csv_2Interface (version v0.1.3, 2011-03-28)

index
/home/gis/Documents/interface_10032011/csv_2Interface.py

```
Converts a CSV point observation dataset to the data model.

Module for reading a CSV table file and exporting it to the data model that is
consisting of the following files: numpy data array, coordinate metadata xml file
and NCML NetCDF XML file.
Data is considered as station, therefore the shape of the output numpy array is:
(time, variable). Find more information in the documentation.
```

Modules

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

Classes

ControlModelCsv
ModelCsvRead

class **ControlModelCsv**

```
Control class for model 'ModelCsvRead'. This class is providing all available functions for reading data
```

Methods defined here:

__init__(self, infile_, option_)

```
Constructor for new control instance of specific file.

INPUT_PARAMETERS:
infile      - name of data file without filename extension (string)
option      - Parser.options arguments

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

completeDataModelManually(self)

```
Complete missing data and metadata manually
```

writeCsvMetadata(self)

```
Get metadata from the CSV file and write metadata to coordinate metadata file and
NCML XML file according to the specifications of the data interface. Function
can be called after function 'writeCsvNumpyData' was executed
```

writeCsvNumpyData(self)

```
Read CSV file and save data as numpy data array according to the specifications
of the data interface
```

class **ModelCsvRead**

```
This class contains functions to handle read operations on CSV data and is controlled by
the class 'ControlModelCsv'
```

Methods defined here:

__init__(self, infile_)

```
Constructor.

INPUT_PARAMETERS:
infile        - name of CSV file name with filename extension (string)
```

choseSpecificData(self, pCsvData_, nodata_, isVarName_)

```
Optional: Extract those information that is wanted and save it in new numpy array
```

completeDataVariables(self)

```
Complete missing data variable value modification manually

Example: Scale data values in case that units prefix have to be changed
(e.g. from hPa to Pa) due to defined unit in standard_name entry.
```

completeMetadataNcml(self)
```
Complete missing data in NCML XML file manually
```

completeMetadataNumpymeta(self)
```
Complete missing data in metadata coordinate XML file manually
```

createCsvNumpy(self, dataType_)
```
Creates a empty numpy array with same shape as CSV file and return this numpy array.
Argument dataType defines the data type of the resulting numpy array.
```

readCsvData(self, pDocCsvNumpy_, nodata_, isVarName_)
```
Save data of CSV file to previously created empty numpy array (pDocCsvNumpy).
Returns numpy array with complete CSV data. Save variable names to variable name
list if variable names are available
```

writeMetadataNcml(self, nodata_, isVarName_)
```
Create new NCML XML file according to the specifications of the data model and
complete this file by the metadata that can be extracted out of the CSV file
```

writeMetadataNumpymeta(self)
```
Create new metadata coordinate XML file according to the specifications of the data model and
complete this file by the metadata that can be extracted out of the CSV file
```

writeNumpyData(self, pNumpyData_)
```
Export numpy data array to file
```

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
program is executed. Start the program by executing it with the following
statement in your shell to get more information: csv_2Interface.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']
COORD_KEYWORDS = ['time', 'height', 'elev', 'depth', 'lat', 'latitude', 'lon', 'longitude', '_id']
CSV_DIALECT = 'excel'
DECLARATION_NETCDF_STATION = '_time_series'
DEFAULT_MODE = 0
DESCRIPTION = 'Conversion tool of CEOP-AEGIS data model for CSV table data considered as station data'
DOUBLE = ['float64', 'double', 'Float64', 'f8']
EPILOG = 'Author: Nicolai Holzer (E-mail: first-name dot last-name @ mailbox.tu-dresden.de)'
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']
MODULE_LOGGER_ROOT = 'csv'
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'
NODATA = -9999
NUMPYDATA_DTYPE = 'float32'
NUMPY_DTYPES = ['bool', 'Bool', 'byte', 'int8', 'i1', 'ubyte', 'UByte', 'uint8', 'u1', 'short', 'int16', 'Int16', 'i2', 'ushort', 'uint16', 'UInt16', 'u2', 'int', 'int32', 'Int32', ...]
RTLD_GLOBAL = 256
RTLD_LOCAL = 0
SHORT = ['short', 'int16', 'Int16', 'i2']
STRING = ['char', 'string', 'S1']
TIME = ['time']
USAGE = '%prog [options] operation data \n[options]: ...file, with or without variable names in first row'
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 v0.1.3 from 2011-03-28'
__author__ = 'Nicolai Holzer'
__author_email__ = 'first-name dot last-name @ mailbox.tu-dresden.de'
__date__ = '2011-03-28'
__version__ = 'v0.1.3'

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 434918 at a018bec>

Author

Nicolai Holzer