interface_Main (version v0.1.2, 2011-03-28)

index
/home/gis/Documents/interface_10032011/interface_Main.py

```
Main Modul.

This module represents the user interface and provides specific functions for
operations on data that can be executed by the user. Find more information in the documentation.
```

Modules

dateutil	numpy	sys	xml
logging	os	termios	
xml.dom.minidom	signal	time	

Classes

MainInterface

class **MainInterface**

```
Main class for data interface.

Class containing functions for operations on data by the use of this data
interface. The module 'interface_Control' provides all possible operations to this class.

COMMENTS:
Suffixes will be automatically assigned and must respect the declarations
in the module 'interface_Settings'.

IMPORTANT:
All input files of the data model must be coherent to each other and respect the
specifications. Please refer to the documentation or to additional information
in the docstrings of the specific functions in case of questions or if problems occure.
```

Methods defined here:

__init__(self, option_)

```
Constructor
```

dataModel2DataModel(self, infile_)

```
Converts data from the data model back to the data model.

This function converts data from the data model (numpy data array, coordinate metadata xml file and
NCML NetCDF XML file) back to the data model.

INPUT_PARAMETERS:
infile      - Name of data files without suffixes (string)
```

dataModel2NetCdf(self, infile_)

```
Converts data from the data model to NetCDF.

This function converts data from the data model (numpy data array, coordinate metadata xml file and
NCML NetCDF XML file) to a new NetCDF file.

INPUT_PARAMETERS:
infile      - Name of data files without suffixes (string)
```

netCdf2DataModel(self, infile_)

```
Converts one or multiple NetCDF files to the data model

This function converts one or multiple NetCDF files to the data model
(numpy data array, coordinate metadata xml file and NCML NetCDF XML file)

INPUT_PARAMETERS:
infile      - NetCDF file name without suffix (string) or that part of the
    NetCDF file name that is shared by all files (for reading multiple files).

COMMENTS:
For reading and aggregating multiple NetCDF files all files need to be similiar
expect of the time coordinate values (but need to share the same time unit).
```

netCdf2NetCdf(self, infile_)

```
Converts one or multiple NetCDF files to a NetCDF file.

This function converts one or multiple NetCDF files to a new NetCDF file.
In case that multiple NetCDF files are read, an new NetCDF file will be created
by aggregating the input data (time aggregation).
```

```
    INPUT_PARAMETERS:
    infile      - NetCDF file name without suffix (string) or that part of the
        NetCDF file name that is shared by all files (for reading multiple files).

    COMMENTS:
    For reading and aggregating multiple NetCDF files all files need to be similiar
    expect of the time coordinate values (but need to share the same time unit).
```

readModel(self, infile_)

```
    Read data from the data model with the possibility to employ operations on it.

    INPUT_PARAMETERS:
    infile      - Name of data files without suffixes (string)
```

readNetCdf(self, infile_)

```
    Read one or multiple NetCDF files with the possibility to employ operations on it.

    INPUT_PARAMETERS:
    infile      - NetCDF file name without suffix (string) or that part of the
        NetCDF file name that is shared by all files (for reading multiple files).

    COMMENTS:
    For reading and aggregating multiple NetCDF files all files need to be similiar
    expect of the time coordinate values (but need to share the same time unit).
```

test(self)

```
    Temporary test
```

utilities(self, infile_)

```
    Various utility options to modify the data model

    INPUT_PARAMETERS:
    infile      - Name of data files (data model) without suffixes (string)

    COMMENTS:
    This utilities can be called if the input operation argument is set to
    'utilities' and if the a option with its related arguments is choosen
```

Functions

POINTER(...)

addressof(...)

```
    addressof(C instance) -> integer
    Return the address of the C instance internal buffer
```

alignment(...)

```
    alignment(C type) -> integer
    alignment(C instance) -> integer
    Return the alignment requirements of a C instance
```

byref(...)

```
    byref(C instance[, offset=0]) -> byref-object
    Return a pointer lookalike to a C instance, only usable
    as function argument
```

date2num(...)

```
    date2num(dates,units,calendar='standard')

    Return numeric time values given datetime objects. The units
    of the numeric time values are described by the L{units} argument
    and the L{calendar} keyword. The datetime objects must
    be in UTC with no time-zone offset.  If there is a
    time-zone offset in C{units}, it will be applied to the
    returned numeric values.

    Like the matplotlib C{date2num} function, except that it allows
    for different units and calendars.  Behaves the same if
    C{units = 'days since 0001-01-01 00:00:00'} and
    C{calendar = 'proleptic_gregorian'}.

    @param dates: A datetime object or a sequence of datetime objects.
     The datetime objects should not include a time-zone offset.

    @param units: a string of the form C{'B{time units} since B{reference time}}'
     describing the time units. B{C{time units}} can be days, hours, minutes
     or seconds.  B{C{reference time}} is the time origin. A valid choice
     would be units=C{'hours since 1800-01-01 00:00:00 -6:00'}.

    @param calendar: describes the calendar used in the time calculations.
     All the values currently defined in the U{CF metadata convention
```

<http://cf-pcmdi.llnl.gov/documents/cf-conventions/>} are supported.
Valid calendars C{'standard', 'gregorian', 'proleptic_gregorian'
'noleap', '365_day', '360_day', 'julian', 'all_leap', '366_day'}.
Default is C{'standard'}, which is a mixed Julian/Gregorian calendar.

@return: a numeric time value, or an array of numeric time values.

The maximum resolution of the numeric time values is 1 second.

get_errno(...)

ioctl(...)

ioctl(fd, opt[, arg[, mutate_flag]])

Perform the requested operation on file descriptor fd. The operation is
defined by opt and is operating system dependent. Typically these codes are
retrieved from the fcntl or termios library modules.

The argument arg is optional, and defaults to 0; it may be an int or a
buffer containing character data (most likely a string or an array).

If the argument is a mutable buffer (such as an array) and if the
mutate_flag argument (which is only allowed in this case) is true then the
buffer is (in effect) passed to the operating system and changes made by
the OS will be reflected in the contents of the buffer after the call has
returned. The return value is the integer returned by the ioctl system
call.

If the argument is a mutable buffer and the mutable_flag argument is not
passed or is false, the behavior is as if a string had been passed. This
behavior will change in future releases of Python.

If the argument is an immutable buffer (most likely a string) then a copy
of the buffer is passed to the operating system and the return value is a
string of the same length containing whatever the operating system put in
the buffer. The length of the arg buffer in this case is not allowed to
exceed 1024 bytes.

If the arg given is an integer or if none is specified, the result value is
an integer corresponding to the return value of the ioctl call in the C
code.

main()

Main function.

This function represents the user interface and is called when the interface
program is executed. For more information about the usage execute this program
with the following statement in your shell: interface_Main.py --help

num2date(...)

num2date(times,units,calendar='standard')

Return datetime objects given numeric time values. The units
of the numeric time values are described by the C{units} argument
and the C{calendar} keyword. The returned datetime objects represent
UTC with no time-zone offset, even if the specified
C{units} contain a time-zone offset.

Like the matplotlib C{num2date} function, except that it allows
for different units and calendars. Behaves the same if
C{units = 'days since 001-01-01 00:00:00'} and
C{calendar = 'proleptic_gregorian'}.

@param times: numeric time values. Maximum resolution is 1 second.

@param units: a string of the form C{'B{time units} since B{reference time}}'
describing the time units. B{C{time units}} can be days, hours, minutes
or seconds. B{C{reference time}} is the time origin. A valid choice
would be units=C{'hours since 1800-01-01 00:00:00 -6:00'}.

@param calendar: describes the calendar used in the time calculations.
All the values currently defined in the U{CF metadata convention
<http://cf-pcmdi.llnl.gov/documents/cf-conventions/>} are supported.
Valid calendars C{'standard', 'gregorian', 'proleptic_gregorian'
'noleap', '365_day', '360_day', 'julian', 'all_leap', '366_day'}.
Default is C{'standard'}, which is a mixed Julian/Gregorian calendar.

@return: a datetime instance, or an array of datetime instances.

The datetime instances returned are 'real' python datetime
objects if the date falls in the Gregorian calendar (i.e.
C{calendar='proleptic_gregorian'}, or C{calendar = 'standard'} or C{'gregorian'}
and the date is after 1582-10-15). Otherwise, they are 'phony' datetime
objects which support some but not all the methods of 'real' python
datetime objects. This is because the python datetime module cannot
the uses the C{'proleptic_gregorian'} calendar, even before the switch
occured from the Julian calendar in 1582. The datetime instances

```
    do not contain a time-zone offset, even if the specified C{units}
    contains one.
```

pointer(...)

resize(...)
```
    Resize the memory buffer of a ctypes instance
```

set_conversion_mode(...)
```
    set_conversion_mode(encoding, errors) -> (previous-encoding, previous-errors)

    Set the encoding and error handling ctypes uses when converting
    between unicode and strings.  Returns the previous values.
```

set_errno(...)

sizeof(...)
```
    sizeof(C type) -> integer
    sizeof(C instance) -> integer
    Return the size in bytes of a C instance
```

Data

ALL_FLOATS = ['float64', 'double', 'Float64', 'f8', 'float', 'float32', 'Float32', 'f4']
ALL_INTS = ['byte', 'int8', 'i1', 'ubyte', 'UByte', 'uint8', 'u1', 'short', 'int16', 'Int16', 'i2', 'ushort', 'uint16', 'UInt16', 'u2', 'int', 'int32', 'Int32', 'integer', 'i4', ...]
BOOL = ['bool', 'Bool']
BYTE = ['byte', 'int8', 'i1']
BasicContext = Context(prec=9, rounding=ROUND_HALF_UP, Emin=-99...n, Underflow, Overflow, Clamped, DivisionByZero])
COORD_KEYWORDS = ['time', 'height', 'elev', 'depth', 'lat', 'latitude', 'lon', 'longitude', '_id']
DECLARATION_NETCDF_STATION = '_time_series'
DEFAULT_MODE = 0
DESCRIPTION = 'Data Model Interface for CEOP-AEGIS data convers...NetCDF files that respect the defined conventions'
DOUBLE = ['float64', 'double', 'Float64', 'f8']
DefaultContext = Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=...aps=[InvalidOperation, Overflow, DivisionByZero])
EPILOG = 'Author: Nicolai Holzer (E-mail: first-name dot last-name @ mailbox.tu-dresden.de)'
ExtendedContext = Context(prec=9, rounding=ROUND_HALF_EVEN, Emin=-..., Emax=999999999, capitals=1, flags=[], traps=[])
FILENAME_DEFAULT_SETTINGS_XML = 'interface_Settings.xml'
FILENAME_SUFFIX_NCML = '__ncml.xml'
FILENAME_SUFFIX_NETCDF = '.nc'
FILENAME_SUFFIX_NUMPYDATA = '__data.npy'
FILENAME_SUFFIX_NUMPYXML = '__coords.xml'
FLOAT = ['float', 'float32', 'Float32', 'f4']
GDAL_DTYPES = ['byte', 'int8', 'i1', 'short', 'int16', 'Int16', 'i2', 'ushort', 'uint16', 'UInt16', 'u2', 'int', 'int32', 'Int32', 'integer', 'i4', 'uint', 'uint32', 'UInt32', 'unsigned_integer', ...]
HEIGHT = ['height', 'elev', 'depth']
HEIGHT_UNITS = ['m', '1']
ID = ['_id']
INTEGER = ['int', 'int32', 'Int32', 'integer', 'i4']
INTERFACE_LOGGER_ROOT = 'interface'
LATITUDE = ['lat', 'latitude']
LATITUDE_UNITS = ['degrees_north']
LONG = ['long', 'int64', 'Int64', 'i8']
LONGITUDE = ['lon', 'longitude']
LONGITUDE_UNITS = ['degrees_east']
MODEL_REFERENCE_TIME_UNITS = ['hours since 1970-01-01 00:00:0.0', 'msec since 1970-01-01 00:00:0.0']
NETCDF3_DTYPES = ['byte', 'int8', 'i1', 'short', 'int16', 'Int16', 'i2', 'int', 'int32', 'Int32', 'integer', 'i4', 'float', 'float32', 'Float32', 'f4', 'float64', 'double', 'Float64', 'f8', ...]
NETCDF_FORMAT = 'NETCDF3_CLASSIC'
NUMPY_DTYPES = ['bool', 'Bool', 'byte', 'int8', 'i1', 'ubyte', 'UByte', 'uint8', 'u1', 'short', 'int16', 'Int16', 'i2', 'ushort', 'uint16', 'UInt16', 'u2', 'int', 'int32', 'Int32', ...]
ROUND_05UP = 'ROUND_05UP'
ROUND_CEILING = 'ROUND_CEILING'
ROUND_DOWN = 'ROUND_DOWN'
ROUND_FLOOR = 'ROUND_FLOOR'
ROUND_HALF_DOWN = 'ROUND_HALF_DOWN'
ROUND_HALF_EVEN = 'ROUND_HALF_EVEN'
ROUND_HALF_UP = 'ROUND_HALF_UP'

ROUND_UP = 'ROUND_UP'
RTLD_GLOBAL = 256
RTLD_LOCAL = 0
SHORT = ['short', 'int16', 'Int16', 'i2']
STRING = ['char', 'string', 'S1']
TIME = ['time']
USAGE = 'Usage: %prog [options] operation data \n[optio...model dataset respecting the defined conventions.'
U_BYTE = ['ubyte', 'UByte', 'uint8', 'u1']
U_INTEGER = ['uint', 'uint32', 'UInt32', 'unsigned_integer', 'u4']
U_LONG = ['ulong', 'uint64', 'UInt64', 'u8']
U_SHORT = ['ushort', 'uint16', 'UInt16', 'u2']
VERSION = '%prog version v0.1.2 from 2011-03-28'
__author__ = 'Nicolai Holzer'
__author_email__ = 'first-name dot last-name @ mailbox.tu-dresden.de'
__date__ = '2011-03-28'
__version__ = 'v0.1.2'
cdll = <ctypes.LibraryLoader object>
default_widgets = [<etc.progressBar.Percentage object>, ' ', <etc.progressBar.Bar object>]
environ = {'LANG': 'en_US.UTF-8', 'USERNAME': 'root', 'TER...36:*.spx=00;36:*.xspf=00;36:', 'DISPLAY': ':0.0'}
memmove = <CFunctionType object>
memset = <CFunctionType object>
pydll = <ctypes.LibraryLoader object>
pythonapi = <PyDLL 'None', handle a92918 at b752f24c>

Author

Nicolai Holzer