Data Request Python API

01.beta.37

Martin Juckes, October 11th, 2016

Executive Summary

The Data Request is presented as two XML files whose schema is described in a separate document. A python module is provided to facilitate use of the Data Request. Some users may prefer to work directly with the XML file or with spreadsheets and web page views, but this software provides some support for those who want to use a programming approach.

Objectives

The python API is designed to provide intuitive access to the complete collection of information.

New in version 01.beta.27

- Problems with volume estimate found and resolved;
- Support for separate supplement added.

Overview

The basic module provides two objects, the first of which contains the full information content. The 2nd provides some indexing arrays to facilitate navigation through the request.

Installation

Download code from subversion

The module is currently a simple script to be kept in the working directory.

```
svn co  http://proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/01.beta.24
# or: svn co  http://proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/latest
cd dreqPy
python simpleCheck.py
```

PyPi repository

The package is available from the test python repository at https://testpypi.python.org/pypi/dreqPy/01.beta.24

To install as user (watch the command response to see where `pip` places the package):

```
pip install -i https://testpypi.python.org/pypi --user  dreqPy==01.beta.24
```

or, with administrator privileges:

```
pip install -i https://testpypi.python.org/pypi dreqPy==01.beta.24
```

Requirements

python 2.6.6, 2.7 or 3.x.

The core modules of the package only uses core python modules:

```
xml, string, re, collections, shelve, sys, os
```

The software runs significantly faster in python 3.x.

In the makeTables.py there is a dependency on xlsxwriter, but this is only required to reproduce a spreadsheet which is already distributed with the package.

Usage

The box shows a piece of sample code to print a list of all the variables defined in the “var” section.

The content: dq.coll

The content object, dq.coll is a dictionary whose elements correspond to the data request sections represented as a “named tuple” of 3 elements: “items” (a list of records), “header” (a named tuple – see below) and “attDefn” (a dictionary with record attribute definitions).

```
from dreqPy import dreq
dq = dreq.loadDreq()
print dq.coll.keys()
print dq.coll['var'].attDefn.keys()
print dq.coll['var'].header.title
print '_'*len( dq.coll['var'].header.title )
print '%20s: %s [%s]' %  (
       tuple( [dq.coll['var'].attDefn[a].title for a in
       ['label','title','units']] )  )
for r in  dq.coll['var'].items[:10]:
   print '%20s: %s [%s]' % (r.label,r.title,r.units)
```

e.g. dq.coll['var'].items[0] is the first item in the “var” section.

The items are instances of a family of classes described below. The “label” etc are available as attributes, e.g. dq.coll['var'].items[0].label is the label of the first record.

dq.coll['var'].attDefn['label'] contains the specification of the “label” attribute from the configuration file. This is also available from the item object itself as, for example, dq.coll['var'].items[0]._a.label.

The following code box shows how this can be used to generate an overview of the content, printing a sample record from each section, using the “title” of each attribute.

```
from dreqPy import dreq
dq = dreq.loadDreq()
for k in sorted( dq.coll.keys() ):
  x = dq.coll[k].items[0]
  for k1 in sorted( x.__dict__.keys() ):
    if k1[0] != '_':
      print '%32s: %s' % (x._a[k1].title, x.__dict__[k1] )
```

dq.coll['CMORvar'].attDefn['vid'].rClass[1] = 'internalLink': this value indicates that the "vid" attribute of records in the "CMORvar" section is an internalLink and so must match the "uid"[2] attribute of another record. To find that record, see the next section.

The index: dq.inx

The index is designed to provide additional information to facilitate use of the information in the data request.

`dq.inx.uid` is a simple look-up table: `dq.inx.uid[thisId]` returns the record corresponding to "thisId". This is a change from the previous release, in which this dictionary returned a tuple with the name of the section as first element. The name of the section is now available through the "_h" attribute of the record (see next section).

`dq.inx.iref_by_uid` gives a list of the IDs of objects which link to a given object, these are returned as a tuple of section name and identifier.

`dq.inx.iref_by_sect` has the same information organised differently: `dq.inx.iref_by_sect[thisId].a['CMORvar']` is a list of the IDs of all the elements in 'CMORvar' which link to the given element.

There are also dictionaries for each section indexed by label and, if relevant, CF standard name.

- dq.inx.var['tas'] will list the IDs of records with label='tas';
- dq.inx.var.sn['air_temperature'] give a list of records with standard name 'air_temperature'.

The record object

As noted above, each section contains a list of items. Each item within a section is an instance of the same class. The classes are generated from a common base class (dreqItemBase), but carry attributes specific to each section.

A summary readable summary of a record content can be obtained through the __info__ method. For example,

```
>>> i = dq.coll['experiment'].items[0]
>>> i._h.__info__()
```

yields:

```
Item <Experiments>: [histALL] __unset__
    nstart: 1
    yps: 171
    starty: 1850.0
    description: * Enlarging ensemble size of the CMIP6 hisorical simulations (2015-2020
under SSP2-4.5 of ScenarioMIP) to at least three members. * DCPP: DCPP proposes a 10
member ensemble of histALL up to 2030 also extended with SSP2-4.5. * Please provide
output data up to 2014 as "CMIP6 historical" and 2015-2020 (or 2030 for DCPP) as SSP2-
4.5 of ScenarioMIP.
    title: __unset__
    endy: 2020.0
    ensz: 2
    label: histALL
   egid: [exptgroup]Damip1 [a684ca9a-8391-11e5-bca6-0f460b96c0cb]
    tier: 1
   mip: [mip]DAMIP [DAMIP]
    ntot: 342
```

[1] Python objects cannot, unfortunately, have attributes with names matching python keywords, so the "class" attribute from the XML document is mapped onto rClass in the pythom API.

[2] "uuid" has been replaced with the more general "uid" for "Unique identifier". Identifiers will still be unique within the document, but will not necessarily follow the uuid specifications.

```
    mcfg: AOGCM/ESM
    comment:
    uid: a684c950-8391-11e5-bca6-0f460b96c0cb
```

Information about the section and the attributes of records in the section can be obtained through the "_h" and "_a" attributes. For example:

```
>>> i = dq.coll['experiment'].items[0]
>>> i._h.__info__()
Item <X.3 Section Attributes>: [experiment] 1.5 Experiments
   uid: SECTION:experiment
   level: 0
   title: 1.5 Experiments
   id: exp
   useClass: vocab
   maxOccurs: 1
   label: experiment
   itemLabelMode: def
   labUnique: No
sectdef(tag=u'table', label=u'experiment', title=u'Experiments', id=u'cmip.drv.012', itemLabelMode=u'def',
level=u'0')
```

Note that in earlier versions the "_h" object was a named tuple, whereas it now has the same basic structure as the record object.

Records to define record attributes (new since 01.beta.11)

Each record contains a collection of attributes with names such as "title", "tier". More information about the usage of each attribute is contained in another record which is attached to the parent class. In the above example, for instance, the value of "i.tier" is 1, the specification of the "tier" attribute is in "i.__class__.tier", which is also a record object so that "i.__class__.tier.__info__()" yields the following:

```
Item <Core Attributes>: [tier] Tier of experiment
    uid: __unset__
    title: Tier of experiment
    techNote: None
    label: tier
    superclass: __unset__
    useClass: None
    type: xs:integer
    description: Experiments are assigned a tier by the MIP specifying the tier,
tier 1 experiments being the most important.
```

and, because the "tier" object has the same methods as the "i" object, "i.__class__.tier.__class__.type.__info__() " yields:

```
Item <Core Attributes>: [type] Record Type
    uid: __core__:type
    title: Record Type
    techNote:
    label: type
    superclass: rdfs:range
    useClass: __core__
    type: xs:string
    description: The type specifies the XSD value type constraint, e.g.
xs:string.
```

This formulation, which embeds all the information, including the definitions of attributes, in the same structure is motivated by the structure of Resource Description Framework (RDF) triples. In RDF and object is defined through a set of triples of the form "object property subject", with the important constraint the "property" must be an RDF object. In the dreqPy implementation the “property” object for “tier” is the record “i.__class__.tier” and the RDF triple is expressed as “i.tier=1”.

This feature provides the mechanism for making the API self-documenting. At present there are many attributes which have little or no information in the record “description”, but this will be filled out in coming revisions.

The header record for each item is now also an item record with the same structure. The command “i._h.__info__()”, where “i” is a record from the “experiment” section as above, yields:

```
Item <Section Attributes>: [experiment] Experiments
    uid: SECTION:experiment
    title: Experiments
    useClass: vocab
    label: experiment
    id: cmip.drv.012
```

Scope.py

An additional module has been added to provide volume estimates. The current draft demonstrates how information can be aggregated, and the basic mechanism for avoiding duplication when multiple MIPs ask for the same data.

The following code will set “x” to the volume, expressed as and estimate of the the number of floating point values, for the C4MIP request with variables up to priority 2:

```
from dreqPy import scope
sc = scope.dreqQuery()
x = sc.volByMip( 'C4MIP', pmax=2 )
```

The conversion to bytes will depend on the choice of compression, which is not yet represented in the API. The volume for multiple MIPs is obtained passing a python set to volByMip, e.g.

```
x = sc.volByMip( {'C4MIP', 'LUMIP'}, pmax=2 )
```

An example is provided in “example.py”.

After a call to `sc.volByMip`, the variable `sc.indexedVol` contains a breakdown of the volume by frequency and the CMOR name of the variable. E.g. `sc.indexedVol['mon'].a['snc']` contains the volume associated with the monthly snow cover data.

The estimate uses a default model configuration. To reset this, change the values in the sc.mcfg dictionary (this part of the module will be improved to support use of a configuration file):

- nho: number of horizontal mesh points in the ocean;
- nlo: number of vertical levels in ocean;
- nha: number of horizontal mesh points in the atmosphere;
- nla: number of vertical levels in atmosphere;
- nlas: number of vertical levels in stratosphere;
- nls: number of levels in soil model;
- nh1: number of latitude points.

The example.py script demonstrates use of the scope.py module to generate volume estimates for three endorsed MIPs individually and in combination (to run this, simple type “python example.py” at the command line).

Supplement

By default the supplement (currently containing suggested ranges for some variables) is not loaded. To load the supplement use:

```
dq = dreq.loadDreq( manifest='out/dreqManifest.txt' )
print len( dq.coll['qcranges'].items )
```

The additional section, “qcranges”, read in from the supplement will then be appended to the other sections.

dreqCmdl.py

A command line interface has been added. From the source directory this can be used as follows:

```
python3 dreqCmdl.py -m HighResMIP -t 1 -p 1 --printVars
--printLinesMax 20
```

With the “--printVars” and “--printLinesMax” arguments the command will print the most significant variables by volume.

Selection by Tier of experiments

The scope.py module now supports selection of experiments by tier. A call of the following form will configure the “sc” object to consider only experiments with tiers up to, and including, tierMax:

`sc.setTierMax( tierMax )`

Intersection of request

An option has been added to support the evaluation of the intersection of requests:

```
python3 dreqCmdl.py –intersection -m HighResMIP,DynVar
--printVars
```

will evaluate the intersection of the HighResMIP and DynVar requests.

Variables required for an experiment

The API will now generate a list of requested variables, in an excel spreadsheet, for any specified combination of MIPs. For this option to work the python “xlsxwriter” package must be installed.

```
drq -m HighResMIP,C4MIP -e historical –xls -p 2
```

The above command will list all variables up to priority 2 requested for the “historical” experiment by either HighResMIP or C4MIP. The results will be placed in “xls/c4.hi-historical_1_1.xlsx”. If inspecting output requested for an experiment with tier greater than 1, the default tier cutoff must also be changed using “-t 3” in the argument string.

Selection by Objectives

All data requested is associated with specific scientific objectives. In some cases the MIPs have specified multiple objectives with different data requirements and modelling groups may elect to provide data only for a selection of objectives.

```
drq -m  RFMIP:RapidAdjustment.AerosolIrf,HighResMIP:Ocean --xls
```

The command in the box below will produce a list of variables required to support the RFMIP "Rapid Adjustment" and "Aerosol Instantaneous Forcing" objectives and the HighResMIP "Ocean" objective. This gives a data volume estimate of 3Tb, compared to 30Tb if all objectives of HighResMIP and RFMIP are supported.

Specifying model grid sizes for volume estimation

The package uses a set of default model grid sizes to estimate the data volumes:

Label	Description	Default
nho	Number of horizontal grid cells in ocean grid	259200
nlo	Number of vertical levels in ocean grid	60
nha	Number of horizontal grid cells in atmosphere grid (also used for land surface fields).	64800
nla	Number of vertical levels in atmosphere grid	40
nlas	Number of vertical levels in the stratosphere (used for a small set of variables)	20
nls	Number of vertical levels in the soil model	5
nh1	Number of latitudes (used for transects and zonal means in both atmosphere and ocean at present, so as to give these fields a non-zero volume).	100

The first four of these, specifying the size of the ocean and atmosphere grids, are the most relevant for volume estimation. The last 3 have little impact. They can be reset as follows:

```
drq -m HighResMIP,C4MIP --mcfg 259200,60,64800,45,20,5,100
```

The above command resets the number of vertical levels in the atmosphere to 45.

Improved volume estimation

```
drq -m HighResMIP,C4MIP --sf –xls -p 2
```

The is a new variant of the volume estimation.

- Better handling of the potential conflicts arising from variables being requested on multiple grids through an explicit Grid Policy logic (see below);
- Produces an additional spreadsheet which shows the breakdown of the volume by frequency and variable shape;

This feature uses a cleaner internal work-flow, so the results, which can differ from those provided by previous versions of the API by a few percent, should be more reliable. The old logic is still supported to enable further testing of the new code.

Grid Policy

The phrase “Grid Policy” is used here to refer to the decisions taken when there are a number of possible options regarding the grid used to store global fields. Prior to 01.beta.37 there were no options in the command line interface. Now there are two parameters which can be set:

--grdpol: “native”, “1deg” (default) or “2deg”: the type of grid to be used for ocean data when no preference has been expressed by any of the MIPs requesting the data.

--allgrd: (on if flag present, off by default): if on, all requested grids for each variable are included. if off, only the highest resolution data (assuming “native” is higher than “1deg”) is preserved.

Setting “--grdpol native” results in a significant increase in data volume relative to the new default. These options only take effect when the “--sf” flag is set (see above).

The grid policy options may be adjusted in the future to reflect priorities set by the WIP, meanwhile, these options make it possible to explore the consequences of various choices.

Important caveats

- A list of issues related to the content of the XML document is given in dreqML.pdf
- There is still a significant amount of duplication within the CMOR variable section of the data request;
- Some variables are listed as choices (e.g. supply either on 4 or 7 pressure levels), but this option needs to be made an explicit part of the schema so that it can easily and reliably be picked up in the API. A sample of variables are now subject to ranked selection. The API needs to be improved to make this feature more visible.