals 0006H = Visual Basic module 0010H = Worksheet 0020H = Chart 0040H = Macro sheet 0100H = Workspace file

4 2 Build identifier 6 2 Build year 8 4 File history flags 12 4 Lowest Excel version that can read all records in this file

BOOK_GLOBAL = 5

CHART = 32

MACROSHEET = 64

VB_MODULE = 6

WORKSHEET = 16

WORKSPACE = 256

class pyExcelerator.BIFFRecords.**BiffRecord**
Bases: object

get()

get_rec_data()

get_rec_header()

get_rec_id()

class pyExcelerator.BIFFRecords.**BlankRecord**(*row*, *col*, *xf_index*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record represents an empty cell.

Record BLANK, BIFF5-BIFF8:

Offset Size Contents 0 2 Index to row 2 2 Index to first column (fc) 4 2 indexes to XF record

class pyExcelerator.BIFFRecords.**BookBoolRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains a Boolean value determining whether to save values linked from external workbooks (CRN records and XCT records). In BIFF3 and BIFF4 this option is stored in the WSBOOL record.

Record BOOKBOOL, BIFF5-BIFF8:

Offset Size Contents 0 2 0 = Save external linked values;

1 = Do not save external linked values

class pyExcelerator.BIFFRecords.**BottomMarginRecord**(*margin*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains the bottom page margin of the current worksheet.

Offset Size Contents 0 8 Bottom page margin in inches

(IEEE 754 floating-point value, 64?bit double precision)

class pyExcelerator.BIFFRecords.**BoundSheetRecord**(*stream_pos*, *visibility*, *sheetname*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is located in the workbook globals area and represents a sheet inside of the workbook. For each sheet a BOUNDSHEET record is written. It stores the sheet name and a stream offset to the BOF record within the workbook stream. The record is also known as BUNDLESHEET.

Record BOUNDSHEET, BIFF5-BIFF8: Offset Size Contents 0 4 Absolute stream position of the BOF record of the sheet represented by this record. This

field is never encrypted in protected files.

4 1 Visibility: 00H = Visible 01H = Hidden 02H = Strong hidden

5 1 Sheet type: 00H = Worksheet 02H = Chart 06H = Visual Basic module

6 var. Sheet name: BIFF5/BIFF7: Byte string, 8-bit string length BIFF8: Unicode string, 8-bit string length

class pyExcelerator.BIFFRecords.**CalcCountRecord**(*calc_count*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It specifies the maximum number of times the formulas should be iteratively calculated. This is a fail-safe against mutually recursive formulas locking up a spreadsheet application.

Record CALCCOUNT, BIFF2-BIFF8:

Offset Size Contents 0 2 Maximum number of iterations allowed in circular references

class pyExcelerator.BIFFRecords.**CalcModeRecord**(*calc_mode*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It specifies whether to calculate formulas manually, automatically or automatically except for multiple table operations.

Record CALCMODE, BIFF2-BIFF8:

Offset Size Contents 0 2 FFFFH = automatic except for multiple table operations

0000H = manually 0001H = automatically (default)

class pyExcelerator.BIFFRecords.**CodepageBiff8Record**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores the text encoding used to write byte strings, stored as MS Windows code page identifier. The CODEPAGE record in BIFF8 always contains the code page 1200 (UTF-16). Therefore it is not possible to obtain the encoding used for a protection password (it is not UTF-16).

Record CODEPAGE, BIFF2-BIFF8:

Offset Size Contents 0 2 Code page identifier used for byte string text encoding:

016FH = 367 = ASCII 01B5H = 437 = IBM PC CP-437 (US) 02D0H = 720 = IBM PC CP-720 (OEM Arabic) 02E1H = 737 = IBM PC CP-737 (Greek) 0307H = 775 = IBM PC CP-775 (Baltic) 0352H = 850 = IBM PC CP-850 (Latin I) 0354H = 852 = IBM PC CP-852 (Latin II (Central European)) 0357H = 855 = IBM PC CP-855 (Cyrillic) 0359H = 857 = IBM PC CP-857 (Turkish) 035AH = 858 = IBM PC CP-858 (Multilingual Latin I with Euro) 035CH = 860 = IBM PC CP-860 (Portuguese) 035DH = 861 = IBM PC CP-861 (Icelandic) 035EH = 862 = IBM PC CP-862 (Hebrew) 035FH = 863 = IBM PC CP-863 (Canadian (French)) 0360H = 864 = IBM PC CP-864 (Arabic) 0361H = 865 = IBM PC CP-865 (Nordic) 0362H = 866 = IBM PC CP-866 (Cyrillic (Russian)) 0365H = 869 = IBM PC CP-869 (Greek (Modern)) 036AH = 874 = Windows CP-874 (Thai) 03A4H = 932 = Windows CP-932 (Japanese Shift-JIS) 03A8H = 936 = Windows CP-936 (Chinese Simplified GBK) 03B5H = 949 = Windows CP-949 (Korean (Wansung)) 03B6H = 950 = Windows CP-950 (Chinese Traditional BIG5) 04B0H = 1200 = UTF-16 (BIFF8) 04E2H = 1250 = Windows CP-1250 (Latin II) (Central European) 04E3H = 1251 = Windows CP-1251 (Cyrillic) 04E4H = 1252 = Windows CP-1252 (Latin I) (BIFF4-BIFF7) 04E5H = 1253 = Windows CP-1253 (Greek) 04E6H = 1254 = Windows CP-1254 (Turkish) 04E7H = 1255 = Windows CP-1255 (Hebrew) 04E8H = 1256 = Windows CP-1256 (Arabic) 04E9H = 1257 = Windows CP-1257 (Baltic) 04EAH = 1258 = Windows CP-1258 (Vietnamese) 0551H = 1361 = Windows CP-1361 (Korean (Johab)) 2710H = 10000 = Apple Roman 8000H = 32768 = Apple Roman 8001H = 32769 = Windows CP-1252 (Latin I) (BIFF2-BIFF3)

UTF_16 = 1200

class pyExcelerator.BIFFRecords.**ColInfoRecord**(*first_col*, *last_col*, *width*, *xf_index*, *options*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies the width for a given range of columns. If a column does not have a corresponding COLINFO record, the width specified in the record STANDARDWIDTH is used. If this record is also not present, the contents of the record DEFCOLWIDTH is used instead. This record also specifies a default XF record to use for cells in the columns that are not described by any cell record (which contain the XF index for that cell). Additionally, the option flags field contains hidden, outline, and collapsed options applied at the columns.

Record COLINFO, BIFF3-BIFF8:

Offset Size Contents 0 2 Index to first column in the range 2 2 Index to last column in the range 4 2 Width of the columns in 1/256 of the width of the zero character, using default font

(first FONT record in the file)

6 2 Index to XF record for default column formatting 8 2 Option flags:

Bits Mask Contents 0 0001H 1 = Columns are hidden 10-8 0700H Outline level of the columns (0 = no outline) 12 1000H 1 = Columns are collapsed

10 2 Not used

class pyExcelerator.BIFFRecords.**ContinueRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Whenever the content of a record exceeds the given limits (see table), the record must be split. Several CONTINUE records containing the additional data are added after the parent record.

BIFF version Maximum data size of a record BIFF2-BIFF7 2080 bytes (2084 bytes including record header) BIFF8 8224 bytes (8228 bytes including record header) (0x2020)

Record CONTINUE, BIFF2-BIFF8: Offset Size Contents 0 var. Data continuation of the previous record

Unicode strings are split in a special way. At the beginning of each CONTINUE record the option flags byte is repeated. Only the character size flag will be set in this flags byte, the Rich-Text flag and the Far-East flag are set to zero. In each CONTINUE record it is possible that the character size changes from 8-bit characters to 16-bit characters and vice versa.

Never a Unicode string is split until and including the first character. That means, all header fields (string length, option flags, optional Rich-Text size, and optional Far-East data size) and the first character of the string have to occur together in the leading record, or have to be moved completely into the CONTINUE record. Formatting runs cannot be split between their components (character index and FONT record index). If a string is split between two formatting runs, the option flags field will not be repeated in the CONTINUE record.

class pyExcelerator.BIFFRecords.**CountryRecord**(*ui_id*, *sys_settings_id*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores two Windows country identifiers. The first represents the user interface language of the Excel version that has saved the file, and the second represents the system regional settings at the time the file was saved.

Record COUNTRY, BIFF3-BIFF8:

Offset Size Contents 0 2 Windows country identifier of the user interface language of Excel 2 2 Windows country identifier of the system regional settings

The following table shows most of the used country identifiers. Most of these identifiers are equal to the international country calling codes.

1 USA 2 Canada 7 Russia

class pyExcelerator.BIFFRecords.**DSFRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies if the file contains an additional BIFF5/BIFF7 workbook stream. Record DSF, BIFF8: Offset Size Contents 0 2 0 = Only the BIFF8 Workbook stream is present

1 = Additional BIFF5/BIFF7 Book stream is in the file

A double stream file can be read by Excel 5.0 and Excel 95, and still contains all new features added to BIFF8 (which are left out in the BIFF5/BIFF7 Book stream).

class pyExcelerator.BIFFRecords.**DateModeRecord**(*from1904*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies the base date for displaying date values. All dates are stored as count of days past this base date. In BIFF2-BIFF4 this record is part of the Calculation Settings Block. In BIFF5-BIFF8 it is stored in the Workbook Globals Substream.

Record DATEMODE, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = Base is 1899-Dec-31 (the cell = 1 represents 1900-Jan-01)

1 = Base is 1904-Jan-01 (the cell = 1 represents 1904-Jan-02)

class pyExcelerator.BIFFRecords.**DefColWidthRecord**(*def_width*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies the default column width for columns that do not have a specific width set using the record COLINFO or COLWIDTH. This record has no effect, if a STANDARDWIDTH record is present in the file.

Record DEFCOLWIDTH, BIFF2-BIFF8:

Offset Size Contents 0 2 Column width in characters, using the width of the zero

character from default font (first FONT record in the file)

class pyExcelerator.BIFFRecords.**DefaultRowHeight**(*options*, *def_height*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies the default height and default flags for rows that do not have a corresponding ROW record.

Record DEFAULTROWHEIGHT, BIFF3-BIFF8:

Offset Size Contents 0 2 Option flags:

Bit Mask Contents 0 0001H 1 = Row height and default font height do not match 1 0002H 1 = Row is hidden 2 0004H 1 = Additional space above the row 3 0008H 1 = Additional space below the row

2 2 Default height for unused rows, in twips = 1/20 of a point

class pyExcelerator.BIFFRecords.**DeltaRecord**(*delta*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It stores the maximum change of the result to exit an iteration.

Record DELTA, BIFF2-BIFF8:

Offset Size Contents 0 8 Maximum change in iteration

(IEEE 754 floating-point value, 64bit double precision)

class pyExcelerator.BIFFRecords.**DimensionsRecord**(*first_used_row*, *last_used_row*, *first_used_col*, *last_used_col*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Record DIMENSIONS, BIFF8:

Offset Size Contents 0 4 Index to first used row 4 4 Index to last used row, increased by 1 8 2 Index to first used column 10 2 Index to last used column, increased by 1 12 2 Not used

class pyExcelerator.BIFFRecords.**EOFRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**ExtSSTRecord**(*sst_stream_pos*, *str_placement*, *portions_len*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record occurs in conjunction with the SST record. It is used by Excel to create a hash table with stream offsets to the SST record to optimise string search operations. Excel may not shorten this record if strings are deleted from the shared string table, so the last part might contain invalid data. The stream indexes in this record divide the SST into portions containing a constant number of strings.

Record EXTSST, BIFF8:

Offset Size Contents 0 2 Number of strings in a portion, this number is >=8 2 var. List of OFFSET structures for all portions. Each OFFSET contains the following data:

Offset Size Contents 0 4 Absolute stream position of first string of the portion 4 2 Position of first string of the portion inside of current record,

including record header. This counter restarts at zero, if the SST record is continued with a CONTINUE record.

6 2 Not used

class pyExcelerator.BIFFRecords.**ExternSheetRecord**(*refs=[]*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

In BIFF8 the record stores a list with indexes to SUPBOOK records (list of REF structures, 6.100). See 5.10.3 for details about external references in BIFF8.

Record EXTERNSHEET, BIFF8: Offset Size Contents

0 2 Number of following REF structures (nm) 2 6nm List of nm REF structures. Each REF contains the following data:

Offset Size Contents 0 2 Index to SUPBOOK record 2 2 Index to first SUPBOOK sheet 4 2 Index to last SUPBOOK sheet

class pyExcelerator.BIFFRecords.**FnGroupCountRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**FontRecord**(*height*, *options*, *colour_index*, *weight*, *escapement*, *underline*, *family*, *charset*, *name*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

WARNING The font with index 4 is omitted in all BIFF versions. This means the first four fonts have zero-based indexes, and the fifth font and all following fonts are referenced with one-based indexes.

Offset Size Contents 0 2 Height of the font (in twips = 1/20 of a point) 2 2 Option flags:

Bit Mask Contents 0 0001H 1 = Characters are bold (redundant, see below) 1 0002H 1 = Characters are italic 2 0004H 1 = Characters are underlined (redundant, see below) 3 0008H 1 = Characters are struck out

0010H 1 = Outline 0020H 1 = Shadow

4 2 Colour index 6 2 Font weight (100-1000).

Standard values are 0190H (400) for normal text and 02BCH (700) for bold text.

8 2 Escapement type: 0000H = None 0001H = Superscript 0002H = Subscript

10 1 Underline type: 00H = None 01H = Single 21H = Single accounting 02H = Double 22H = Double accounting

11 1 Font family: 00H = None (unknown or don't care) 01H = Roman (variable width, serifed) 02H = Swiss (variable width, sans-serifed) 03H = Modern (fixed width, serifed or sans-serifed) 04H = Script (cursive) 05H = Decorative (specialised, i.e. Old English, Fraktur)

12 1 Character set: 00H = 0 = ANSI Latin 01H = 1 = System default 02H = 2 = Symbol 4DH = 77 = Apple Roman 80H = 128 = ANSI Japanese Shift-JIS 81H = 129 = ANSI Korean (Hangul) 82H = 130 = ANSI Korean (Johab) 86H = 134 = ANSI Chinese Simplified GBK 88H = 136 = ANSI Chinese Traditional BIG5 A1H = 161 = ANSI Greek A2H = 162 = ANSI Turkish A3H = 163 = ANSI Vietnamese B1H = 177 = ANSI Hebrew B2H = 178 = ANSI Arabic BAH = 186 = ANSI Baltic CCH = 204 = ANSI Cyrillic DEH = 222 = ANSI Thai EEH = 238 = ANSI Latin II (Central European) FFH = 255 = OEM Latin I

13 1 Not used 14 var. Font name:

BIFF5/BIFF7: Byte string, 8-bit string length BIFF8: Unicode string, 8-bit string length

The boldness and underline flags are still set in the options field, but not used on reading the font. Font weight and underline type are specified in separate fields instead.

class pyExcelerator.BIFFRecords.**FooterRecord**(*footer_str*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Semantic is equal to HEADER record

class pyExcelerator.BIFFRecords.**FormulaRecord**(*row*, *col*, *xf_index*, *result*, *opts*, *rpn*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Offset Size Contents 0 2 Index to row 2 2 Index to column 4 2 Index to XF record 6 8 Result of the formula 14 2 Option flags:

Bit Mask Contents 0 0001H 1 = Recalculate always 1 0002H 1 = Calculate on open 3 0008H 1 = Part of a shared formula

16 4 Not used 20 var. Formula data (RPN token array)

class pyExcelerator.BIFFRecords.**GridSetRecord**(*print_grid_changed*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies if the option to print sheet grid lines (record PRINTGRIDLINES) has ever been changed.

Record GRIDSET, BIFF3-BIFF8:

Offset Size Contents 0 2 0 = Print grid lines option never changed

1 = Print grid lines option changed

class pyExcelerator.BIFFRecords.**GutsRecord**(*row_gut_width*, *col_gut_height*, *row_visible_levels*, *col_visible_levels*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains information about the layout of outline symbols.

Record GUTS, BIFF3-BIFF8:

Offset Size Contents 0 2 Width of the area to display row outlines (left of the sheet), in pixel 2 2 Height of the area to display column outlines (above the sheet), in pixel 4 2 Number of visible row outline levels (used row levels + 1; or 0, if not used) 6 2 Number of visible column outline levels (used column levels + 1; or 0, if not used)

class pyExcelerator.BIFFRecords.**HCenterRecord**(*is_horz_center*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It specifies if the sheet is centred horizontally when printed.

Record HCENTER, BIFF3-BIFF8:

Offset Size Contents 0 2 0 = Print sheet left aligned

1 = Print sheet centred horizontally

class pyExcelerator.BIFFRecords.**HeaderRecord**(*header_str*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It specifies the page header string for the current worksheet. If this record is not present or completely empty (record size is 0), the sheet does not contain a page header.

Record HEADER for non-empty page header, BIFF2-BIFF8: Offset Size Contents 0 var. Page header string

BIFF2-BIFF7: Non-empty byte string, 8bit string length BIFF8: Non-empty Unicode string, 16bit string length

The header string may contain special commands, i.e. placeholders for the page number, current date, or text formatting attributes. These fields are represented by single letters (exception: font name and size, see below) with a leading ampersand ("&"). If the ampersand is part of the regular header text, it will be duplicated ("&&"). The page header is divided into 3 sections: the left, the centred, and the right section. Each section is introduced by a special command. All text and all commands following are part of the selected section. Each section starts with the text formatting specified in the default font (first FONT record in the file). Active formatting attributes from a previous section do not go into the next section.

The following table shows all available commands:

Command Contents && The "&" character itself &L Start of the left section &C Start of the centred section &R Start of the right section &P Current page number &N Page count &D Current date &T Current time &A Sheet name (BIFF5-BIFF8) &F File name without path &Z File path without file name (BIFF8X) &G Picture (BIFF8X) &B Bold on/off (BIFF2-BIFF4) &I Italic on/off (BIFF2-BIFF4) &U Underlining on/off &E Double underlining on/off (BIFF5-BIFF8) &S Strikeout on/off &X Superscript on/off (BIFF5-BIFF8) &Y Subscript on/off (BIFF5-BIFF8) &"<fontname>" Set new font <fontname> &"<fontname>,<fontstyle>"

Set new font with specified style <fontstyle>. The style <fontstyle> is in most cases one of "Regular", "Bold", "Italic", or "Bold Italic". But this setting is dependent on the used font, it may differ (localised style names, or "Standard", "Oblique", ...). (BIFF5-BIFF8)

&<fontheight> Set font height in points (<fontheight> is a decimal value). If this command is followed by a plain number to be printed in the header, it will be separated from the font height with a space character.

class pyExcelerator.BIFFRecords.**HideObjRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies whether and how to show objects in the workbook.

Record HIDEOBJ, BIFF3-BIFF8: Offset Size Contents 0 2 Viewing mode for objects:

0 = Show all objects 1 = Show placeholders 2 = Do not show objects

class pyExcelerator.BIFFRecords.**HorizontalPageBreaksRecord**(*breaks_list*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains all horizontal manual page breaks.

Record HORIZONTALPAGEBREAKS, BIFF8: Offset Size Contents 0 2 Number of following row index structures (nm) 2 6nm List of nm row index structures. Each row index

structure contains: Offset Size Contents 0 2 Index to first row below the page break 2 2 Index to first column of this page break 4 2 Index to last column of this page break

The row indexes in the lists must be ordered ascending. If in BIFF8 a row contains several page breaks, they must be ordered ascending by start column index.

class pyExcelerator.BIFFRecords.**HyperlinkRecord**(*frow*, *lrow*, *fcol*, *lcol*, *url*, *target=None*, *description=None*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Each HLINK record starts with the same data items and continues with special data related to the current type of hyperlink. It starts with a cell range. Each cell of this range will contain the same hyperlink.

Record HLINK, BIFF8:

Offset Size Contents

0 8 Cell range address of all cells containing this hyperlink (3.13.1) 8 16 GUID of StdLink:

D0H C9H EAH 79H F9H BAH CEH 11H 8CH 82H 00H AAH 00H 4BH A9H 0BH (79EAC9D0-BAF9-11CE-8C82-00AA004BA90B)

24 4 Unknown value: 00000002H 28 4 Option flags (see below)

[32] 4 (optional, see option flags) Character count of description text, including trailing zero word

550.

[36] 2dl (optional, see option flags) Character array of description text, no Unicode string header, always 16-bit characters, zero-terminated

[var.] 4 (optional, see option flags) Character count of target frame, including trailing zero word (fl)

[var.] 2fl (optional, see option flags) Character array of target frame, no Unicode string header,

always 16-bit characters, zero-terminated

var. var. Special data (6.53.2 and following)

[var.] 4 (optional, see option flags) Character count of the text mark, including trailing zero word (tl)

[var.] 2tl (optional, see option flags) Character array of the text mark without "#" sign, no Unicode string header, always 16-bit characters, zero-terminated

class pyExcelerator.BIFFRecords.**InteraceEndRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**InteraceHdrRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**InternalReferenceSupBookRecord**(*num_sheets*)
Bases: *pyExcelerator.BIFFRecords.SupBookRecord*

In each file occurs a SUPBOOK that is used for internal 3D references. It stores the number of sheets of the own document.

Record SUPBOOK for 3D references, BIFF8: Offset Size Contents

0 2 Number of sheets in this document 2 2 01H 04H (relict of BIFF5/BIFF7, the byte string "<04H>", see 3.9.1)

class pyExcelerator.BIFFRecords.**IterationRecord**(*iterations_on*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It stores if iterations are allowed while calculating recursive formulas.

Record ITERATION, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = Iterations off; 1 = Iterations on

class pyExcelerator.BIFFRecords.**LabelSSTRecord**(*row*, *col*, *xf_idx*, *sst_idx*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record represents a cell that contains a string. It replaces the LABEL record and RSTRING record used in BIFF2-BIFF7.

class pyExcelerator.BIFFRecords.**LeftMarginRecord**(*margin*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains the left page margin of the current worksheet.

Record LEFTMARGIN, BIFF2-BIFF8:

Offset Size Contents 0 8 Left page margin in inches

(IEEE 754 floating-point value, 64bit double precision)

class pyExcelerator.BIFFRecords.**MMSRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**MergedCellsRecord**(*merged_list*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains all merged cell ranges of the current sheet.

Record MERGEDCELLS, BIFF8:

Offset Size Contents 0 var. Cell range address list with all merged ranges

A cell range address list consists of a field with the number of ranges and the list of the range addresses.

Cell range address list, BIFF8:

Offset Size Contents 0 2 Number of following cell range addresses (nm) 2 8*nm List of nm cell range addresses

Cell range address, BIFF8:

Offset Size Contents 0 2 Index to first row 2 2 Index to last row 4 2 Index to first column 6 2 Index to last column

get()

class pyExcelerator.BIFFRecords.**MulBlankRecord**(*row*, *first_col*, *last_col*, *xf_index*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record represents a cell range of empty cells. All cells are located in the same row.

Record MULBLANK, BIFF5-BIFF8:

Offset Size Contents 0 2 Index to row 2 2 Index to first column (fc) 4 2*nc List of nc=lc-fc+1 16-bit indexes to XF records 4+2*nc 2 Index to last column (lc)

class pyExcelerator.BIFFRecords.**NameRecord**(*options*, *keyboard_shortcut*, *name*, *sheet_index*, *rpn*, *menu_text=''*, *desc_text=''*, *help_text=''*, *status_text=''*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of a Link Table. It contains the name and the token array of an internal defined name. Token arrays of defined names contain tokens with aberrant token classes.

Record NAME, BIFF5/BIFF7: Offset Size Contents

0 2 Option flags, see below 2 1 Keyboard shortcut (only for command macro names, see below) 3 1 Length of the name (character count, ln) 4 2 Size of the formula data (sz) 6 2 0 = Global name, otherwise index to EXTERNSHEET record (one-based) 8 2 0 = Global name, otherwise index to sheet (one-based)

10 1 Length of menu text (character count, lm) 11 1 Length of description text (character count, ld) 12 1 Length of help topic text (character count, lh) 13 1 Length of status bar text (character count, ls) 14 ln Character array of the name

14+ln sz Formula data (RPN token array without size field, 4)

14+ln+sz lm Character array of menu text

var. ld Character array of description text var. lh Character array of help topic text var. ls Character array of status bar text

Record NAME, BIFF8: Offset Size Contents

0 2 Option flags, see below 2 1 Keyboard shortcut (only for command macro names, see below) 3 1 Length of the name (character count, ln) 4 2 Size of the formula data (sz) 6 2 Not used 8 2 0 = Global name, otherwise index to sheet (one-based)

10 1 Length of menu text (character count, lm) 11 1 Length of description text (character count, ld) 12 1 Length of help topic text (character count, lh) 13 1 Length of status bar text (character count, ls) 14 var. Name (Unicode string without length field, 3.4)

var. sz Formula data (RPN token array without size field, 4)

[var.] var. (optional, only if lm > 0) Menu text (Unicode string without length field, 3.4) [var.] var. (optional, only if ld > 0) Description text (Unicode string without length field, 3.4) [var.] var. (optional, only if lh > 0) Help topic text (Unicode string without length field, 3.4) [var.] var. (optional, only if ls > 0) Status bar text (Unicode string without length field, 3.4)

class pyExcelerator.BIFFRecords.**NumberFormatRecord**(*idx*, *fmtstr*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Record FORMAT, BIFF8: Offset Size Contents 0 2 Format index used in other records 2 var. Number format string (Unicode string, 16-bit string length)

From BIFF5 on, the built-in number formats will be omitted. The built-in formats are dependent on the current regional settings of the operating system. The following table shows which number formats are used by default in a US-English environment. All indexes from 0 to 163 are reserved for built-in formats. The first user-defined format starts at 164.

The built-in number formats, BIFF5-BIFF8

Index Type Format string 0 General General 1 Decimal 0 2 Decimal 0.00 3 Decimal #,##0 4 Decimal #,##0.00 5 Currency "$"#,##0_);("$"#,## 6 Currency "$"#,##0_);[Red]("$"#,## 7 Currency "$"#,##0.00_);("$"#,## 8 Currency "$"#,##0.00_);[Red]("$"#,## 9 Percent 0% 10 Percent 0.00% 11 Scientific 0.00E+00 12 Fraction # ?/? 13 Fraction # ??/?? 14 Date M/D/YY 15 Date D-MMM-YY 16 Date D-MMM 17 Date MMM-YY 18 Time h:mm AM/PM 19 Time h:mm:ss AM/PM 20 Time h:mm 21 Time h:mm:ss 22 Date/Time M/D/YY h:mm 37 Account _(#,##0_);(#,##0) 38 Account _(#,##0_);[Red](#,##0) 39 Account _(#,##0.00_);(#,##0.00) 40 Account _(#,##0.00_);[Red](#,##0.00) 41 Currency _("$"* #,##0_);_("$"* (#,##0);_("$"* "-"_);_(@_) 42 Currency _(* #,##0_);_(* (#,##0);_(* "-"_);_(@_) 43 Currency _("$"* #,##0.00_);_("$"* (#,##0.00);_("$"* "-"??_);_(@_) 44 Currency _(* #,##0.00_);_(* (#,##0.00);_(* "-"??_);_(@_) 45 Time mm:ss 46 Time [h]:mm:ss 47 Time mm:ss.0 48 Scientific ##0.0E+0 49 Text @

class pyExcelerator.BIFFRecords.**NumberRecord**(*row*, *col*, *xf_index*, *number*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record represents a cell that contains an IEEE-754 floating-point value.

class pyExcelerator.BIFFRecords.**ObjectProtectRecord**(*objprotect*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the worksheet/workbook protection. It determines whether the objects of the current sheet are protected. Object protection is not active, if this record is omitted.

class pyExcelerator.BIFFRecords.**PaletteRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains the definition of all user-defined colours available for cell and object formatting.

Record PALETTE, BIFF3-BIFF8:

Offset Size Contents 0 2 Number of following colours (nm). Contains 16 in BIFF3-BIFF4 and 56 in BIFF5-BIFF8. 2 4*nm List of nm RGB colours

The following table shows how colour indexes are used in other records:

Colour index Resulting colour or internal list index 00H Built-in Black (R = 00H, G = 00H, B = 00H) 01H Built-in White (R = FFH, G = FFH, B = FFH) 02H Built-in Red (R = FFH, G = 00H, B = 00H) 03H Built-in Green (R = 00H, G = FFH, B = 00H) 04H Built-in Blue (R = 00H, G = 00H, B = FFH) 05H Built-in Yellow (R = FFH, G = FFH, B = 00H) 06H Built-in Magenta (R = FFH, G = 00H, B = FFH) 07H Built-in Cyan (R = 00H, G = FFH, B = FFH) 08H First user-defined colour from the PALETTE record (entry 0 from record colour list) .....

17H (BIFF3-BIFF4) Last user-defined colour from the PALETTE record (entry 15 or 55 from record colour list) 3FH (BIFF5-BIFF8)

18H (BIFF3-BIFF4) System window text colour for border lines (used in records XF, CF, and 40H (BIFF5-BIFF8) WINDOW2 (BIFF8 only))

19H (BIFF3-BIFF4) System window background colour for pattern background (used in records XF, and CF) 41H (BIFF5-BIFF8)

43H System face colour (dialogue background colour) 4DH System window text colour for chart border lines 4EH System window background colour for chart areas 4FH Automatic colour for chart border lines (seems to be always Black) 50H System ToolTip background colour (used in note objects) 51H System ToolTip text colour (used in note objects) 7FFFH System window text colour for fonts (used in records FONT, EFONT, and CF)

class pyExcelerator.BIFFRecords.**PanesRecord**(*px*, *py*, *first_row_bottom*, *first_col_right*, *active_pane*)

Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores the position of window panes. It is part of the Sheet View Settings Block. If the sheet does not contain any splits, this record will not occur. A sheet can be split in two different ways, with unfrozen panes or with frozen panes. A flag in the WINDOW2 record specifies, if the panes are frozen, which affects the contents of this record.

Record PANE, BIFF2-BIFF8: Offset Size Contents 0 2 Position of the vertical split

(px, 0 = No vertical split): Unfrozen pane: Width of the left pane(s) (in twips = 1/20 of a point) Frozen pane: Number of visible columns in left pane(s)

2 2 Position of the horizontal split (py, 0 = No horizontal split): Unfrozen pane: Height of the top pane(s) (in twips = 1/20 of a point) Frozen pane: Number of visible rows in top pane(s)

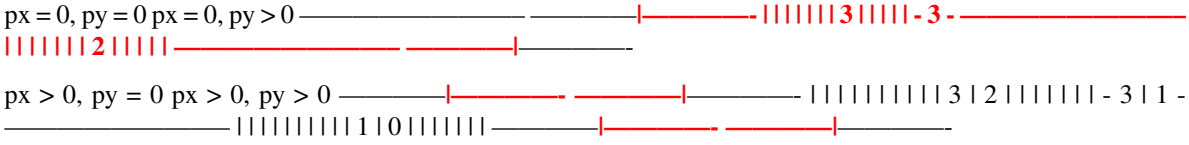
4 2 Index to first visible row in bottom pane(s)

6 2 Index to first visible column in right pane(s)

8 1 Identifier of pane with active cell cursor

[9] 1 Not used (BIFF5-BIFF8 only, not written in BIFF2-BIFF4)

If the panes are frozen, pane 0 is always active, regardless of the cursor position. The correct identifiers for all possible combinations of visible panes are shown in the following pictures.



class pyExcelerator.BIFFRecords.**PasswordRecord**(*passwd=''*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the worksheet/workbook protection. It stores a 16-bit hash value, calculated from the worksheet or workbook protection password.

passwd_hash(*plaintext*)
Based on the algorithm provided by Daniel Rentz of OpenOffice.

class pyExcelerator.BIFFRecords.**PrecisionRecord**(*use_real_values*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores if formulas use the real cell values for calculation or the values displayed on the screen. In BIFF2- BIFF4 this record is part of the Calculation Settings Block. In BIFF5-BIFF8 it is stored in the Workbook Globals Substream.

Record PRECISION, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = Use displayed values;

1 = Use real cell values

class pyExcelerator.BIFFRecords.**PrintGridLinesRecord**(*print_grid*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores if sheet grid lines will be printed.

Record PRINTGRIDLINES, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = Do not print sheet grid lines;

1 = Print sheet grid lines

class pyExcelerator.BIFFRecords.**PrintHeadersRecord**(*print_headers*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores if the row and column headers (the areas with row numbers and column letters) will be printed.

Record PRINTHEADERS, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = Do not print row/column headers;

1 = Print row/column headers

class pyExcelerator.BIFFRecords.**Prot4RevPassRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**Prot4RevRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**ProtectRecord**(*protect*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the worksheet/workbook protection. It specifies whether a worksheet or a workbook is protected against modification. Protection is not active, if this record is omitted.

class pyExcelerator.BIFFRecords.**QuicktipRecord**(*frow*, *lrow*, *fcol*, *lcol*, *hint*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains the cell range and text for a tool tip. It occurs in conjunction with the HLINK record for hyperlinks (6.53) in the Hyperlink Table (5.13). This record is only available in Excel 9.0 (Excel 2000) and later.

Record QUICKTIP, BIFF8:

Offset Size Contents 0 2 0800H (repeated record identifier) 2 8 Cell range address of all cells containing the tool tip (3.13.1) 10 var. Character array of the tool tip, no Unicode string header, always 16-bit characters, zero-

terminated

class pyExcelerator.BIFFRecords.**RKRecord**(*row*, *col*, *xf_index*, *rk_encoded*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record represents a cell that contains an RK value (encoded integer or floating-point value). If a floating-point value cannot be encoded to an RK value, a NUMBER record will be written.

class pyExcelerator.BIFFRecords.**RefModeRecord**(*ref_mode*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It stores which method is used to show cell addresses in formulas. The RC mode uses numeric indexes for rows and columns, i.e. R(1)C(-1), or R1C1:R2C2. The A1 mode uses characters for columns and numbers for rows, i.e. B1, or $A$1:$B$2.

Record REFMODE, BIFF2-BIFF8:

Offset Size Contents 0 2 0 = RC mode; 1 = A1 mode

class pyExcelerator.BIFFRecords.**RefreshAllRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**RightMarginRecord**(*margin*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains the right page margin of the current worksheet.

Offset Size Contents 0 8 Right page margin in inches

(IEEE 754 floating-point value, 64?bit double precision)

class pyExcelerator.BIFFRecords.**RowRecord**(*index*, *first_col*, *last_col*, *height_options*, *options*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains the properties of a single row in a sheet. Rows and cells in a sheet are divided into blocks of 32 rows.

Record ROW, BIFF3-BIFF8:

Offset Size Contents 0 2 Index of this row 2 2 Index to column of the first cell which is described by a cell record 4 2 Index to column of the last cell which is described by a cell record,

increased by 1

6 2 Bit Mask Contents 14-0 7FFFH Height of the row, in twips = 1/20 of a point 15 8000H 0 = Row has custom height; 1 = Row has default height

8 2 Not used 10 2 In BIFF3-BIFF4 this field contains a relative offset

to calculate stream position of the first cell record for this row. In BIFF5-BIFF8 this field is not used anymore, but the DBCELL record instead.

12 4 Option flags and default row formatting: Bit Mask Contents 2-0 00000007H Outline level of the row 4 00000010H 1 = Outline group starts or ends here (depending

on where the outline buttons are located, see WSBOOL record), and is collapsed

5 00000020H 1 = Row is hidden (manually, or by a filter or outline group) 6 00000040H 1 = Row height and default font height do not match 7 00000080H 1 = Row has explicit default format (fl) 8 00000100H Always 1 27-16 0FFF0000H If fl=1: Index to default XF record 28 10000000H 1 = Additional space above the row. 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.

29 20000000H 1 = Additional space below the row. 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.

class pyExcelerator.BIFFRecords.**SSTRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record contains a list of all strings used anywhere in the workbook. Each string occurs only once. The workbook uses indexes into the list to reference the strings.

Record SST, BIFF8: Offset Size Contents 0 4 Total number of strings in the workbook (see below) 4 4 Number of following strings (nm) 8 var. List of nm Unicode strings, 16-bit string length

The first field of the SST record counts the total occurrence of strings in the workbook. For instance, the string AAA is used 3 times and the string BBB is used 2 times. The first field contains 5 and the second field contains 2, followed by the two strings.

class pyExcelerator.BIFFRecords.**SaveRecalcRecord**(*recalc*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Calculation Settings Block. It contains the Recalculate before save option in Excel's calculation settings dialogue.

Record SAVERECALC, BIFF3-BIFF8:

Offset Size Contents 0 2 0 = Do not recalculate;

1 = Recalculate before saving the document

class pyExcelerator.BIFFRecords.**ScenProtectRecord**(*scenprotect*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the worksheet/workbook protection. It determines whether the scenarios of the current sheet are protected. Scenario protection is not active, if this record is omitted.

class pyExcelerator.BIFFRecords.**SetupPageRecord**(*paper*, *scaling*, *start_num*, *fit_width_to*, *fit_height_to*, *options*, *hres*, *vres*, *header_margin*, *footer_margin*, *num_copies*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It stores the page format settings of the current sheet. The pages may be scaled in percent or by using an absolute number of pages. This setting is located in the WSBOOL record. If pages are scaled in percent, the scaling factor in this record is used, otherwise the "Fit to pages" values. One of the "Fit to pages" values may be 0. In this case the sheet is scaled to fit only to the other value.

Record SETUP, BIFF5-BIFF8:

Offset Size Contents 0 2 Paper size (see below) 2 2 Scaling factor in percent 4 2 Start page number 6 2 Fit worksheet width to this number of pages

(0 = use as many as needed)

8 2 Fit worksheet height to this number of pages (0 = use as many as needed)

10 2 Option flags: Bit Mask Contents 0 0001H 0 = Print pages in columns

1 = Print pages in rows

1 0002H 0 = Landscape 1 = Portrait

2 0004H 1 = Paper size, scaling factor, paper orientation (portrait/landscape), print resolution and number of copies are not initialised

3 0008H 0 = Print coloured 1 = Print black and white

4 0010H 0 = Default print quality 1 = Draft quality

5 0020H 0 = Do not print cell notes 1 = Print cell notes

6 0040H 0 = Paper orientation setting is valid 1 = Paper orientation setting not initialised

7 0080H 0 = Automatic page numbers 1 = Use start page number

The following flags are valid for BIFF8 only: 9 0200H 0 = Print notes as displayed

1 = Print notes at end of sheet

11-10 0C00H 00 = Print errors as displayed 01 = Do not print errors 10 = Print errors as "–" 11 = Print errors as "#N/A!"

12 2 Print resolution in dpi 14 2 Vertical print resolution in dpi 16 8 Header margin (IEEE 754 floating-point value,

64bit double precision)

24 8 Footer margin (IEEE 754 floating-point value, 64bit double precision)

32 2 Number of copies to print

PAPER TYPES:

Index Paper type Paper size 0 Undefined 1 Letter 8 1/2" x 11" 2 Letter small 8 1/2" x 11" 3 Tabloid 11" x 17" 4 Ledger 17" x 11" 5 Legal 8 1/2" x 14" 6 Statement 5 1/2" x 8 1/2" 7 Executive 7 1/4" x 10 1/2" 8 A3 297mm x 420mm 9 A4 210mm x 297mm 10 A4 small 210mm x 297mm 11 A5 148mm x 210mm 12 B4 (JIS) 257mm x 364mm 13 B5 (JIS) 182mm x 257mm 14 Folio 8 1/2" x 13" 15 Quarto 215mm x 275mm 16 10x14 10" x 14" 17 11x17 11" x 17" 18 Note 8 1/2" x 11" 19 Envelope #9 3 7/8" x 8 7/8" 20 Envelope #10 4 1/8" x 9 1/2" 21 Envelope #11 4 1/2" x 10 3/8" 22 Envelope #12 4 3/4" x 11" 23 Envelope #14 5" x 11 1/2" 24 C 17" x 22" 25 D 22" x 34" 26 E 34" x 44" 27 Envelope DL 110mm x 220mm 28 Envelope C5 162mm x 229mm 29 Envelope C3 324mm x 458mm 30 Envelope C4 229mm x 324mm 31 Envelope C6 114mm x 162mm 32 Envelope C6/C5 114mm x 229mm 33 B4 (ISO) 250mm x 353mm 34 B5 (ISO) 176mm x 250mm 35 B6 (ISO) 125mm x 176mm 36 Envelope Italy 110mm x 230mm 37 Envelope Monarch 3 7/8" x 7 1/2" 38 63/4 Envelope 3 5/8" x 6 1/2" 39 US Standard Fanfold 14 7/8" x 11" 40 German Std. Fanfold 8 1/2" x 12" 41 German Legal Fanfold 8 1/2" x 13" 42 B4 (ISO) 250mm x 353mm 43 Japanese Postcard 100mm x 148mm 44 9x11 9" x 11" 45 10x11 10" x 11" 46 15x11 15" x 11" 47 Envelope Invite 220mm x 220mm 48 Undefined 49 Undefined 50 Letter Extra 9 1/2" x 12" 51 Legal Extra 9 1/2" x 15" 52 Tabloid Extra 11 11/16" x 18" 53 A4 Extra 235mm x 322mm 54 Letter Transverse 8 1/2" x 11" 55 A4 Transverse 210mm x 297mm 56 Letter Extra Transv. 9 1/2" x 12" 57 Super A/A4 227mm x 356mm 58 Super B/A3 305mm x 487mm 59 Letter Plus 8 1/2" x 12 11/16" 60 A4 Plus 210mm x 330mm 61 A5 Transverse 148mm x 210mm 62 B5 (JIS) Transverse 182mm x 257mm 63 A3 Extra 322mm x 445mm 64 A5 Extra 174mm x 235mm 65 B5 (ISO) Extra 201mm x 276mm 66 A2 420mm x 594mm 67 A3 Transverse 297mm x 420mm 68 A3 Extra Transverse 322mm x 445mm 69 Dbl. Japanese Postcard 200mm x 148mm 70 A6 105mm x 148mm 71 72 73 74 75 Letter Rotated 11" x 8 1/2" 76 A3 Rotated 420mm x 297mm 77 A4 Rotated 297mm x 210mm 78 A5 Rotated 210mm x 148mm 79 B4 (JIS) Rotated 364mm x 257mm 80 B5 (JIS) Rotated 257mm x 182mm 81 Japanese Postcard Rot. 148mm x 100mm 82 Dbl. Jap. Postcard Rot. 148mm x 200mm 83 A6 Rotated 148mm x 105mm 84 85 86 87 88 B6 (JIS) 128mm x 182mm 89 B6 (JIS) Rotated 182mm x 128mm 90 12x11 12" x 11"

class pyExcelerator.BIFFRecords.**SharedStringTable**
Bases: object

add_str(*s*)

get_biff_record()

str_index(*s*)

class pyExcelerator.BIFFRecords.**StringRecord**(*s*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Offset Size Contents 0 1 or 2 Length of the string (character count, ln) 1 or 2 1 Option flags:

Bit Mask Contents

01H

0 Character compression (ccompr):

02 = Compressed (8-bit characters) 12 = Uncompressed (16-bit characters)

04H

2 Asian phonetic settings (phonetic):

02 = Does not contain Asian phonetic settings 12 = Contains Asian phonetic settings

08H

3 Rich-Text settings (richtext): 02 = Does not contain Rich-Text settings 12 = Contains Rich-Text settings

[2 or 3] 2 (optional, only if richtext=1) Number of Rich-Text formatting runs (rt) [var.] 4 (optional, only if phonetic=1) Size of Asian phonetic settings block (in bytes, sz) var. ln or 2ln Character array (8-bit characters or 16-bit characters, dependent on ccompr) [var.] 4rt (optional, only if richtext=1) List of rt formatting runs (3.2)

sz

[var.] (optional, only if phonetic=1) Asian Phonetic Settings Block (3.4.2)

class pyExcelerator.BIFFRecords.**StyleRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

STYLE record for user-defined cell styles, BIFF3-BIFF8: Offset Size Contents 0 2 Bit Mask Contents

11-0 0FFFH Index to style XF record 15 8000H Always 0 for user-defined styles

2 var. BIFF2-BIFF7: Non-empty byte string, 8-bit string length BIFF8: Non-empty Unicode string, 16-bit string length

STYLE record for built-in cell styles, BIFF3-BIFF8: Offset Size Contents 0 2 Bit Mask Contents

11-0 0FFFH Index to style XF record 15 8000H Always 1 for built-in styles

2 1 Identifier of the built-in cell style: 00H = Normal 01H = RowLevel_lv (see next field) 02H = ColLevel_lv (see next field) 03H = Comma 04H = Currency 05H = Percent 06H = Comma [0] (BIFF4-BIFF8) 07H = Currency [0] (BIFF4-BIFF8) 08H = Hyperlink (BIFF8) 09H = Followed Hyperlink (BIFF8)

3 1 Level for RowLevel or ColLevel style (zero-based, lv), FFH otherwise

The RowLevel and ColLevel styles specify the formatting of subtotal cells in a specific outline level. The level is specified by the last field in the STYLE record. Valid values are 0-6 for the outline levels 1-7.

class pyExcelerator.BIFFRecords.**SupBookRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record mainly stores the URL of an external document and a list of sheet names inside this document. Furthermore it is used to store DDE and OLE object links, or to indicate an internal 3D reference or an add-in function. See 5.10.3 for details about external references in BIFF8.

class pyExcelerator.BIFFRecords.**TabIDRecord**(*sheetcount*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.BIFFRecords.**TopMarginRecord**(*margin*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains the top page margin of the current worksheet.

Offset Size Contents 0 8 Top page margin in inches

(IEEE 754 floating-point value, 64?bit double precision)

class pyExcelerator.BIFFRecords.**UseSelfsRecord**
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record specifies if the formulas in the workbook can use natural language formulas. This type of formula can refer to cells by its content or the content of the column or row header cell.

Record USESELFS, BIFF8:

Offset Size Contents 0 2 0 = Do not use natural language formulas

1 = Use natural language formulas

class pyExcelerator.BIFFRecords.**VCenterRecord**(*is_vert_center*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It specifies if the sheet is centred vertically when printed.

Record VCENTER, BIFF3-BIFF8:

Offset Size Contents 0 2 0 = Print sheet aligned at top page border

1 = Print sheet vertically centred

class pyExcelerator.BIFFRecords.**VerticalPageBreaksRecord**(*breaks_list*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the Page Settings Block. It contains all vertical manual page breaks.

Record VERTICALPAGEBREAKS, BIFF8: Offset Size Contents 0 2 Number of following column index structures (nm) 2 6nm List of nm column index structures. Each column index

structure contains: Offset Size Contents 0 2 Index to first column following the page

break

2 2 Index to first row of this page break 4 2 Index to last row of this page break

The column indexes in the lists must be ordered ascending. If in BIFF8 a column contains several page breaks, they must be ordered ascending by start row index.

class pyExcelerator.BIFFRecords.**WSBoolRecord**(*options*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record stores a 16 bit value with Boolean options for the current sheet. From BIFF5 on the "Save external linked values" option is moved to the record BOOKBOOL.

Option flags of record WSBOOL, BIFF3-BIFF8:

Bit Mask Contents 0 0001H 0 = Do not show automatic page breaks

1 = Show automatic page breaks

4 0010H 0 = Standard sheet 1 = Dialogue sheet (BIFF5-BIFF8)

5 0020H 0 = No automatic styles in outlines 1 = Apply automatic styles to outlines

6 0040H 0 = Outline buttons above outline group 1 = Outline buttons below outline group

7 0080H 0 = Outline buttons left of outline group 1 = Outline buttons right of outline group

8 0100H 0 = Scale printout in percent 1 = Fit printout to number of pages

9 0200H 0 = Save external linked values (BIFF3?BIFF4 only) 1 = Do not save external linked values (BIFF3?BIFF4 only)

10 0400H 0 = Do not show row outline symbols 1 = Show row outline symbols

11 0800H 0 = Do not show column outline symbols 1 = Show column outline symbols

13-12 3000H These flags specify the arrangement of windows. They are stored in BIFF4 only. 00 = Arrange windows tiled 01 = Arrange windows horizontal 10 = Arrange windows vertical112 = Arrange windows cascaded

The following flags are valid for BIFF4-BIFF8 only: 14 4000H 0 = Standard expression evaluation

1 = Alternative expression evaluation

15 8000H 0 = Standard formula entries 1 = Alternative formula entries

class pyExcelerator.BIFFRecords.**Window1Record**(*hpos_twips*, *vpos_twips*, *width_twips*, *height_twips*, *flags*, *active_sheet*, *first_tab_index*, *selected_tabs*, *tab_width*)

Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Offset Size Contents 0 2 Horizontal position of the document window (in twips = 1/20 of a point) 2 2 Vertical position of the document window (in twips = 1/20 of a point) 4 2 Width of the document window (in twips = 1/20 of a point) 6 2 Height of the document window (in twips = 1/20 of a point) 8 2 Option flags:

Bits Mask Contents 0 0001H 0 = Window is visible 1 = Window is hidden 1 0002H 0 = Window is open 1 = Window is minimised 3 0008H 0 = Horizontal scroll bar hidden 1 = Horizontal scroll bar visible 4 0010H 0 = Vertical scroll bar hidden 1 = Vertical scroll bar visible 5 0020H 0 = Worksheet tab bar hidden 1 = Worksheet tab bar visible

10 2 Index to active (displayed) worksheet 12 2 Index of first visible tab in the worksheet tab bar 14 2 Number of selected worksheets (highlighted in the worksheet tab bar) 16 2 Width of worksheet tab bar (in 1/1000 of window width). The remaining space is used by the

horizontal scrollbar.

class pyExcelerator.BIFFRecords.**Window2Record**(*options*, *first_visible_row*, *first_visible_col*, *grid_colour*, *preview_magn*, *normal_magn*)

Bases: *pyExcelerator.BIFFRecords.BiffRecord*

Record WINDOW2, BIFF8:

Offset Size Contents 0 2 Option flags (see below) 2 2 Index to first visible row 4 2 Index to first visible column 6 2 Colour index of grid line colour. Note that in BIFF2-BIFF7 an RGB colour is

written instead.

8 2 Not used 10 2 Cached magnification factor in page break preview (in percent); 0 = Default (60%) 12 2 Cached magnification factor in normal view (in percent); 0 = Default (100%) 14 4 Not used

In BIFF8 this record stores used magnification factors for page break preview and normal view. These values are used to restore the magnification, when the view is changed. The real magnification of the currently active view is stored in the SCL record. The type of the active view is stored in the option flags field (see below).

0 0001H 0 = Show formula results 1 = Show formulas 1 0002H 0 = Do not show grid lines 1 = Show grid lines 2 0004H 0 = Do not show sheet headers 1 = Show sheet headers 3 0008H 0 = Panes are not frozen 1 = Panes are frozen (freeze) 4 0010H 0 = Show zero values as empty cells 1 = Show zero values 5 0020H 0 = Manual grid line colour 1 = Automatic grid line colour 6 0040H 0 = Columns from left to right 1 = Columns from right to left 7 0080H 0 = Do not show outline symbols 1 = Show outline symbols 8 0100H 0 = Keep splits if pane freeze is removed 1 = Remove splits if pane freeze is removed 9 0200H 0 = Sheet not selected 1 = Sheet selected (BIFF5-BIFF8)

10 0400H 0 = Sheet not visible 1 = Sheet visible (BIFF5-BIFF8) 11 0800H 0 = Show in normal view 1 = Show in page break preview (BIFF8)

The freeze flag specifies, if a following PANE record describes unfrozen or frozen panes.

class pyExcelerator.BIFFRecords.**WindowProtectRecord**(*wndprotect*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the worksheet/workbook protection. It determines whether the window configuration of this document is protected. Window protection is not active, if this record is omitted.

class pyExcelerator.BIFFRecords.**WriteAccessRecord**(*owner*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

This record is part of the file protection. It contains the name of the user that has saved the file. The user name is always stored as an equal-sized string. All unused characters after the name are filled with space characters. It is not required to write the mentioned string length. Every other length will be accepted too.

class pyExcelerator.BIFFRecords.**XFRecord**(*xf*, *xftype='cell'*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

XF_TYPE_PROT XF Type and Cell Protection (3 Bits), BIFF3-BIFF8 These 3 bits are part of a specific data byte. Bit Mask Contents 0 01H 1 = Cell is locked 1 02H 1 = Formula is hidden 2 04H 0 = Cell XF; 1 = Style XF

XF_USED_ATTRIB Attributes Used from Parent Style XF (6 Bits), BIFF3-BIFF8 Each bit describes the validity of a specific group of attributes. In cell XFs a cleared bit means the attributes of the parent style XF are used (but only if the attributes are valid there), a set bit means the attributes of this XF are used. In style XFs a cleared bit means the attribute setting is valid, a set bit means the attribute should be ignored. Bit Mask Contents 0 01H Flag for number format 1 02H Flag for font 2 04H Flag for horizontal and vertical alignment, text wrap, indentation, orientation, rotation, and

text direction

3 08H Flag for border lines 4 10H Flag for background area style 5 20H Flag for cell protection (cell locked and formula hidden)

XF_HOR_ALIGN Horizontal Alignment (3 Bits), BIFF2-BIFF8 The horizontal alignment consists of 3 bits and is part of a specific data byte. Value Horizontal alignment 00H General 01H Left 02H Centred 03H Right 04H Filled 05H Justified (BIFF4-BIFF8X) 06H Centred across selection (BIFF4-BIFF8X) 07H Distributed (BIFF8X)

XF_VERT_ALIGN Vertical Alignment (2 or 3 Bits), BIFF4-BIFF8 The vertical alignment consists of 2 bits (BIFF4) or 3 bits (BIFF5-BIFF8) and is part of a specific data byte. Vertical alignment is not available in BIFF2 and BIFF3. Value Vertical alignment 00H Top 01H Centred 02H Bottom 03H Justified (BIFF5-BIFF8X) 04H Distributed (BIFF8X)

XF_ORIENTATION Text Orientation (2 Bits), BIFF4-BIFF7 In the BIFF versions BIFF4-BIFF7, text can be rotated in steps of 90 degrees or stacked. The orientation mode consists of 2 bits and is part of a specific data byte. In BIFF8 a rotation angle occurs instead of these flags. Value Text orientation 00H Not rotated 01H Letters are stacked top-to-bottom, but not rotated 02H Text is rotated 90 degrees counterclockwise 03H Text is rotated 90 degrees clockwise

XF_ROTATION Text Rotation Angle (1 Byte), BIFF8 Value Text rotation 0 Not rotated 1-90 1 to 90 degrees counterclockwise 91-180 1 to 90 degrees clockwise 255 Letters are stacked top-to-bottom, but not rotated

XF_BORDER_34 Cell Border Style (4 Bytes), BIFF3-BIFF4 Cell borders contain a line style and a line colour for each line of the border. Bit Mask Contents 2-0 00000007H Top line style 7-3 000000F8H Colour index for top line colour 10-8 00000700H Left line style 15-11 0000F800H Colour index for left line colour 18-16 00070000H Bottom line style 23-19 00F80000H Colour index for bottom line colour 26-24 07000000H Right line style 31-27 F8000000H Colour index for right line colour

XF_AREA_34 Cell Background Area Style (2 Bytes), BIFF3-BIFF4 A cell background area style contains an area pattern and a foreground and background colour. Bit Mask Contents 5-0 003FH Fill pattern 10-6 07C0H Colour index for pattern colour 15-11 F800H Colour index for pattern background

Record XF, BIFF8: Offset Size Contents 0 2 Index to FONT record 2 2 Index to FORMAT record 4 2 Bit Mask Contents

2-0 0007H XF_TYPE_PROT . XF type, cell protection (see above) 15-4 FFF0H Index to parent style XF (always FFFH in style XFs)

6 1 Bit Mask Contents 2-0 07H XF_HOR_ALIGN . Horizontal alignment (see above) 3 08H 1 = Text is wrapped at right border 6-4 70H XF_VERT_ALIGN . Vertical alignment (see above)

7 1 XF_ROTATION: Text rotation angle (see above) 8 1 Bit Mask Contents

3-0 0FH Indent level 4 10H 1 = Shrink content to fit into cell 5 merge 7-6 C0H Text direction (BIFF8X only)

00b = According to context 01b = Left-to-right 10b = Right-to-left

9 1 Bit Mask Contents 7-2 FCH XF_USED_ATTRIB . Used attributes (see above)

10 4 Cell border lines and background area: Bit Mask Contents 3-0 0000000FH Left line style 7-4 000000F0H Right line style 11-8 00000F00H Top line style 15-12 0000F000H Bottom line style 22-16 007F0000H Colour index for left line colour 29-23 3F800000H Colour index for right line colour 30 40000000H 1 = Diagonal line from top left to right bottom 31 80000000H 1 = Diagonal line from bottom left to right top

14 4 Bit Mask Contents 6-0 0000007FH Colour index for top line colour 13-7 00003F80H Colour index for bottom line colour 20-14 001FC000H Colour index for diagonal line colour 24-21 01E00000H Diagonal line style 31-26 FC000000H Fill pattern

18 2 Bit Mask Contents 6-0 007FH Colour index for pattern colour 13-7 3F80H Colour index for pattern background

1.3 `Bitmap` Module

class pyExcelerator.Bitmap.**ImDataBmpRecord**(*filename*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

class pyExcelerator.Bitmap.**ObjBmpRecord**(*row*, *col*, *sheet*, *im_data_bmp*, *x*, *y*, *scale_x*, *scale_y*)
Bases: *pyExcelerator.BIFFRecords.BiffRecord*

1.4 Cell Module

class pyExcelerator.Cell.**BlankCell**(*parent*, *idx*, *xf_idx*)
Bases: object

get_biff_data()

class pyExcelerator.Cell.**FormulaCell**(*parent*, *idx*, *xf_idx*, *frmla*)
Bases: object

get_biff_data()

get_result()

result

set_result(*value*)

class pyExcelerator.Cell.**MulBlankCell**(*parent*, *col1*, *col2*, *xf_idx*)
Bases: object

get_biff_data()

class pyExcelerator.Cell.**MulNumberCell**(*parent*, *idx*, *xf_idx*, *sst_idx*)
Bases: object

get_biff_data()

class pyExcelerator.Cell.**NumberCell**(*parent*, *idx*, *xf_idx*, *number*)
Bases: object

get_biff_data()

class pyExcelerator.Cell.**StrCell**(*parent*, *idx*, *xf_idx*, *sst_idx*)
Bases: object

get_biff_data()

1.5 Column Module

class pyExcelerator.Column.**Column**(**args*, ***kwds*)
Bases: object

get_biff_record()

get_index()

set_style(**args*, ***kwds*)

1.6 CompoundDoc Module

class pyExcelerator.CompoundDoc.**Reader**(*filename*, *dump=False*)

get_stream_data(*data*, *SAT*, *start_sid*, *sect_size*)

class pyExcelerator.CompoundDoc.**XlsDoc**

MIN_LIMIT = 4096

SECTOR_SIZE = 512

SID_END_OF_CHAIN = -2

SID_FREE_SECTOR = -1

SID_USED_BY_MSAT = -4

SID_USED_BY_SAT = -3

save(*filename*, *stream*)

pyExcelerator.CompoundDoc.**print_bin_data**(*data*)

1.7 Deco Module

pyExcelerator.Deco.**accepts**(**types*)

pyExcelerator.Deco.**returns**(*rtype*)

1.8 ExcelFormula Module

class pyExcelerator.ExcelFormula.**ErrorCode**(*s*)
Bases: object

error_msg = {u'#NUM!': 36, u'#DIV/0!': 7, u'#N/A!': 42, u'#REF!': 23, u'#VALUE!': 15, u'#NAME?': 29, u'#NULL!'

i = (29, u'#NAME?')

int()

class pyExcelerator.ExcelFormula.**Formula**(*s*, *default=None*, *opts=None*)
Bases: object

default

get_default()

get_opts()

opts

rpn()
Offset Size Contents 0 2 Size of the following formula data (sz) 2 sz Formula data (RPN token array) [2+sz] var. (optional) Additional data for specific tokens

set_default(*val*)

set_opts(**args*, ***kwds*)

text()

1.9 ExcelFormulaLexer Module

class pyExcelerator.ExcelFormulaLexer.**Lexer**(*text*)

Bases: *pyExcelerator.antlr.TokenStream*

curr_ch()

isEOF()

is_whitespace()

match_pattern(*pattern*, *toktype*)

nextToken()

next_ch(*n=1*)

rest()

1.10 ExcelFormulaParser Module

class pyExcelerator.ExcelFormulaParser.**Parser**(**args*, ***kwargs*)

Bases: *pyExcelerator.antlr.LLkParser*

expr(*arg_type*)

expr_list(*arg_type_list*, *min_argc*, *max_argc*)

formula()

prec0_expr(*arg_type*)

prec1_expr(*arg_type*)

prec2_expr(*arg_type*)

prec3_expr(*arg_type*)

prec4_expr(*arg_type*)

prec5_expr(*arg_type*)

primary(*arg_type*)

pyExcelerator.ExcelFormulaParser.**mk_tokenSet_0**()

1.11 ExcelMagic Module

1.12 Formatting Module

The XF record is able to store explicit cell formatting attributes or the attributes of a cell style. Explicit formatting includes the reference to a cell style XF record. This allows to extend a defined cell style with some explicit attributes. The formatting attributes are divided into 6 groups:

1.12.1 Group Attributes

Number format Number format index (index to FORMAT record) Font Font index (index to FONT record) Alignment Horizontal and vertical alignment, text wrap, indentation,

orientation/rotation, text direction

Border Border line styles and colours Background Background area style and colours Protection Cell locked, formula hidden

For each group a flag in the cell XF record specifies whether to use the attributes contained in that XF record or in the referenced style XF record. In style XF records, these flags specify whether the attributes will overwrite explicit cell formatting when the style is applied to a cell. Changing a cell style (without applying this style to a cell) will change all cells which already use that style and do not contain explicit cell attributes for the changed style attributes. If a cell XF record does not contain explicit attributes in a group (if the attribute group flag is not set), it repeats the attributes of its style XF record.

class pyExcelerator.Formatting.**Alignment**

Bases: *pyExcelerator.Formatting.CopyableObject*

DIRECTION_GENERAL = 0

DIRECTION_LR = 1

DIRECTION_RL = 2

HORZ_CENTER = 2

HORZ_CENTER_ACROSS_SEL = 6

HORZ_DISTRIBUTED = 7

HORZ_FILLED = 4

HORZ_GENERAL = 0

HORZ_JUSTIFIED = 5

HORZ_LEFT = 1

HORZ_RIGHT = 3

NOT_SHRINK_TO_FIT = 0

NOT_WRAP_AT_RIGHT = 0

ORIENTATION_90_CC = 2

ORIENTATION_90_CW = 3

ORIENTATION_NOT_ROTATED = 0

ORIENTATION_STACKED = 1

ROTATION_0_ANGLE = 0

ROTATION_STACKED = 255

SHRINK_TO_FIT = 1

VERT_BOTTOM = 2

VERT_CENTER = 1

VERT_DISIRIBUTED = 4

VERT_JUSTIFIED = 3

VERT_TOP = 0

WRAP_AT_RIGHT = 1

class pyExcelerator.Formatting.**Borders**
Bases: *pyExcelerator.Formatting.CopyableObject*

DASHED = 3

DOTTED = 4

DOUBLE = 6

HAIR = 7

MEDIUM = 2

MEDIUM_DASHED = 8

MEDIUM_DASH_DOTTED = 10

MEDIUM_DASH_DOT_DOTTED = 12

NEED_DIAG1 = 1

NEED_DIAG2 = 1

NO_LINE = 0

NO_NEED_DIAG1 = 0

NO_NEED_DIAG2 = 0

SLANTED_MEDIUM_DASH_DOTTED = 13

THICK = 5

THIN = 1

THIN_DASH_DOTTED = 9

THIN_DASH_DOT_DOTTED = 11

class pyExcelerator.Formatting.**CopyableObject**
Bases: object

copy()

class pyExcelerator.Formatting.**Font**
Bases: *pyExcelerator.Formatting.CopyableObject*

CHARSET_ANSI_ARABIC = 178

CHARSET_ANSI_BALTIC = 186

CHARSET_ANSI_CHINESE_BIG5 = 136

CHARSET_ANSI_CHINESE_GBK = 134

CHARSET_ANSI_CYRILLIC = 204

CHARSET_ANSI_GREEK = 161

CHARSET_ANSI_HEBREW = 177

CHARSET_ANSI_JAP_SHIFT_JIS = 128

CHARSET_ANSI_KOR_HANGUL = 129

CHARSET_ANSI_KOR_JOHAB = 130

CHARSET_ANSI_LATIN = 0

CHARSET_ANSI_LATIN_II = 238

CHARSET_ANSI_THAI = 222

CHARSET_ANSI_TURKISH = 162

CHARSET_ANSI_VIETNAMESE = 163

CHARSET_APPLE_ROMAN = 77

CHARSET_OEM_LATIN_I = 255

CHARSET_SYMBOL = 2

CHARSET_SYS_DEFAULT = 1

ESCAPEMENT_NONE = 0

ESCAPEMENT_SUBSCRIPT = 2

ESCAPEMENT_SUPERSCRIPT = 1

FAMILY_DECORARTIVE = 5

FAMILY_MODERN = 3

FAMILY_NONE = 0

FAMILY_ROMAN = 1

FAMILY_SCRIPT = 4

FAMILY_SWISS = 2

UNDERLINE_DOUBLE = 2

UNDERLINE_DOUBLE_ACC = 34

UNDERLINE_NONE = 0

UNDERLINE_SINGLE = 1

UNDERLINE_SINGLE_ACC = 33

get_biff_record()

class pyExcelerator.Formatting.**Pattern**

Bases: *pyExcelerator.Formatting.CopyableObject*

NO_PATTERN = 0

SOLID_PATTERN = 1

get_pattern_back_colour()

get_pattern_fore_colour()

pattern_back_colour

pattern_fore_colour

set_pattern_back_colour(*c*)

set_pattern_fore_colour(*c*)

class pyExcelerator.Formatting.**Protection**

Bases: *pyExcelerator.Formatting.CopyableObject*

pyExcelerator.Formatting.**get_colour_val**(*c*)

1.13 ImportXLS Module

pyExcelerator.ImportXLS.**parse_xls**(*filename*, *encoding=None*)

1.14 Row Module

class pyExcelerator.Row.**Row**(*index*, *parent_sheet*)

Bases: object

collapse

frmla_opts

get_cells_biff_data()

get_cells_count()

get_frmla_opts()

get_height()

get_height_in_pixels()

get_index()

get_max_col()

get_min_col()

get_row_biff_data()

get_str_count()

get_xf_index()

height

hidden

level

set_frmla_opts(**args*, ***kwds*)

set_height(*h*)

set_style(**args*, ***kwds*)

space_above

space_below

write(**args*, ***kwds*)

write_blanks(**args*, ***kwds*)

1.15 Style Module

class pyExcelerator.Style.**StyleCollection**

Bases: object

add(*style*)

get_biff_data()

class pyExcelerator.Style.**XFStyle**
Bases: object

1.16 UnicodeUtils Module

From BIFF8 on, strings are always stored using UTF-16LE text encoding. The character array is a sequence of 16-bit values4. Additionally it is possible to use a compressed format, which omits the high bytes of all characters, if they are all zero.

The following tables describe the standard format of the entire string, but in many records the strings differ from this format. This will be mentioned separately. It is possible (but not required) to store Rich-Text formatting information and Asian phonetic information inside a Unicode string. This results in four different ways to store a string. The character array is not zero-terminated.

The string consists of the character count (as usual an 8-bit value or a 16-bit value), option flags, the character array and optional formatting information. If the string is empty, sometimes the option flags field will not occur. This is mentioned at the respective place.

Offset Size Contents 0 1 or 2 Length of the string (character count, ln) 1 or 2 1 Option flags:

Bit Mask Contents 0 01H Character compression (ccompr):

0 = Compressed (8-bit characters) 1 = Uncompressed (16-bit characters)

2 04H Asian phonetic settings (phonetic): 0 = Does not contain Asian phonetic settings 1 = Contains Asian phonetic settings

3 08H Rich-Text settings (richtext): 0 = Does not contain Rich-Text settings 1 = Contains Rich-Text settings

[2 or 3] 2 (optional, only if richtext=1) Number of Rich-Text formatting runs (rt) [var.] 4 (optional, only if phonetic=1) Size of Asian phonetic settings block (in bytes, sz) var. ln or

2·ln Character array (8-bit characters or 16-bit characters, dependent on ccompr)

[var.] 4·rt (optional, only if richtext=1) List of rt formatting runs [var.] sz (optional, only if phonetic=1) Asian Phonetic Settings Block

pyExcelerator.UnicodeUtils.**u2bytes**(*ustr*)

pyExcelerator.UnicodeUtils.**u2ints**(*ustr*)

pyExcelerator.UnicodeUtils.**upack1**(*_str*)

pyExcelerator.UnicodeUtils.**upack2**(*_str*)

1.17 Utils Module

pyXLWriter.utilites

Utilities for work with reference to cells

pyExcelerator.Utils.**cell_to_packed_rowcol**(*cell*)
pack row and column into the required 4 byte format

pyExcelerator.Utils.**cell_to_rowcol**(*cell*)
Convert an Excel cell reference string in A1 notation to numeric row/col notation.

Returns: row, col, row_abs, col_abs

pyExcelerator.Utils.**cell_to_rowcol2**(*cell*)
Convert an Excel cell reference string in A1 notation to numeric row/col notation.

Returns: row, col

pyExcelerator.Utils.**cellrange_to_rowcol_pair**(*cellrange*)
Convert cell range string in A1 notation to numeric row/col pair.

Returns: row1, col1, row2, col2

pyExcelerator.Utils.**col_by_name**(*colname*)

pyExcelerator.Utils.**pts_to_px**(*pts*)

pyExcelerator.Utils.**px_to_pts**(*px*)

pyExcelerator.Utils.**px_to_tw**(*px*)

pyExcelerator.Utils.**rowcol_to_cell**(*row*, *col*, *row_abs=False*, *col_abs=False*)
Convert numeric row/col notation to an Excel cell reference string in A1 notation.

pyExcelerator.Utils.**tw_to_px**(*tw*)

1.18 Workbook Module

Record Order in BIFF8

Workbook Globals Substream

BOF Type = workbook globals Interface Header MMS Interface End WRITEACCESS CODEPAGE DSF TABID FNGROUPCOUNT Workbook Protection Block

WINDOWPROTECT PROTECT PASSWORD PROT4REV PROT4REVPASS

BACKUP HIDEOBJ WINDOW1 DATEMODE PRECISION REFRESHALL BOOKBOOL FONT + FORMAT * XF + STYLE +

? PALETTE USESELFS

BOUNDSHEET +

COUNTRY

? Link Table SST ExtSST EOF

class pyExcelerator.Workbook.**Workbook**(**args*, ***kwds*)
Bases: object

active_sheet

add_sheet(**args*, ***kwds*)

add_str(**args*, ***kwds*)

add_style(**args*, ***kwds*)

backup_on_save

country_code

dates_1904

default_style

get_active_sheet()

get_backup_on_save()

get_biff_data()

get_country_code()

get_dates_1904()

get_default_style()

get_height()

get_hpos()

get_hscroll_visible()

get_obj_protect()

get_owner()

get_protect()

get_sheet(**args*, ***kwds*)

get_tab_width()

get_tabs_visible()

get_use_cell_values()

get_vpos()

get_vscroll_visible()

get_width()

get_wnd_mini()

get_wnd_protect()

get_wnd_visible()

height

hpos

hscroll_visible

macros = {'Print_Area': 6, 'Data_Form': 9, 'Auto_Deactivate': 11, 'Database': 4, 'Consolidate_Area': 0, 'Criteria': 5, '

obj_protect

owner

print_area(**args*, ***kwds*)

protect

save(*filename*)

set_active_sheet(**args*, ***kwds*)

set_backup_on_save(**args*, ***kwds*)

set_country_code(**args*, ***kwds*)

set_dates_1904(**args*, ***kwds*)

set_height(**args*, ***kwds*)

set_hpos(**args*, ***kwds*)

set_hscroll_visible(**args*, ***kwds*)

set_obj_protect(**args*, ***kwds*)

set_owner(**args*, ***kwds*)

set_protect(**args*, ***kwds*)

set_tab_width(**args*, ***kwds*)

set_tabs_visible(**args*, ***kwds*)

set_use_cell_values(**args*, ***kwds*)

set_vpos(**args*, ***kwds*)

set_vscroll_visible(**args*, ***kwds*)

set_width(**args*, ***kwds*)

set_wnd_mini(**args*, ***kwds*)

set_wnd_protect(**args*, ***kwds*)

set_wnd_visible(**args*, ***kwds*)

str_index(**args*, ***kwds*)

tab_width

tabs_visible

use_cell_values

vpos

vscroll_visible

width

wnd_mini

wnd_protect

wnd_visible

1.19 Worksheet Module

BOF UNCALCED INDEX Calculation Settings Block PRINTHEADERS PRINTGRIDLINES GRIDSET GUTS DEFAULTROWHEIGHT WSBOOL Page Settings Block Worksheet Protection Block DEFCOLWIDTH COLINFO SORT DIMENSIONS Row Blocks WINDOW2 SCL PANE SELECTION STANDARDWIDTH MERGEDCELLS LABELRANGES PHONETIC Conditional Formatting Table Hyperlink Table Data Validity Table SHEETLAYOUT (BIFF8X only) SHEETPROTECTION (BIFF8X only) RANGEPROTECTION (BIFF8X only) EOF

class pyExcelerator.Worksheet.**Worksheet**(**args*, ***kwds*)

Bases: object

RC_ref_mode

class Workbook(**args*, ***kwds*)

Bases: object

active_sheet

add_sheet(**args*, ***kwds*)

add_str(**args*, ***kwds*)

add_style(**args*, ***kwds*)

backup_on_save

country_code

dates_1904

default_style

get_active_sheet()

get_backup_on_save()

get_biff_data()

get_country_code()

get_dates_1904()

get_default_style()

get_height()

get_hpos()

get_hscroll_visible()

get_obj_protect()

get_owner()

get_protect()

get_sheet(**args*, ***kwds*)

get_tab_width()

get_tabs_visible()

get_use_cell_values()

get_vpos()

get_vscroll_visible()

get_width()

get_wnd_mini()

get_wnd_protect()

get_wnd_visible()

height

hpos

hscroll_visible

macros = {'Print_Area': 6, 'Data_Form': 9, 'Auto_Deactivate': 11, 'Database': 4, 'Consolidate_Area': 0, 'Criteria'

obj_protect

owner

print_area(**args*, ***kwds*)

protect

save(*filename*)

set_active_sheet(**args*, ***kwds*)

set_backup_on_save(**args*, ***kwds*)

set_country_code(**args*, ***kwds*)

set_dates_1904(**args*, ***kwds*)

set_height(**args*, ***kwds*)

set_hpos(**args*, ***kwds*)

set_hscroll_visible(**args*, ***kwds*)

set_obj_protect(**args*, ***kwds*)

set_owner(**args*, ***kwds*)

set_protect(**args*, ***kwds*)

set_tab_width(**args*, ***kwds*)

set_tabs_visible(**args*, ***kwds*)

set_use_cell_values(**args*, ***kwds*)

set_vpos(**args*, ***kwds*)

set_vscroll_visible(**args*, ***kwds*)

set_width(**args*, ***kwds*)

set_wnd_mini(**args*, ***kwds*)

set_wnd_protect(**args*, ***kwds*)

set_wnd_visible(**args*, ***kwds*)

str_index(**args*, ***kwds*)

tab_width

tabs_visible

use_cell_values

vpos

vscroll_visible

width

wnd_mini

wnd_protect

wnd_visible

alt_expr_eval

alt_formula_entries

auto_colour_grid

auto_style_outline

bmp_rec

bottom_margin

calc_count

calc_mode

col(*indx*)

col_default_width

col_width(*col*)

cols

cols_right_to_left

copies_num

delta

dialogue_sheet

first_visible_col

first_visible_row

fit_height_to_pages

fit_num_pages

fit_width_to_pages

footer_margin

footer_str

frmla_opts

get_RC_ref_mode()

get_alt_expr_eval()

get_alt_formula_entries()

get_auto_colour_grid()

get_auto_style_outline()

get_biff_data()

get_bmp_rec()

get_bottom_margin()

get_calc_count()

get_calc_mode()

get_col_default_width()

get_cols()

get_cols_right_to_left()

get_copies_num()

get_delta()

get_dialogue_sheet()

get_first_visible_col()

get_first_visible_row()

get_fit_height_to_pages()

get_fit_num_pages()

get_fit_width_to_pages()

get_footer_margin()

get_footer_str()

get_frmla_opts()

get_grid_colour()

get_header_margin()

get_header_str()

get_hidden()

get_horz_page_breaks()

get_horz_split_first_visible()

get_horz_split_pos()

get_iterations_on()

get_labels_count()

get_left_margin()

get_merged_ranges()

get_name()

get_normal_magn()

get_obj_protect()

get_outline_below()

get_outline_right()

get_page_preview()

get_panes_frozen()

get_paper_size_code()

get_parent()

get_password()

get_portrait()

get_preview_magn()

get_print_centered_horz()

get_print_centered_vert()

get_print_colour()

get_print_draft()

get_print_grid()

get_print_headers()

get_print_hres()

get_print_in_rows()

get_print_notes()

get_print_notes_at_end()

get_print_omit_errors()

get_print_scaling()

get_print_vres()

get_protect()

get_remove_splits()

get_right_margin()

get_row_default_height()

get_rows()

get_save_recalc()

get_scen_protect()

get_selected()

get_show_auto_page_breaks()

get_show_col_outline()

get_show_empty_as_zero()

get_show_formulas()

get_show_grid()

get_show_headers()

get_show_outline()

get_show_row_outline()

get_start_page_number()

get_top_margin()

get_vert_page_breaks()

get_vert_split_first_visible()

get_vert_split_pos()

get_wnd_protect()

grid_colour

header_margin

header_str

hidden

horz_page_breaks

horz_split_first_visible

horz_split_pos

insert_bitmap (*filename*, *row*, *col*, *x=0*, *y=0*, *scale_x=1*, *scale_y=1*)

iterations_on

left_margin

merge (*r1*, *r2*, *c1*, *c2*, *style=<pyExcelerator.Style.XFStyle object>*)

merged_ranges

name

normal_magn

obj_protect

outline_below

outline_right

page_preview

panes_frozen

paper_size_code

parent

password

portrait

preview_magn

print_area (*rstart*, *rend*, *cstart*, *cend*)

print_centered_horz

print_centered_vert

print_colour

print_draft

print_grid

print_headers

print_hres

print_in_rows

print_notes

print_notes_at_end

print_omit_errors

print_scaling

print_vres

protect

remove_splits

right_margin

row (*indx*)

row_default_height

row_height(*row*)

rows

save_recalc

scen_protect

selected

set_RC_ref_mode(**args*, ***kwds*)

set_alt_expr_eval(**args*, ***kwds*)

set_alt_formula_entries(**args*, ***kwds*)

set_auto_colour_grid(**args*, ***kwds*)

set_auto_style_outline(**args*, ***kwds*)

set_bottom_margin(**args*, ***kwds*)

set_calc_count(**args*, ***kwds*)

set_calc_mode(**args*, ***kwds*)

set_col_default_width(**args*, ***kwds*)

set_cols_right_to_left(**args*, ***kwds*)

set_column(*col_range*, *width=None*, *format=<pyExcelerator.Style.XFStyle object>*)

set_columns(*col_range*, *width=None*, *format=<pyExcelerator.Style.XFStyle object>*)

set_copies_num(**args*, ***kwds*)

set_delta(**args*, ***kwds*)

set_dialogue_sheet(**args*, ***kwds*)

set_first_visible_col(**args*, ***kwds*)

set_first_visible_row(**args*, ***kwds*)

set_fit_height_to_pages(**args*, ***kwds*)

set_fit_num_pages(**args*, ***kwds*)

set_fit_width_to_pages(**args*, ***kwds*)

set_footer_margin(**args*, ***kwds*)

set_footer_str(**args*, ***kwds*)

set_frmla_opts(**args*, ***kwds*)

set_grid_colour(**args*, ***kwds*)

set_header_margin(**args*, ***kwds*)

set_header_str(**args*, ***kwds*)

set_hidden(**args*, ***kwds*)

set_horz_page_breaks(**args*, ***kwds*)

set_horz_split_first_visible(**args*, ***kwds*)

set_horz_split_pos(**args*, ***kwds*)

set_iterations_on(**args*, ***kwds*)

set_left_margin(**args*, ***kwds*)

set_link(*x*, *y*, *url*, *target=None*, *description=None*)

set_name(**args*, ***kwds*)

set_normal_magn(**args*, ***kwds*)

set_obj_protect(**args*, ***kwds*)

set_outline_below(**args*, ***kwds*)

set_outline_right(**args*, ***kwds*)

set_page_preview(**args*, ***kwds*)

set_panes_frozen(**args*, ***kwds*)

set_paper_size_code(**args*, ***kwds*)

set_password(**args*, ***kwds*)

set_portrait(**args*, ***kwds*)

set_preview_magn(**args*, ***kwds*)

set_print_centered_horz(**args*, ***kwds*)

set_print_centered_vert(**args*, ***kwds*)

set_print_colour(**args*, ***kwds*)

set_print_draft(**args*, ***kwds*)

set_print_grid(**args*, ***kwds*)

set_print_headers(**args*, ***kwds*)

set_print_hres(**args*, ***kwds*)

set_print_in_rows(**args*, ***kwds*)

set_print_notes(**args*, ***kwds*)

set_print_notes_at_end(**args*, ***kwds*)

set_print_omit_errors(**args*, ***kwds*)

set_print_scaling(**args*, ***kwds*)

set_print_vres(**args*, ***kwds*)

set_protect(**args*, ***kwds*)

set_remove_splits(**args*, ***kwds*)

set_right_margin(**args*, ***kwds*)

set_row_default_height(**args*, ***kwds*)

set_save_recalc(**args*, ***kwds*)

set_scen_protect(**args*, ***kwds*)

set_selected(**args*, ***kwds*)

set_show_auto_page_breaks(**args*, ***kwds*)

set_show_col_outline(**args*, ***kwds*)

set_show_empty_as_zero(**args*, ***kwds*)

set_show_formulas(**args*, ***kwds*)

set_show_grid(**args*, ***kwds*)

set_show_headers(**args*, ***kwds*)

set_show_outline(**args*, ***kwds*)

set_show_row_outline(**args*, ***kwds*)

set_start_page_number(**args*, ***kwds*)

set_top_margin(**args*, ***kwds*)

set_vert_page_breaks(**args*, ***kwds*)

set_vert_split_first_visible(**args*, ***kwds*)

set_vert_split_pos(**args*, ***kwds*)

set_wnd_protect(**args*, ***kwds*)

show_auto_page_breaks

show_col_outline

show_empty_as_zero

show_formulas

show_grid

show_headers

show_outline

show_row_outline

start_page_number

top_margin

vert_page_breaks

vert_split_first_visible

vert_split_pos

wnd_protect

write(*r*, *c*, *label=''*, *style=<pyExcelerator.Style.XFStyle object>*)

write_cols(*r*, *cstart*, *cdata*, *style=<pyExcelerator.Style.XFStyle object>*)

write_merge(*r1*, *r2*, *c1*, *c2*, *label=''*, *style=<pyExcelerator.Style.XFStyle object>*)

1.20 antlr Module

exception pyExcelerator.antlr.**ANTLRException**(**args*)
Bases: exceptions.Exception

class pyExcelerator.antlr.**AST**
Bases: object

addChild(*c*)

equals(*t*)

equalsList(*t*)

equalsListPartial(*t*)

equalsTree(*t*)

equalsTreePartial(*t*)

findAll(*tree*)

findAllPartial(*subtree*)

getColumn()

getFirstChild()

getLine()

getNextSibling()

getNumberOfChildren()

getText()

getType()

initialize(*t*)

setFirstChild(*c*)

setNextSibling(*n*)

setText(*text*)

setType(*ttype*)

toString()

toStringList()

toStringTree()

class pyExcelerator.antlr.**ASTFactory**(*table=None*)
Bases: object

create(**args*)

dup(*t*)

dupList(*t*)

dupTree(*t*)

error(*e*)

getASTNodeClass()

getASTNodeType(*tokenType*)
For a given token type return the AST node type. First we lookup a mapping table, second we try _class and finally we resolve to "antlr.CommonAST".

getTokenTypeToASTClassMap()

maptype(*tokenType*, *className*)
Specify a mapping between a token type and a (AST) class.

setASTNodeClass(*className=None*)

setASTNodeType(*className=None*)

setTokenTypeASTNodeType(*tokenType*, *className*)
Specify a mapping between a token type and a (AST) class.

setTokenTypeToASTClassMap(*amap*)

class pyExcelerator.antlr.**ASTNULLType**
Bases: *pyExcelerator.antlr.AST*

getText()

getType()

class pyExcelerator.antlr.**ASTPair**
Bases: object

advanceChildToEnd()

copy()

toString()

class pyExcelerator.antlr.**ASTVisitor**(**args*)
Bases: object

visit(*ast*)

class pyExcelerator.antlr.**BaseAST**
Bases: *pyExcelerator.antlr.AST*

addChild(*node*)

doWorkForFindAll(*v*, *target*, *partialMatch*)

equals(*t*)

equalsList(*t*)

equalsListPartial(*t*)

equalsTree(*t*)

equalsTreePartial(*t*)

findAll(*target*)

findAllPartial(*sub*)

getColumn()

getFirstChild()

getLine()

getNextSibling()

getNumberOfChildren()

getText()

getTokenNames()

getType()

removeChildren()

setFirstChild(*c*)

setNextSibling(*n*)

setText(*text*)

setType(*ttype*)

static **setVerboseStringConversion**(*verbose*, *names*)

toString()

toStringList()

toStringTree()

tokenNames = None

verboseStringConversion = False

class pyExcelerator.antlr.**BitSet**(*data=None*)
Bases: object

BITS = 64

LOG_BITS = 6

MOD_MASK = 63

NIBBLE = 4

add(*bit*, *on=True*)

at(*bit*)

bitMask(*bit*)

member(*item*)

off(*bit*, *off=True*)

set(*bit*, *on=True*)

wordNumber(*bit*)

class pyExcelerator.antlr.**CharBuffer**(*reader*)
Bases: *pyExcelerator.antlr.InputBuffer*

fill(*amount*)

class pyExcelerator.antlr.**CharScanner**(**argv*, ***kwargs*)
Bases: *pyExcelerator.antlr.TokenStream*

EOF_CHAR = ''

LA(*i*)

NO_CHAR = 0

append(*c*)

commit()

consume()

consumeUntil_bitset(*bitset*)

consumeUntil_char(*c*)

default(*la1*)

filterdefault(*la1*, **args*)

getCaseSensitive()

getCaseSensitiveLiterals()

getColumn()

getCommitToPath()

getFilename()

getInputBuffer()

getInputState()

getLine()

getTabSize()

getText()

getTokenObject()

makeToken(*type*)

mark()

match(*item*)

matchNot(*c*)

matchRange(*c1*, *c2*)

newline()

panic(*s=''*)

raise_NoViableAlt(*la1=None*)

reportError(*s*)

reportWarning(*s*)

resetText()

rewind(*pos*)

setCaseSensitive(*t*)

setColumn(*c*)

setCommitToPath(*commit*)

setFilename(*f*)

setInput(**argv*)

setInputState(*state*)

setLine(*line*)

setTabSize(*size*)

setText(*s*)

setTokenObjectClass(*cl*)

set_return_token(*_create*, *_token*, *_ttype*, *_offset*)

tab()

testForLiteral(*token*)

testLiteralsTable(**args*)

toLower(*c*)

traceIn(*rname*)

traceIndent()

traceOut(*rname*)

uponEOF()

class pyExcelerator.antlr.**CharScannerIterator**(*inst*)

next()

exception pyExcelerator.antlr.**CharStreamException**(**args*)
Bases: *pyExcelerator.antlr.ANTLRException*

exception pyExcelerator.antlr.**CharStreamIOException**(**args*)
Bases: *pyExcelerator.antlr.CharStreamException*

class pyExcelerator.antlr.**CommonAST**(*token=None*)
Bases: *pyExcelerator.antlr.BaseAST*

getText()

getType()

initialize(**args*)

setText(*text_*)

setType(*ttype_*)

class pyExcelerator.antlr.**CommonASTWithHiddenTokens**(**args*)
Bases: *pyExcelerator.antlr.CommonAST*

getHiddenAfter()

getHiddenBefore()

initialize(**args*)

class pyExcelerator.antlr.**CommonHiddenStreamToken**(**args*)
Bases: *pyExcelerator.antlr.CommonToken*

getHiddenAfter()

getHiddenBefore()

setHiddenAfter(*t*)

setHiddenBefore(*t*)

class pyExcelerator.antlr.**CommonToken**(***argv*)
Bases: *pyExcelerator.antlr.Token*

getColumn()

getLine()

getText()

setColumn(*col*)

setLine(*line*)

setText(*text*)

toString()

class pyExcelerator.antlr.**InputBuffer**
Bases: object

LA(*k*)

commit()

consume()

fill(*k*)

getLAChars()

getMarkedChars()

isMarked()

mark()

reset()

rewind(*mark*)

syncConsume()

class pyExcelerator.antlr.**LLkParser**(**args*, ***kwargs*)
Bases: *pyExcelerator.antlr.Parser*

LA(*i*)

LT(*i*)

consume()

set_k(*index*, **args*)

trace(*ee*, *rname*)

traceIn(*rname*)

traceOut(*rname*)

class pyExcelerator.antlr.**LexerSharedInputState**(*ibuf*)
Bases: object

LA(*k*)

reset()

exception pyExcelerator.antlr.**MismatchedCharException**(**args*)
Bases: *pyExcelerator.antlr.RecognitionException*

CHAR = 1

NONE = 0

NOT_CHAR = 2

NOT_RANGE = 4

NOT_SET = 6

RANGE = 3

SET = 5

appendCharName(*sb*, *c*)

exception pyExcelerator.antlr.**MismatchedTokenException**(**args*)
Bases: *pyExcelerator.antlr.RecognitionException*

NONE = 0

NOT_RANGE = 4

NOT_SET = 6

NOT_TOKEN = 2

RANGE = 3

SET = 5

TOKEN = 1

appendTokenName(*sb*, *tokenType*)

exception pyExcelerator.antlr.**NoViableAltException**(**args*)
Bases: *pyExcelerator.antlr.RecognitionException*

exception pyExcelerator.antlr.**NoViableAltForCharException**(**args*)
Bases: *pyExcelerator.antlr.RecognitionException*

class pyExcelerator.antlr.**Parser**(**args*, ***kwargs*)
Bases: object

LA(*i*)

LT(*i*)

addASTChild(*currentAST*, *child*)

addMessageListener(*l*)

addParserListener(*l*)

addParserMatchListener(*l*)

addParserTokenListener(*l*)

addSemanticPredicateListener(*l*)

addSyntacticPredicateListener(*l*)

addTraceListener(*l*)

consume()

consumeUntil(*arg*)

defaultDebuggingSetup()

getAST()

getASTFactory()

getFilename()

getInputState()

getTokenName(*num*)

getTokenNames()

getTokenTypeToASTClassMap()

isDebugMode()

makeASTRoot(*currentAST*, *root*)

mark()

match(*set*)

matchNot(*t*)

removeMessageListener(*l*)

removeParserListener(*l*)

removeParserMatchListener(*l*)

removeParserTokenListener(*l*)

removeSemanticPredicateListener(*l*)

removeSyntacticPredicateListener(*l*)

removeTraceListener(*l*)

reportError(*x*)

reportWarning(*s*)

rewind(*pos*)

setASTFactory(*f*)

setASTNodeClass(*cl*)

setASTNodeType(*nodeType*)

setDebugMode(*debugMode*)

setFilename(*f*)

setIgnoreInvalidDebugCalls(*value*)

setInputState(*state*)

setTokenBuffer(*t*)

traceIn(*rname*)

traceIndent()

traceOut(*rname*)

class pyExcelerator.antlr.**ParserSharedInputState**
Bases: object

reset()

class pyExcelerator.antlr.**Queue**
Bases: object

append(*item*)

elementAt(*index*)

length()

removeFirst()

reset()

class pyExcelerator.antlr.**Reader**(*stream*)
Bases: object

read(*num*)

exception pyExcelerator.antlr.**RecognitionException**(**args*)
Bases: *pyExcelerator.antlr.ANTLRException*

exception pyExcelerator.antlr.**SemanticException**(**args*)
Bases: *pyExcelerator.antlr.RecognitionException*

class pyExcelerator.antlr.**StringBuffer**(*string=None*)

append(*c*)

getString(*a=None*, *length=None*)

length()

setLength(*sz*)

toString(*a=None*, *length=None*)

class pyExcelerator.antlr.**Token**(***argv*)
Bases: object

EOF = 1

EOF_TYPE = 1

INVALID_TYPE = 0

MIN_USER_TYPE = 4

NULL_TREE_LOOKAHEAD = 3

SKIP = -1

badToken = ["<no text>",<INVALID_TYPE>]

getColumn()

getFilename()

getLine()

getText()

getType()

isEOF()

setColumn(*column*)

setFilename(*name*)

setLine(*line*)

setText(*text*)

setType(*type*)

toString()

class pyExcelerator.antlr.**TokenBuffer**(*stream*)
Bases: object

LA(*k*)

LT(*k*)

consume()

fill(*amount*)

getInput()

mark()

reset()

rewind(*mark*)

syncConsume()

class pyExcelerator.antlr.**TokenStream**
Bases: object

nextToken()

class pyExcelerator.antlr.**TokenStreamBasicFilter**(*input*)
Bases: *pyExcelerator.antlr.TokenStream*

discard(*arg*)

nextToken()

exception pyExcelerator.antlr.**TokenStreamException**(**args*)
Bases: *pyExcelerator.antlr.ANTLRException*

class pyExcelerator.antlr.**TokenStreamHiddenTokenFilter**(*input*)
Bases: *pyExcelerator.antlr.TokenStreamBasicFilter*

LA(*i*)

consume()

consumeFirst()

getDiscardMask()

getHiddenAfter(*t*)

getHiddenBefore(*t*)

getHideMask()

getInitialHiddenToken()

hide(*m*)

nextToken()

exception pyExcelerator.antlr.**TokenStreamIOException**(**args*)
Bases: *pyExcelerator.antlr.TokenStreamException*

class pyExcelerator.antlr.**TokenStreamIterator**(*inst*)
Bases: object

next()

exception pyExcelerator.antlr.**TokenStreamRecognitionException**(**args*)
Bases: *pyExcelerator.antlr.TokenStreamException*

exception pyExcelerator.antlr.**TokenStreamRetryException**(**args*)
Bases: *pyExcelerator.antlr.TokenStreamException*

class pyExcelerator.antlr.**TokenStreamSelector**
Bases: *pyExcelerator.antlr.TokenStream*

addInputStream(*stream*, *key*)

getCurrentStream()

getStream(*sname*)

nextToken()

pop()

push(*arg*)

retry()

select(*arg*)

class pyExcelerator.antlr.**TreeParser**(**args*, ***kwargs*)
Bases: object

addASTChild(*currentAST*, *child*)

getAST()

getASTFactory()

getTokenName(*num*)

getTokenNames()

makeASTRoot(*currentAST*, *root*)

match(*t*, *set*)

matchNot(*t*, *ttype*)

reportError(*ex*)

reportWarning(*s*)

setASTFactory(*f*)

setASTNodeClass(*nodeType*)

setASTNodeType(*nodeType*)

traceIn(*rname*, *t*)

traceIndent()

traceOut(*rname*, *t*)

class pyExcelerator.antlr.**TreeParserSharedInputState**
Bases: object

exception pyExcelerator.antlr.**TryAgain**
Bases: exceptions.Exception

pyExcelerator.antlr.**cmptree**(*s*, *t*, *partial*)

pyExcelerator.antlr.**dup**(*t*, *factory*)

pyExcelerator.antlr.**dupList**(*t*, *factory*)

pyExcelerator.antlr.**dupTree**(*t*, *factory*)

pyExcelerator.antlr.**error**(*fmt*, **args*)

pyExcelerator.antlr.**ifelse**(*cond*, *_then*, *_else*)

pyExcelerator.antlr.**illegalarg_ex**(*func*)

pyExcelerator.antlr.**make**(**nodes*)

pyExcelerator.antlr.**rightmost**(*ast*)

pyExcelerator.antlr.**runtime_ex**(*func*)

CHAPTER 2

tools Package

CHAPTER 3

examples Package

3.1 merged Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: merged.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    fnt = Font()
    fnt.name = 'Arial'
    fnt.colour_index = 4
    fnt.bold = True

    borders = Borders()
    borders.left = 6
    borders.right = 6
    borders.top = 6
    borders.bottom = 6

    al = Alignment()
    al.horz = Alignment.HORZ_CENTER
    al.vert = Alignment.VERT_CENTER

    style = XFStyle()
    style.font = fnt
    style.borders = borders
    style.alignment = al


    wb = Workbook()
```

```
    ws0 = wb.add_sheet('sheet0')
    ws1 = wb.add_sheet('sheet1')
    ws2 = wb.add_sheet('sheet2')

    for i in range(0, 0x200, 2):
        ws0.write_merge(i, i+1, 1, 5, 'test %d' % i, style)
        ws1.write_merge(i, i, 1, 7, 'test %d' % i, style)
        ws2.write_merge(i, i+1, 1, 7 + (i%10), 'test %d' % i, style)


    wb.save('merged.xls')
```

3.2 row_styles Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: row_styles.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    for i in range(6, 80):
        fnt = Font()
        fnt.height = i*20
        style = XFStyle()
        style.font = fnt
        ws.write(i, 1, 'Test')
        ws.row(i).set_style(style)
    w.save('row_styles.xls')
```

3.3 row_styles_empty Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: row_styles_empty.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    for i in range(6, 80):
        fnt = Font()
        fnt.height = i*20
```

```
        style = XFStyle()
        style.font = fnt
        ws.row(i).set_style(style)
    w.save('row_styles_empty.xls')
```

3.4 outline Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: outline.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    fnt = Font()
    fnt.name = 'Arial'
    fnt.colour_index = 4
    fnt.bold = True

    borders = Borders()
    borders.left = 6
    borders.right = 6
    borders.top = 6
    borders.bottom = 6

    style = XFStyle()
    style.font = fnt
    style.borders = borders

    wb = Workbook()

    ws0 = wb.add_sheet('Rows Outline')

    ws0.write_merge(1, 1, 1, 5, 'test 1', style)
    ws0.write_merge(2, 2, 1, 4, 'test 1', style)
    ws0.write_merge(3, 3, 1, 3, 'test 2', style)
    ws0.write_merge(4, 4, 1, 4, 'test 1', style)
    ws0.write_merge(5, 5, 1, 4, 'test 3', style)
    ws0.write_merge(6, 6, 1, 5, 'test 1', style)
    ws0.write_merge(7, 7, 1, 5, 'test 4', style)
    ws0.write_merge(8, 8, 1, 4, 'test 1', style)
    ws0.write_merge(9, 9, 1, 3, 'test 5', style)

    ws0.row(1).level = 1
    ws0.row(2).level = 1
    ws0.row(3).level = 2
    ws0.row(4).level = 2
    ws0.row(5).level = 2
    ws0.row(6).level = 2
    ws0.row(7).level = 2
    ws0.row(8).level = 1
    ws0.row(9).level = 1
```

```
    ws1 = wb.add_sheet('Columns Outline')

    ws1.write_merge(1, 1, 1, 5, 'test 1', style)
    ws1.write_merge(2, 2, 1, 4, 'test 1', style)
    ws1.write_merge(3, 3, 1, 3, 'test 2', style)
    ws1.write_merge(4, 4, 1, 4, 'test 1', style)
    ws1.write_merge(5, 5, 1, 4, 'test 3', style)
    ws1.write_merge(6, 6, 1, 5, 'test 1', style)
    ws1.write_merge(7, 7, 1, 5, 'test 4', style)
    ws1.write_merge(8, 8, 1, 4, 'test 1', style)
    ws1.write_merge(9, 9, 1, 3, 'test 5', style)

    ws1.col(1).level = 1
    ws1.col(2).level = 1
    ws1.col(3).level = 2
    ws1.col(4).level = 2
    ws1.col(5).level = 2
    ws1.col(6).level = 2
    ws1.col(7).level = 2
    ws1.col(8).level = 1
    ws1.col(9).level = 1


    ws2 = wb.add_sheet('Rows and Columns Outline')

    ws2.write_merge(1, 1, 1, 5, 'test 1', style)
    ws2.write_merge(2, 2, 1, 4, 'test 1', style)
    ws2.write_merge(3, 3, 1, 3, 'test 2', style)
    ws2.write_merge(4, 4, 1, 4, 'test 1', style)
    ws2.write_merge(5, 5, 1, 4, 'test 3', style)
    ws2.write_merge(6, 6, 1, 5, 'test 1', style)
    ws2.write_merge(7, 7, 1, 5, 'test 4', style)
    ws2.write_merge(8, 8, 1, 4, 'test 1', style)
    ws2.write_merge(9, 9, 1, 3, 'test 5', style)

    ws2.row(1).level = 1
    ws2.row(2).level = 1
    ws2.row(3).level = 2
    ws2.row(4).level = 2
    ws2.row(5).level = 2
    ws2.row(6).level = 2
    ws2.row(7).level = 2
    ws2.row(8).level = 1
    ws2.row(9).level = 1

    ws2.write_merge(1, 1, 1, 5, 'test 1', style)
    ws2.write_merge(2, 2, 1, 4, 'test 1', style)
    ws2.write_merge(3, 3, 1, 3, 'test 2', style)
    ws2.write_merge(4, 4, 1, 4, 'test 1', style)
    ws2.write_merge(5, 5, 1, 4, 'test 3', style)
    ws2.write_merge(6, 6, 1, 5, 'test 1', style)
    ws2.write_merge(7, 7, 1, 5, 'test 4', style)
    ws2.write_merge(8, 8, 1, 4, 'test 1', style)
    ws2.write_merge(9, 9, 1, 3, 'test 5', style)

    ws2.col(1).level = 1
    ws2.col(2).level = 1
```

```
    ws2.col(3).level = 2
    ws2.col(4).level = 2
    ws2.col(5).level = 2
    ws2.col(6).level = 2
    ws2.col(7).level = 2
    ws2.col(8).level = 1
    ws2.col(9).level = 1


    wb.save('outline.xls')
```

3.5 unicode2 Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: unicode2.py,v 1.1 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws1 = w.add_sheet(u'\N{GREEK SMALL LETTER ALPHA}\N{GREEK SMALL LETTER BETA}\N
↪{GREEK SMALL LETTER GAMMA}\u2665\u041e\u041b\u042f\u2665')

    fnt = Font()
    fnt.height = 26*20
    style = XFStyle()
    style.font = fnt

    for i in range(0x10000):
        ws1.write(i/0x10, i%0x10, unichr(i), style)

    w.save('unicode2.xls')
```

3.6 blanks Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: blanks.py,v 1.2 2005/07/22 08:22:26 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    font0 = Font()
    font0.name = 'Times New Roman'
    font0.struck_out = True
    font0.bold = True
```

```
    style0 = XFStyle()
    style0.font = font0


    wb = Workbook()
    ws0 = wb.add_sheet('0')

    ws0.write(1, 1, 'Test', style0)

    for i in range(0, 0x53):
        borders = Borders()
        borders.left = i
        borders.right = i
        borders.top = i
        borders.bottom = i

        style = XFStyle()
        style.borders = borders

        ws0.write(i, 2, '', style)
        ws0.write(i, 3, hex(i), style0)

    ws0.write_merge(5, 8, 6, 10, "")

    wb.save('blanks.xls')
```

3.7 sst Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: sst.py,v 1.2 2005/03/25 07:31:12 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    font0 = Formatting.Font()
    font0.name = 'Arial'
    font1 = Formatting.Font()
    font1.name = 'Arial Cyr'
    font2 = Formatting.Font()
    font2.name = 'Times New Roman'
    font3 = Formatting.Font()
    font3.name = 'Courier New Cyr'

    num_format0 = '0.00000'
    num_format1 = '0.000000'
    num_format2 = '0.0000000'
    num_format3 = '0.00000000'

    st0 = XFStyle()
    st1 = XFStyle()
    st2 = XFStyle()
    st3 = XFStyle()
```

```
    st4 = XFStyle()

    st0.font = font0
    st0.num_format = num_format0

    st1.font = font1
    st1.num_format = num_format1

    st2.font = font2
    st2.num_format = num_format2

    st3.font = font3
    st3.num_format = num_format3

    wb = Workbook()

    wb.add_style(st0)
    wb.add_style(st1)
    wb.add_style(st2)
    wb.add_style(st3)

    ws0 = wb.add_sheet('0')
    ws0.write(0, 0, 'Olya'*0x4000, st0)

    #for i in range(0, 0x10):
    #    ws0.write(i, 2, ('%d'%i)*0x4000, st1)

    wb.save('sst.xls')
```

3.8 wsprops Module

```
__rev_id__ = """$Id: wsprops.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

props = \
[
        'name',
        'parent',
        'rows',
        'cols',
        'merged_ranges',
        'bmp_rec',
        'show_formulas',
        'show_grid',
        'show_headers',
        'panes_frozen',
        'show_empty_as_zero',
        'auto_colour_grid',
        'cols_right_to_left',
        'show_outline',
        'remove_splits',
        'selected',
        'hidden',
        'page_preview',
        'first_visible_row',
```

```
        'first_visible_col',
        'grid_colour',
        'preview_magn',
        'normal_magn',
        'row_gut_width',
        'col_gut_height',
        'show_auto_page_breaks',
        'dialogue_sheet',
        'auto_style_outline',
        'outline_below',
        'outline_right',
        'fit_num_pages',
        'show_row_outline',
        'show_col_outline',
        'alt_expr_eval',
        'alt_formula_entries',
        'row_default_height',
        'col_default_width',
        'calc_mode',
        'calc_count',
        'RC_ref_mode',
        'iterations_on',
        'delta',
        'save_recalc',
        'print_headers',
        'print_grid',
        'grid_set',
        'vert_page_breaks',
        'horz_page_breaks',
        'header_str',
        'footer_str',
        'print_centered_vert',
        'print_centered_horz',
        'left_margin',
        'right_margin',
        'top_margin',
        'bottom_margin',
        'paper_size_code',
        'print_scaling',
        'start_page_number',
        'fit_width_to_pages',
        'fit_height_to_pages',
        'print_in_rows',
        'portrait',
        'print_not_colour',
        'print_draft',
        'print_notes',
        'print_notes_at_end',
        'print_omit_errors',
        'print_hres',
        'print_vres',
        'header_margin',
        'footer_margin',
        'copies_num',
]
if __name__ == '__main__':
    from pyExcelerator import *
```

```
    wb = Workbook()
    ws = wb.add_sheet('sheet')

    print ws.name
    print ws.parent
    print ws.rows
    print ws.cols
    print ws.merged_ranges
    print ws.bmp_rec
    print ws.show_formulas
    print ws.show_grid
    print ws.show_headers
    print ws.panes_frozen
    print ws.show_empty_as_zero
    print ws.auto_colour_grid
    print ws.cols_right_to_left
    print ws.show_outline
    print ws.remove_splits
    print ws.selected
    print ws.hidden
    print ws.page_preview
    print ws.first_visible_row
    print ws.first_visible_col
    print ws.grid_colour
    print ws.preview_magn
    print ws.normal_magn
    #print ws.row_gut_width
    #print ws.col_gut_height
    print ws.show_auto_page_breaks
    print ws.dialogue_sheet
    print ws.auto_style_outline
    print ws.outline_below
    print ws.outline_right
    print ws.fit_num_pages
    print ws.show_row_outline
    print ws.show_col_outline
    print ws.alt_expr_eval
    print ws.alt_formula_entries
    print ws.row_default_height
    print ws.col_default_width
    print ws.calc_mode
    print ws.calc_count
    print ws.RC_ref_mode
    print ws.iterations_on
    print ws.delta
    print ws.save_recalc
    print ws.print_headers
    print ws.print_grid
    #print ws.grid_set
    print ws.vert_page_breaks
    print ws.horz_page_breaks
    print ws.header_str
    print ws.footer_str
    print ws.print_centered_vert
    print ws.print_centered_horz
    print ws.left_margin
    print ws.right_margin
    print ws.top_margin
```

```
    print ws.bottom_margin
    print ws.paper_size_code
    print ws.print_scaling
    print ws.start_page_number
    print ws.fit_width_to_pages
    print ws.fit_height_to_pages
    print ws.print_in_rows
    print ws.portrait
    print ws.print_colour
    print ws.print_draft
    print ws.print_notes
    print ws.print_notes_at_end
    print ws.print_omit_errors
    print ws.print_hres
    print ws.print_vres
    print ws.header_margin
    print ws.footer_margin
    print ws.copies_num
```

3.9 num_formats Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: num_formats.py,v 1.1 2005/07/20 07:24:11 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    fmts = [
        'general',
        '0',
        '0.00',
        '#,##0',
        '#,##0.00',
        '"$"#,##0_);("$"#,##',
        '"$"#,##0_);[Red]("$"#,##',
        '"$"#,##0.00_);("$"#,##',
        '"$"#,##0.00_);[Red]("$"#,##',
        '0%',
        '0.00%',
        '0.00E+00',
        '# ?/?',
        '# ??/??',
        'M/D/YY',
        'D-MMM-YY',
        'D-MMM',
        'MMM-YY',
        'h:mm AM/PM',
        'h:mm:ss AM/PM',
        'h:mm',
```

```
        'h:mm:ss',
        'M/D/YY h:mm',
        '_(#,##0_);(#,##0)',
        '_(#,##0_);[Red](#,##0)',
        '_(#,##0.00_);(#,##0.00)',
        '_(#,##0.00_);[Red](#,##0.00)',
        '_("$"* #,##0_);_("$"* (#,##0);_("$"* "-"_);_(@_)',
        '_(* #,##0_);_(* (#,##0);_(* "-"_);_(@_)',
        '_("$"* #,##0.00_);_("$"* (#,##0.00);_("$"* "-"??_);_(@_)',
        '_(* #,##0.00_);_(* (#,##0.00);_(* "-"??_);_(@_)',
        'mm:ss',
        '[h]:mm:ss',
        'mm:ss.0',
        '##0.0E+0',
        '@'
    ]

    i = 0
    for fmt in fmts:
        ws.write(i, 0, fmt)

        style = XFStyle()
        style.num_format_str = fmt

        ws.write(i, 4, -1278.9078, style)

        i += 1

    w.save('num_formats.xls')
```

3.10 `xls2html` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: xls2html.py,v 1.1 2005/05/19 09:27:42 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *
    import sys

    me, args = sys.argv[0], sys.argv[1:]


    if args:
        for arg in args:
            print >>sys.stderr, 'extracting data from', arg
            print '<html>'
            for sheet_name, values in parse_xls(arg, 'cp1251'): # parse_xls(arg) --␣
↪default encoding
                matrix = [[]]
                print 'Sheet = "%s"' % sheet_name.encode('cp1251', 'backslashreplace')
                print '<table border="2" cellpadding="1" cellspacing="1" >'
```

```
                for row_idx, col_idx in sorted(values.keys()):
                    v = values[(row_idx, col_idx)]
                    if isinstance(v, unicode):
                        v = v.encode('cp1251', 'backslashreplace')
                    else:
                        v = str(v)
                    last_row, last_col = len(matrix), len(matrix[-1])
                    while last_row < row_idx:
                        matrix.extend([[]])
                        last_row = len(matrix)

                    while last_col < col_idx:
                        matrix[-1].extend([''])
                        last_col = len(matrix[-1])

                    matrix[-1].extend([v])

                for row in matrix:
                    print '<tr>'
                    for col in row:
                        print '<td> %s </td>' % col
                    print '</tr>'

                print '</table>'
            print '</html>'

    else:
        print 'usage: %s (inputfile)+' % me
```

3.11 merged0 Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: merged0.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    wb = Workbook()
    ws0 = wb.add_sheet('sheet0')


    fnt = Font()
    fnt.name = 'Arial'
    fnt.colour_index = 4
    fnt.bold = True

    borders = Borders()
    borders.left = 6
    borders.right = 6
    borders.top = 6
    borders.bottom = 6
```

```
    style = XFStyle()
    style.font = fnt
    style.borders = borders

    ws0.write_merge(3, 3, 1, 5, 'test1', style)
    ws0.write_merge(4, 10, 1, 5, 'test2', style)
    ws0.col(1).width = 0x0d00

    wb.save('merged0.xls')
```

3.12 format Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: format.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    style0 = XFStyle()
    style0.font.name = 'Times New Roman'
    style0.font.struck_out = True
    style0.font.bold = True


    wb = Workbook()
    ws0 = wb.add_sheet('0')

    ws0.write(1, 1, 'Test', style0)

    for i in range(0, 0x53):
        style = XFStyle()
        style.font.name = 'Arial'
        style.font.colour_index = i
        style.font.outline = True
        style.borders.left = i

        ws0.write(i, 2, 'colour', style)
        ws0.write(i, 3, hex(i), style0)

    wb.save('format.xls')
```

3.13 merged1 Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: merged1.py,v 1.1 2005/07/22 08:22:26 rvk Exp $"""
```

```
if __name__ == '__main__':
    from pyExcelerator import *

    wb = Workbook()
    ws0 = wb.add_sheet('sheet0')

    fnt1 = Font()
    fnt1.name = 'Verdana'
    fnt1.bold = True
    fnt1.height = 18*0x14

    pat1 = Pattern()
    pat1.pattern = Pattern.SOLID_PATTERN
    pat1.pattern_fore_colour = 0x16

    brd1 = Borders()
    brd1.left = 0x06
    brd1.right = 0x06
    brd1.top = 0x06
    brd1.bottom = 0x06

    fnt2 = Font()
    fnt2.name = 'Verdana'
    fnt2.bold = True
    fnt2.height = 14*0x14

    brd2 = Borders()
    brd2.left = 0x01
    brd2.right = 0x01
    brd2.top = 0x01
    brd2.bottom = 0x01

    pat2 = Pattern()
    pat2.pattern = Pattern.SOLID_PATTERN
    pat2.pattern_fore_colour = 0x01F

    fnt3 = Font()
    fnt3.name = 'Verdana'
    fnt3.bold = True
    fnt3.italic = True
    fnt3.height = 12*0x14

    brd3 = Borders()
    brd3.left = 0x07
    brd3.right = 0x07
    brd3.top = 0x07
    brd3.bottom = 0x07

    fnt4 = Font()

    al1 = Alignment()
    al1.horz = Alignment.HORZ_CENTER
    al1.vert = Alignment.VERT_CENTER

    al2 = Alignment()
    al2.horz = Alignment.HORZ_RIGHT
    al2.vert = Alignment.VERT_CENTER
```

```
    al3 = Alignment()
    al3.horz = Alignment.HORZ_LEFT
    al3.vert = Alignment.VERT_CENTER

    style1 = XFStyle()
    style1.font = fnt1
    style1.alignment = al1
    style1.pattern = pat1
    style1.borders = brd1

    style2 = XFStyle()
    style2.font = fnt2
    style2.alignment = al1
    style2.pattern = pat2
    style2.borders = brd2

    style3 = XFStyle()
    style3.font = fnt3
    style3.alignment = al1
    style3.pattern = pat2
    style3.borders = brd3

    price_style = XFStyle()
    price_style.font = fnt4
    price_style.alignment = al2
    price_style.borders = brd3
    price_style.num_format_str = '_(#,##0.00_) "money"'

    ware_style = XFStyle()
    ware_style.font = fnt4
    ware_style.alignment = al3
    ware_style.borders = brd3


    ws0.merge(3, 3, 1, 5, style1)
    ws0.merge(4, 10, 1, 6, style2)
    ws0.merge(14, 16, 1, 7, style3)
    ws0.col(1).width = 0x0d00


    wb.save('merged1.xls')
```

3.14 `big-35Mb` Module

```
#!/usr/bin/env python
# tries stress SST, SAT and MSAT
__rev_id__ = """$Id: big-35Mb.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from time import *
    from pyExcelerator import *

    style = XFStyle()
```

```
    wb = Workbook()
    ws0 = wb.add_sheet('0')

    colcount = 200 + 1
    rowcount = 6000 + 1

    t0 = time()
    print "\nstart: %s" % ctime(t0)

    print "Filling..."
    for col in xrange(colcount):
        print "[%d]" % col,
        for row in xrange(rowcount):
            ws0.write(row, col, "BIG(%d, %d)" % (row, col))
            #ws0.write(row, col, "BIG")

    t1 = time() - t0
    print "\nsince starting elapsed %.2f s" % (t1)

    print "Storing..."
    wb.save('big-35Mb.xls')

    t2 = time() - t0
    print "since starting elapsed %.2f s" % (t2)
```

3.15 `col_width` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: col_width.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    for i in range(6, 80):
        fnt = Font()
        fnt.height = i*20
        style = XFStyle()
        style.font = fnt
        ws.write(1, i, 'Test')
        ws.col(i).width = 0x0d00 + i
    w.save('col_width.xls')
```

3.16 dates Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: dates.py,v 1.1 2005/07/20 07:24:11 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *
    from datetime import datetime

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    fmts = [
        'M/D/YY',
        'D-MMM-YY',
        'D-MMM',
        'MMM-YY',
        'h:mm AM/PM',
        'h:mm:ss AM/PM',
        'h:mm',
        'h:mm:ss',
        'M/D/YY h:mm',
        'mm:ss',
        '[h]:mm:ss',
        'mm:ss.0',
    ]

    i = 0
    for fmt in fmts:
        ws.write(i, 0, fmt)

        style = XFStyle()
        style.num_format_str = fmt

        ws.write(i, 4, datetime.now(), style)

        i += 1

    w.save('dates.xls')
```

3.17 unicode1 Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: unicode1.py,v 1.1 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws1 = w.add_sheet(u'\N{GREEK SMALL LETTER ALPHA}\N{GREEK SMALL LETTER BETA}\N
↪{GREEK SMALL LETTER GAMMA}')
```

```
    ws1.write(0, 0, u'\N{GREEK SMALL LETTER ALPHA}\N{GREEK SMALL LETTER BETA}\N{GREEK␣
↪SMALL LETTER GAMMA}')
    ws1.write(1, 1, u'\N{GREEK SMALL LETTER DELTA}x = 1 + \N{GREEK SMALL LETTER DELTA}
↪')

    ws1.write(2,0, u'A\u2262\u0391.')      # RFC2152 example
    ws1.write(3,0, u'Hi Mom -\u263a-!')    # RFC2152 example
    ws1.write(4,0, u'\u65E5\u672C\u8A9E') # RFC2152 example
    ws1.write(5,0, u'Item 3 is \u00a31.') # RFC2152 example
    ws1.write(8,0, u'\N{INTEGRAL}')        # RFC2152 example

    w.add_sheet(u'A\u2262\u0391.')      # RFC2152 example
    w.add_sheet(u'Hi Mom -\u263a-!')    # RFC2152 example
    one_more_ws = w.add_sheet(u'\u65E5\u672C\u8A9E') # RFC2152 example
    w.add_sheet(u'Item 3 is \u00a31.') # RFC2152 example

    one_more_ws.write(0, 0, u'\u2665\u2665')

    w.add_sheet(u'\N{GREEK SMALL LETTER ETA WITH TONOS}')
    w.save('unicode1.xls')
```

3.18 protection Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: protection.py,v 1.1 2005/10/26 07:44:24 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    fnt = Font()
    fnt.name = 'Arial'
    fnt.colour_index = 4
    fnt.bold = True

    borders = Borders()
    borders.left = 6
    borders.right = 6
    borders.top = 6
    borders.bottom = 6

    style = XFStyle()
    style.font = fnt
    style.borders = borders

    wb = Workbook()

    ws0 = wb.add_sheet('Rows Outline')

    ws0.write_merge(1, 1, 1, 5, 'test 1', style)
    ws0.write_merge(2, 2, 1, 4, 'test 1', style)
    ws0.write_merge(3, 3, 1, 3, 'test 2', style)
```

```
    ws0.write_merge(4, 4, 1, 4, 'test 1', style)
    ws0.write_merge(5, 5, 1, 4, 'test 3', style)
    ws0.write_merge(6, 6, 1, 5, 'test 1', style)
    ws0.write_merge(7, 7, 1, 5, 'test 4', style)
    ws0.write_merge(8, 8, 1, 4, 'test 1', style)
    ws0.write_merge(9, 9, 1, 3, 'test 5', style)

    ws0.row(1).level = 1
    ws0.row(2).level = 1
    ws0.row(3).level = 2
    ws0.row(4).level = 2
    ws0.row(5).level = 2
    ws0.row(6).level = 2
    ws0.row(7).level = 2
    ws0.row(8).level = 1
    ws0.row(9).level = 1


    ws1 = wb.add_sheet('Columns Outline')

    ws1.write_merge(1, 1, 1, 5, 'test 1', style)
    ws1.write_merge(2, 2, 1, 4, 'test 1', style)
    ws1.write_merge(3, 3, 1, 3, 'test 2', style)
    ws1.write_merge(4, 4, 1, 4, 'test 1', style)
    ws1.write_merge(5, 5, 1, 4, 'test 3', style)
    ws1.write_merge(6, 6, 1, 5, 'test 1', style)
    ws1.write_merge(7, 7, 1, 5, 'test 4', style)
    ws1.write_merge(8, 8, 1, 4, 'test 1', style)
    ws1.write_merge(9, 9, 1, 3, 'test 5', style)

    ws1.col(1).level = 1
    ws1.col(2).level = 1
    ws1.col(3).level = 2
    ws1.col(4).level = 2
    ws1.col(5).level = 2
    ws1.col(6).level = 2
    ws1.col(7).level = 2
    ws1.col(8).level = 1
    ws1.col(9).level = 1


    ws2 = wb.add_sheet('Rows and Columns Outline')

    ws2.write_merge(1, 1, 1, 5, 'test 1', style)
    ws2.write_merge(2, 2, 1, 4, 'test 1', style)
    ws2.write_merge(3, 3, 1, 3, 'test 2', style)
    ws2.write_merge(4, 4, 1, 4, 'test 1', style)
    ws2.write_merge(5, 5, 1, 4, 'test 3', style)
    ws2.write_merge(6, 6, 1, 5, 'test 1', style)
    ws2.write_merge(7, 7, 1, 5, 'test 4', style)
    ws2.write_merge(8, 8, 1, 4, 'test 1', style)
    ws2.write_merge(9, 9, 1, 3, 'test 5', style)

    ws2.row(1).level = 1
    ws2.row(2).level = 1
    ws2.row(3).level = 2
    ws2.row(4).level = 2
    ws2.row(5).level = 2
```

```
    ws2.row(6).level = 2
    ws2.row(7).level = 2
    ws2.row(8).level = 1
    ws2.row(9).level = 1

    ws2.write_merge(1, 1, 1, 5, 'test 1', style)
    ws2.write_merge(2, 2, 1, 4, 'test 1', style)
    ws2.write_merge(3, 3, 1, 3, 'test 2', style)
    ws2.write_merge(4, 4, 1, 4, 'test 1', style)
    ws2.write_merge(5, 5, 1, 4, 'test 3', style)
    ws2.write_merge(6, 6, 1, 5, 'test 1', style)
    ws2.write_merge(7, 7, 1, 5, 'test 4', style)
    ws2.write_merge(8, 8, 1, 4, 'test 1', style)
    ws2.write_merge(9, 9, 1, 3, 'test 5', style)

    ws2.col(1).level = 1
    ws2.col(2).level = 1
    ws2.col(3).level = 2
    ws2.col(4).level = 2
    ws2.col(5).level = 2
    ws2.col(6).level = 2
    ws2.col(7).level = 2
    ws2.col(8).level = 1
    ws2.col(9).level = 1


    ws0.protect = True
    ws0.wnd_protect = True
    ws0.obj_protect = True
    ws0.scen_protect = True
    ws0.password = "123456"

    ws1.protect = True
    ws1.wnd_protect = True
    ws1.obj_protect = True
    ws1.scen_protect = True
    ws1.password = "abcdefghij"

    ws2.protect = True
    ws2.wnd_protect = True
    ws2.obj_protect = True
    ws2.scen_protect = True
    ws2.password = "ok"

    wb.protect = True
    wb.wnd_protect = True
    wb.obj_protect = True
    wb.save('protection.xls')
```

3.19 numbers Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
```

```
__rev_id__ = """$Id: numbers.py,v 1.1 2005/07/20 07:24:11 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')

    ws.write(0, 0, 1)
    ws.write(1, 0, 1.23)
    ws.write(2, 0, 12345678)
    ws.write(3, 0, 123456.78)

    ws.write(0, 1, -1)
    ws.write(1, 1, -1.23)
    ws.write(2, 1, -12345678)
    ws.write(3, 1, -123456.78)

    ws.write(0, 2, -17867868678687.0)
    ws.write(1, 2, -1.23e-5)
    ws.write(2, 2, -12345678.90780980)
    ws.write(3, 2, -123456.78)

    w.save('numbers.xls')
```

3.20 `xls2csv` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: xls2csv.py,v 1.1 2005/05/19 09:27:42 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *
    import sys

    me, args = sys.argv[0], sys.argv[1:]


    if args:
        for arg in args:
            print >>sys.stderr, 'extracting data from', arg
            for sheet_name, values in parse_xls(arg, 'cp1251'): # parse_xls(arg) --␣
↪default encoding
                matrix = [[]]
                print 'Sheet = "%s"' % sheet_name.encode('cp866', 'backslashreplace')
                print '----------------'
                for row_idx, col_idx in sorted(values.keys()):
                    v = values[(row_idx, col_idx)]
                    if isinstance(v, unicode):
                        v = v.encode('cp866', 'backslashreplace')
                    else:
                        v = str(v)
```

```
                    last_row, last_col = len(matrix), len(matrix[-1])
                    while last_row < row_idx:
                        matrix.extend([[]])
                        last_row = len(matrix)

                    while last_col < col_idx:
                        matrix[-1].extend([''])
                        last_col = len(matrix[-1])

                    matrix[-1].extend([v])

                for row in matrix:
                    csv_row = ','.join(row)
                    print csv_row

    else:
        print 'usage: %s (inputfile)+' % me
```

3.21 parse-fmla Module

```
if __name__ == '__main__':
    from pyExcelerator import ExcelFormulaParser, ExcelFormula
    import sys

    f = ExcelFormula.Formula(
    """ -((1.80 + 2.898 * 1)/(1.80 + 2.898))*
    AVERAGE((1.80 + 2.898 * 1)/(1.80 + 2.898);
            (1.80 + 2.898 * 1)/(1.80 + 2.898);
            (1.80 + 2.898 * 1)/(1.80 + 2.898)) +
    SIN(PI()/4)""")

    #for t in f.rpn():
    #    print "%15s %15s" % (ExcelFormulaParser.PtgNames[t[0]], t[1])
```

3.22 panes Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: panes.py,v 1.1 2005/10/26 07:44:24 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws1 = w.add_sheet('sheet 1')
    ws2 = w.add_sheet('sheet 2')
    ws3 = w.add_sheet('sheet 3')
    ws4 = w.add_sheet('sheet 4')
```

```
    ws5 = w.add_sheet('sheet 5')
    ws6 = w.add_sheet('sheet 6')

    for i in range(0x100):
        ws1.write(i/0x10, i%0x10, i)

    for i in range(0x100):
        ws2.write(i/0x10, i%0x10, i)

    for i in range(0x100):
        ws3.write(i/0x10, i%0x10, i)

    for i in range(0x100):
        ws4.write(i/0x10, i%0x10, i)

    for i in range(0x100):
        ws5.write(i/0x10, i%0x10, i)

    for i in range(0x100):
        ws6.write(i/0x10, i%0x10, i)

    ws1.panes_frozen = True
    ws1.horz_split_pos = 2

    ws2.panes_frozen = True
    ws2.vert_split_pos = 2

    ws3.panes_frozen = True
    ws3.horz_split_pos = 1
    ws3.vert_split_pos = 1

    ws4.panes_frozen = False
    ws4.horz_split_pos = 12
    ws4.horz_split_first_visible = 2

    ws5.panes_frozen = False
    ws5.vert_split_pos = 40
    ws4.vert_split_first_visible = 2

    ws6.panes_frozen = False
    ws6.horz_split_pos = 12
    ws4.horz_split_first_visible = 2
    ws6.vert_split_pos = 40
    ws4.vert_split_first_visible = 2

    w.save('panes.xls')
```

3.23 `xls2txt` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: xls2txt.py,v 1.3 2005/05/19 09:27:42 rvk Exp $"""
```

```
if __name__ == '__main__':
    from pyExcelerator import *
    import sys

    me, args = sys.argv[0], sys.argv[1:]

    if args:
        for arg in args:
            print >>sys.stderr, 'extracting data from', arg
            for sheet_name, values in parse_xls(arg, 'cp1251'): # parse_xls(arg) --␣
↪default encoding
                print 'Sheet = "%s"' % sheet_name.encode('cp866', 'backslashreplace')
                print '----------------'
                for row_idx, col_idx in sorted(values.keys()):
                    v = values[(row_idx, col_idx)]
                    if isinstance(v, unicode):
                        v = v.encode('cp866', 'backslashreplace')
                    print '(%d, %d) =' % (row_idx, col_idx), v
                print '----------------'
    else:
        print 'usage: %s (inputfile)+' % me
```

3.24 `hyperlinks` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: hyperlinks.py,v 1.1 2005/10/26 07:44:24 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    f = Font()
    f.height = 20*72
    f.name = 'Verdana'
    f.bold = True
    f.underline = Font.UNDERLINE_DOUBLE
    f.colour_index = 4

    h_style = XFStyle()
    h_style.font = f

    w = Workbook()
    ws = w.add_sheet('F')
    ws_A = w.add_sheet('A')

    ##############
    ## NOTE: parameters are separated by semicolon!!!
    ##############

    n = "HYPERLINK"
    ws.write_merge(1, 1, 1, 10, Formula(n + '("http://www.irs.gov/pub/irs-pdf/f1000.
↪pdf";"f1000.pdf")'), h_style)
```

```
    ws.write_merge(2, 2, 2, 25, Formula(n + '("mailto:roman.kiseliov@gmail.com?
↪subject=pyExcelerator-feedback&Body=Hello,%20Roman!";"pyExcelerator-feedback")'), h_
↪style)

    ws.write(3, 0, "Goto Google")
    ws.set_link(3, 0, "http://www.google.com", description="Google")
    ws.write(4, 0, "Goto Next Page")
    ws.set_link(4, 0, "#A!A2", description="page A")

    w.save("hyperlinks.xls")
```

3.25 image Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman

__rev_id__ = """$Id: image.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Image')
    ws.insert_bitmap('python.bmp', 2, 2)
    ws.insert_bitmap('python.bmp', 10, 2)

    w.save('image.xls')
```

3.26 formulas Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: formulas.py,v 1.4 2005/10/26 07:44:24 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *
    from pyExcelerator.Worksheet import *

    w = Workbook()
    ws = w.add_sheet('F')
    ws.frmla_opts=NoCalcs

    ws.write(0, 0, Formula("-(1+1)", None))
    ws.write(1, 0, Formula("-(1+1)/(-2-2)", 3.14))
    ws.write(2, 0, Formula("-(134.8780789+1)", ErrorCode("#NULL!"), CalcOnOpen))
    ws.write(3, 0, Formula("-(134.8780789e-10+1)", ErrorCode("#NAME?")))
    ws.write(4, 0, Formula("-1/(1+1)+9344", 12345))
```

```
    ws.write(0, 1, Formula("-(1+1)", "this is 2", CalcOnOpen))
    ws.write(1, 1, Formula("-(1+1)/(-2-2)", u"this is 0.5"))
    ws.write(2, 1, Formula("-(134.8780789+1)", ""))
    ws.write(3, 1, Formula("-(134.8780789e-10+1)"))
    ws.write(4, 1, Formula("-1/(1+1)+9344"))

    ws.write(0, 2, Formula("A1*B1"))
    ws.write(1, 2, Formula("A2*B2"))
    ws.write(2, 2, Formula("A3*B3"))
    ws.write(3, 2, Formula("A4*B4*sin(pi()/4)"))
    ws.write(4, 2, Formula("A5%*B5*pi()/1000"))

    ##############
    ## NOTE: parameters are separated by semicolon!!!
    ##############


    ws.write(5, 2, Formula("C1+C2+C3+C4+C5/(C1+C2+C3+C4/(C1+C2+C3+C4/
↪(C1+C2+C3+C4)+C5)+C5)-20.3e-2"))
    ws.write(5, 3, Formula("C1^2"))
    ws.write(6, 2, Formula("SUM(C1;C2;;;;;C3;;;C4)"))
    ws.write(6, 3, Formula("SUM($A$1:$C$5)"))

    ws.write(7, 0, Formula('"lkjljllkllkl"'))
    ws.write(7, 1, Formula('"yuyiyiyiyi"'))
    ws.write(7, 2, Formula('A8 & B8 & A8'))
    ws.write(8, 2, Formula('now()'))

    ws.write(10, 2, Formula('TRUE'))
    ws.write(11, 2, Formula('FALSE'))
    ws.write(12, 3, Formula('IF(A1>A2;3;"hkjhjkhk")'))

    w.save('formulas.xls')
```

3.27 `mini` Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: mini.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('Hey, Dude')
    w.save('mini.xls')
```

3.28 formula_names Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: formula_names.py,v 1.1 2005/08/11 08:53:48 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws = w.add_sheet('F')

    i = 0
    for n in sorted(ExcelMagic.std_func_by_name):
        ws.write(i, 0, n)
        ws.write(i, 3, Formula(n + "($A$1)"))
        i += 1

    w.save('formula_names.xls')
```

3.29 big-16Mb Module

```
#!/usr/bin/env python
# tries stress SST, SAT and MSAT
__rev_id__ = """$Id: big-16Mb.py,v 1.3 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from time import *
    from pyExcelerator.Workbook import *
    from pyExcelerator.Style import *

    style = XFStyle()

    wb = Workbook()
    ws0 = wb.add_sheet('0')

    colcount = 200 + 1
    rowcount = 6000 + 1

    t0 = time()
    print "\nstart: %s" % ctime(t0)

    print "Filling..."
    for col in xrange(colcount):
        print "[%d]" % col,
        for row in xrange(rowcount):
            #ws0.write(row, col, "BIG(%d, %d)" % (row, col))
            ws0.write(row, col, "BIG")

    t1 = time() - t0
    print "\nsince starting elapsed %.2f s" % (t1)

    print "Storing..."
```

```
    wb.save('big-16Mb.xls')

    t2 = time() - t0
    print "since starting elapsed %.2f s" % (t2)
```

3.30 unicode0 Module

```
#!/usr/bin/env python
# -*- coding: windows-1251 -*-
# Copyright (C) 2005 Kiseliov Roman
__rev_id__ = """$Id: unicode0.py,v 1.1 2005/03/27 12:47:06 rvk Exp $"""

if __name__ == '__main__':
    from pyExcelerator import *

    w = Workbook()
    ws1 = w.add_sheet('cp1251')

    UnicodeUtils.DEFAULT_ENCODING = 'cp1251'
    ws1.write(0, 0, 'Îëÿ')

    w.save('unicode0.xls')
```

Python Module Index

p

Index

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X