interface_ProcessingTools

(version v0.1.2, 2011-03-28) index
/home/gis/Documents/interface_10032011/interface_ProcessingTools.py

```
Modul with processing tools.

This module contains classes as tools for processing interface related
data and is used by the module 'interface_Model'. Furthermore this module can be
used by other modules that are intended to convert data to and fromt the data model
```

Modules

dateutil	xml.dom.minidom	time
logging	numpy	xml

Classes

- ProcessXml
  - ProcessNcml
  - ProcessNumpymeta
- ProcessingTool

class **ProcessNcml**(ProcessXml)

```
Class with functions for processing a NCML NetCDF XML file. All entries in this
file must be in accordance with the NetCdf NCML XML file schema. This class inherits
from 'ProcessXML'
```

Methods defined here:

__init__(self, xmlFileName_)
```
Constructor
```

addDimension(self, dimName_, dimLength_, dimIsUnlimited_)
```
Creates a dimension element in a NCML XML file with the name 'dimName' (string),
the lenght 'dimLength' (string) and the value 'dimIsUnlimited' (string)
in a NCML XML file unless 'dimIsUnlimited' is not set to ''
```

addGlobalAttribute(self, attrName_, attrValue_, attrType_)
```
Creates a global attribute element in a NCML XML file with the name 'attrName' (string),
the value 'attrValue' (string) and the type 'attrType' (string) in a NCML XML file
unless type is not set to ''
```

addLocalAttribute(self, posVarName_, attrName_, attrValue_, attrType_)
```
Add a local attribute with name 'attrName' (string), value 'attrValue' (string) and
type 'attrType' (string) to the existing variable with name 'posVarName' (string) in a
NCML XML file. If the type 'attrType' is set to '' there will be no entry
```

addVariable(self, varName_, varShape_, varType_)
```
Add a new variable to a NCML XML file with the name 'varName' (string),
the shape 'varShape' (string) and the type 'varType' (string)
```

changeDimension(self, posDimName_, posDimAttr_, newAttrValue_)
```
Change the attribute value of the dimensions entry 'posDimAttr' (string)
at the dimension 'posDimName' (string) with the new value 'newAttrValue' (string)
in a NCML XML file
```

changeGlobalAttribute(self, posAttrName_, posAttrType_, newAttrValue_)
```
Change a attribute value of the global attributes entry 'posAttrType' (string)
at the global attribute 'posAttrName' (string) with the new value 'newAttrValue' (string)
in a NCML XML file
```

changeLocalAttribute(self, posVarName_, posAttrName_, posAttrType_, newAttrValue_)
```
Change a local attribute value with the new value 'newAttrValue' (string)
at the entry 'posAttrType' (string) at the local attribute with name 'posAttrName'
(string) attached to the variable with name 'posVarName' (string) in a NCML XML file
```

changeMacroForStation(self)
```
Change entries of a NCML NetCDF XML macro file that was created for gridded data
to the Dapper In-situ Data Conventions (epic-insitu-1.0) so that the resulting
NetCDF file can be handled by DChart/Daper as in-situ time series data.

IMPORTANT:
-
 Data variables (in CF format of type float32) must furthermore have the attribute 'coordinates' with the value
    'time elev latitude longitude'
- A NetCDF file that is considered as sation time series data must have the filename
    suffix '_time_series.nc'
```

changeVariable(self, posVarName_, posVarAttr_, newAttrValue_)
```
Change a variables metadata value with the new value 'newAttrValue' (string)
at the entry 'posVarAttr' (string) at the variable with name 'posVarName' (string)
in a NCML XML file
```

createMacroNcmlFile(self)

```
Creates raw xml file according NetCDF Ncml XML schema, but without
data variables. Attribute values are set according to CF-1.4 convention and internal
definitions for gridded data, related to CEOP-AEGIS project.
```

fillNcmlMacroWithNumpy(self, pNumpyData_)

```
Fill metadata in NCML file by the metadata that can be extracted out of
input numpy data array (numpy array in shape of grid or station file)
```

removeLocalAttribute(self, posVarName_, posAttrName_)

```
"Remove a local attribute with name 'posAttrName'
(string) attached to the variable with name 'posVarName' (string) in a NCML XML file
```

Methods inherited from ProcessXml:

checkIfElementExists(self, posParentNode_, nameElement_)

```
Check if element (string) in parent node (string) exists and return boolean
```

createElement(self, posParentNode_, nameElement_)

```
Create new element (string) in parent node (string)
```

createEmptyXmlFile(self, namespaceURI_, qualifiedName_)

```
Creating new XML file with a namespace URI (string) and a qualified name (string)
```

printXmlOnScreen(self)

```
Print Xml file on screen by using PrettyPrint
```

readAttribute(self, posParentNode_, posNameElement_, posAttrName_)

```
Read attribute value with name 'posAttrName' (string) at element with name
'posNameElement' (string) and its parent node with name 'posParentNode' (string)
```

setAttribute(self, posParentNode_, posNameElement_, attrName_, attrValue_)

```
Set new attribute in XML file with name 'attrName' (string) and value 'attrValue'
(string) at element with name 'posNameElement' (string) and parent node 'posParentNode' (string)
```

class **ProcessNumpymeta**(ProcessXml)

```
Class with functions for processing a coordinate metadata file (numpymeta). This class inherits
from 'ProcessXML'
```

Methods defined here:

__init__(self, xmlFileName_)

```
Constructor
```

addNumpymetaValues(self, pDataNumpy_, tag_)

```
Adds values from a unidimensional numpy array to the XML coordinate
metadata file at the element 'tag'(string) with its attribute 'values' that
must already exist in the file.
```

createMacroNumpymetaFile(self)

```
Creating a XML coordinate metatadata file by the use of the following macro
```

writeNumpyMetadataValues(self, pNumpy_, tag_)

```
This function write values from a unidimensional numpy array to the XML coordinate
metadata file at the element 'tag'(string) and creates this element if it does not exist.
If data in the numpy array is equally distributed only the minimum and maximum value
will be written to the XML file, otherwise all values.
```

Methods inherited from ProcessXml:

checkIfElementExists(self, posParentNode_, nameElement_)

```
Check if element (string) in parent node (string) exists and return boolean
```

createElement(self, posParentNode_, nameElement_)

```
Create new element (string) in parent node (string)
```

createEmptyXmlFile(self, namespaceURI_, qualifiedName_)

```
Creating new XML file with a namespace URI (string) and a qualified name (string)
```

printXmlOnScreen(self)

```
Print Xml file on screen by using PrettyPrint
```

readAttribute(self, posParentNode_, posNameElement_, posAttrName_)

```
Read attribute value with name 'posAttrName' (string) at element with name
'posNameElement' (string) and its parent node with name 'posParentNode' (string)
```

setAttribute(self, posParentNode_, posNameElement_, attrName_, attrValue_)

```
Set new attribute in XML file with name 'attrName' (string) and value 'attrValue'
(string) at element with name 'posNameElement' (string) and parent node 'posParentNode' (string)
```

class **ProcessXml**

```
Class with functions for processing xml files
```

Methods defined here:

__init__(self, xmlFileName_)
```
    Constructor
```

checkIfElementExists(self, posParentNode_, nameElement_)
```
    Check if element (string) in parent node (string) exists and return boolean
```

createElement(self, posParentNode_, nameElement_)
```
    Create new element (string) in parent node (string)
```

createEmptyXmlFile(self, namespaceURI_, qualifiedName_)
```
    Creating new XML file with a namespace URI (string) and a qualified name (string)
```

printXmlOnScreen(self)
```
    Print Xml file on screen by using PrettyPrint
```

readAttribute(self, posParentNode_, posNameElement_, posAttrName_)
```
    Read attribute value with name 'posAttrName' (string) at element with name
    'posNameElement' (string) and its parent node with name 'posParentNode' (string)
```

setAttribute(self, posParentNode_, posNameElement_, attrName_, attrValue_)
```
    Set new attribute in XML file with name 'attrName' (string) and value 'attrValue'
    (string) at element with name 'posNameElement' (string) and parent node 'posParentNode' (string)
```

class **ProcessingTool**

```
Class containing frequently used functions by the module 'interface_Model' that
is designed so that it can also be used by other modules
```

Methods defined here:

checkDapperTimeSeriesFilename(self, infile_)
```
    Check if a filename already ends with constant 'DECLARATION_NETCDF_STATION'
    and attaches this suffix if this is not the case
```

checkNumpyEqualDataDistribution(self, pInNumpy_)
```
    Check if values in a numpy array area equally distributed

    This function compares the intervals between values of a numpy array. If
    at least one of these intervals differentiates the data is not equally
    distributed. If it is equally distribute it can for example be represented
    with a minimum and maximum value as well as a value defining the number of values

    INPUT_PARAMETERS:
    pInNumpy        - Numpy array with one dimensions

    RETURN_VALUE:
    True if equally distributed, False if not
```

checkUdunitsUnit(self, unit_)
```
    Check if a unit is conform to the Udunits2 library.

    COMMENT:
    The program code of this function was adapted from the program 'cfchecks.py', version 2.0.2,
    written by Rosalyn Hatcher (Met Office, UK).

    IMPORTANT:
    Udunits2 must have been installed correctly with correct settings in this function as
    well as correct settings of the global Udunits constants in 'interface_Settings.py'.
```

convertBool(self, inBool_)
```
    Converts an input boolean in the form of a string to a Python boolean
```

createEvenlySpacedNumpy(self, min_, max_, quantity_, dataType_)
```
    Create numpy array with evenly spaced values.

    This function creates a numpy array with evenly spaced values based on a minimum
    and maximum value as well as a value defining the number of values to generate and
    a data type

    INPUT_PARAMETERS:
    min         - Minimum value, first value in numpy array (float)
    max         - Maximum value, last value in numpy array (float)
    quantity    - Number of values to generate (integer)
    dataType    - Numpy data type defining the output data (numpy dtype)

    RETURN_VALUE:
    Numpy array with evenly spaced values
```

createTimeValuesNumpy(self, units_, quantity_, timeStep_)
```
    Creates numpy array with time values

    This function creates a one dimensional numpy array with time values defined by
    a reference time, a unit and a starting date, a number of values and a multiplicator
    for timedelta.

    INPUT_PARAMETERS:
    units           - Time unit and starting date of values (string) in the form of
        'unit since reference time' (e.g. 'hours since 1970-01-01 00:00:0.0'). The
    following unit values are allowed: 'days', 'hours', 'minutes', 'seconds'
quantity        - Duration, number of values to generate from the starting date (integer)
timeStep        - Repeat interval, as multiplicator for timedelta (float), for example '0.5' (hours)

RETURN_VALUE:
Numpy array (one dimension) with time values

COMMENTS:
The calculated time values in the numpy array have its reference date defined
in the constant 'self.pDefaultSettings.varTimeAttrUnits' (module interface_Contants)
```

dataType_2Gdal(self, intype_)

```
Converts an input data type string to the related gdal data type string
```

dataType_2NetCdf(self, intype_)

```
Converts an input data type string to the related NetCDF data type string
```

dataType_2Numpy(self, intype_)

```
Converts an input data type string to the related numpy dtype data type string
```

list2String(self, inList_, oldSeparator_, newSeparator_)

```
Converts a list to a string by the use of an input list and a separator. Furthermore the separator
can be changed to a new separator for the string (e.g. ',' replaced by ' ')
```

scaleNumpyDataVariable(self, pNumpyData_, varNr_, scaleFactor_)

```
Scale values of data variable #varNr in data variable array
pNumpyData with value scaleFactor. varNr must be of same type as
numpy data array
```

string2List(self, inString_, separator_)

```
Converts a string to a list
```

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(...)

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']
DECLARATION_NETCDF_STATION = '_time_series'
DEFAULT_MODE = 0
DOUBLE = ['float64', 'double', 'Float64', 'f8']
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', ...]
RTLD_GLOBAL = 256
RTLD_LOCAL = 0
SHORT = ['short', 'int16', 'Int16', 'i2']
STRING = ['char', 'string', 'S1']
TIME = ['time']
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']
__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>
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 d4d918 at b7408cec>

Author

Nicolai Holzer