Data Request Python API

Martin Juckes, September 28th, 2015

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.

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

The module is currently a simple script to be kept in the working directory. It will be packaged for pypi in the near future.

```
svn co  http://proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/01.beta.02
cd src
python simpleCheck.py
```

Usage

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

```
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)
```

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).

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

The items are named tuples again, so that 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.

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. Information about the section and the attributes of records in the section can be obtained through the “_h” and “_a” attributes. For eaxmple:

```
>>> i = dq.coll['experiment'].items[0]
>>> print i._h
sectdef(tag=u'table', label=u'experiment', title=u'Experiments', id=u'cmip.drv.012', itemLabelMode=u'def',
level=u'0')
```

A summary readable summary of a record content can be obtained through the __info__ method. For example, after creating the “i” variable as above, “i.__info__()” yields:

```
Item <Experiments>: [NTCFRESPbc] __unset__
    nstart: 1
    yps: 41
    starty: 2015.0
    description: Perturbation: Only black carbon emissions as in NTCFRESP-SSP3-7ntcf
    title: __unset__
    endy: 2055.0
    ensz: 1
    label: NTCFRESPbc
   egid: [exptgroup]Aerchemmip3b [d653f206-6293-11e5-af00-5404a60d96b5]
    tier: 1
    mip: AerChemMIP
     ntot: 41
```

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: AGCM
comment:
uid: d653f4c2-6293-11e5-af00-5404a60d96b5
```

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:

```
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.

Important caveats

- In the present draft there is no support for selection by Tier of experiment, or by objective: these will be added at the next iteration.
- The removal of duplication works by checking back to the requested CMOR variables. There is still a significant amount of duplication within the CMOR variable section of the data request.
- Estimates of experiment duration are used – the link between request items and experiment specifications needs to be improved.

Outlook

The current version demonstrates the core functionality. Extending the functionality of the API will depend primarily on cleaning up the content. Support for more complex queries of the form “CMOR variables which link to MIP variables with standard name 'precipitation_flux'” will be added. At some point the API will also be put behind a web service.