Data Request Python API

01.00.27

Martin Juckes, July 26th, 2018

Table of Contents

1 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.

2 Introduction

2.1 Objectives

To provide intuitive access to the complete collection of information held in the data request document.

2.2 New in this version

- extensions/versions.py added to support handling multiple versions;
- version information extended to include installation directory.

2.3 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.

3 Installation

3.1 PyPi repository

The package is available from the test python repository at
https://pypi.python.org/pypi/dreqPy/01.00.27

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

```
pip install -i https://pypi.python.org/pypi --user  dreqPy==01.00.27
```

or, with administrator privileges:

```
pip install -i https://pypi.python.org/pypi dreqPy==01.00.27
```

3.2 Download code from subversion

The code is kept in a subversion repository, with a tag for each release.

```
svn co  http://proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/01.00.27
# or: svn co  http://proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/latest
cd dreqPy
python simpleCheck.py
```

After extracting the code, the script “simpleCheck.py” will run a few basic checks and report any problems.

3.3 Requirements

The following python versions are supported: 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 command line tool there is a dependency on xlsxwriter, which is used to write tables of variables.

3.4 Getting started

Version

To print the version number:

```
drq -v
```

Help

```
drq -h
```

Tests

To run some test, checking a range of package modules.

```
drq --unitTest
```

Import

To load the library in a python session:

```
from dreqPy import dreq
```

See section 5 for further details.

4 Directories and Environment Variables

The XML files for the data request version current at the time of the package build are stored in the “/docs” subdirectory of the directory holding the package (this directory path is printed out in the response to the “drq -v” command in version 01.00.27 and later).

The python package can be used with XML files from compatible versions. The environment variable DRQ_VERSION_DIR can be used to specify a directory to hold different versions of the XML document. If this environment variable is unset, the package will default to using "<HOME>/.dreqPy". XML files for a given version should be stored in "<HOME>/.dreqPy/<version>".

5 Command line interface

Once the package is installed, it can be invoked with the "drq" command. It can also be invoked from the directory containing the source code using "python dreqCmdl.py <args>", using the same arguments as for "drq", as described below.

5.1 Obtaining a list of variables requested by a MIP

```
drq -m HighResMIP -t 1 -p 1 --printVars --printLinesMax 20
-e historical --xls
```

This command will provide a list of variables requested by HigResMIP for the CMIP6 historical experiment, and provide them in a spreadsheet in the "xls" directory. If the shorter form "drq -m HighResMIP" is used, the software will print out a volume estimate and not provide the list of variables.

The "-t" and "-p" arguments refer to the tier of experiments and priority of variables, described in section 4.3 below. If omitted, they both default to unity.

The "--printVars" and "--printLinesMax" arguments can be used to generate a print-out of the largest variables by estimated volume. This is only intended as a summary. More complete information is provided in the spreadsheets.

The "-e" argument takes the name of an experiment, or of an endorsed MIP, or "CMIP". If an endorsed MIP name is given, the code will provide results for experiments defined by that MIP. For example, "drq -m C4MIP -e ScenarioMIP" provides information on the data requested by C4MIP from experiments defined by ScenarioMIP. If "-e CMIP" is used, the information is provided for the DECK and CMIP6 historical experiments.

By default. the output includes the data requested specifically be the endorsed MIP identified in the argument and the core request. To omit the latter, see section 4.2.

5.2 The Core Request, Multiple MIPs and Selection by Objective

To obtain information about multiple MIPs, simply list them, separated by commas:

```
drq -m C4MIP,LS3MIP,LUMIP -t 1 -p 1 --printVars
--printLinesMax 20 -e historical --xls
```

By default, the output includes the core request which is requested from all models participating in CMIP6. To omit this, add the argument "--omitCmip".

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.

5.3 Selection by Tier of Experiments and Priority of Variables

Within the request, each experiment is assigned a Tier, and requests for variables all have priorities. In CMIP5, a priority was assigned to each variable, but in CMIP6 variables may have different significance for different MIPs, so the priority is assigned for each request of a variable.

There is also a dependency of the variables requested on the tier: LS3MIP has a request for 3-hourly data which should cover the whole length of the historical simulation if their tier 2 experiments are being performed, but only a limited time slice if only tier 1 is being performed.

5.4 Intersection of request

An option has been added to support the evaluation of the intersection of requests: the following command will evaluate the intersection of the HighResMIP and DynVar requests, i.e. the variables requested by both MIPs:

```
drq --intersection -m HighResMIP,DynVar  --printVars
```

5.5 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

Table 1: Model configuration

The first four of these, specifying the size of the ocean and atmosphere grids, are the most relevant for volume estimation, since they determine the shape and dimension of the largest requested diagnostics.The last 3 have little impact. The following command, for example, resets the number

of vertical levels in the atmosphere to 45:

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

5.6 Legacy volume estimation

```
drq -m HighResMIP,C4MIP --legacy --xls -p 2
```

Earlier versions of the code used a different set of internal routines to generate volume estimates and variable lists. There were some problems dealing with the logic of grid selection, so the code was reconfigured. The old routines can still be accessed using the "--legacy" flag, as above.

5.7 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 have no effect when the "--legacy" 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.

5.8 Spreadsheet formats

The data request tool produces two categories of spreadsheets:

- variable lists: files with names starting "cmv";
- volume estimate workings: files with names starting "requestVol".

File names are of the form <stem>_<id1>_<id2>_<tier>_<priority>[_<sfx>].xlsx

- stem: "cmvmm" (variable list for experiments requested by a MIP), "cmvme" (variable list for experiment), "cmvume" (variable list which are requested by one MIP but not by others). These are extended with "fr" if the option "--xfr" is used to generate sheets sorted by frequency and realm;
- id1: the requesting MIP or, if more than one MIP is involved, a string formed using the first two letters of each MIP, e.g. "cm.ls.lu" for CMIP, LS3MIP and LUMIP, or "TOTAL" for the aggregate of requests by all MIPs;
- id2: the experiment or MIP specifying a group or experiments, or "TOTAL" for an aggregate listing of all variables requested across all experiments;
- tier: the maximum tier set in the query;
- priority: the maximum priority set in the query;
- sfx: the suffix is added if variations on the default grid selection option are used:

- "fn" if "--grdforce native" is used (all data to be put on native grid),
- "fl" if "--grdforce 1deg", and
- "dn" if "--grdpol native" is used (use preference expressed by requesting MIPs, when present, otherwise use the native grid).
- Default: use the preference expressed by requesting MIPs, when present, otherwise use 1 degree grid.

```
drq -m HighResMIP,C4MIP --xmlVersion 01.00.21
```

Variable lists

The variable lists are modelled on the CMIP5 standard output spreadsheet. If the command is invoked for multiple MIPs and experiments, there will be a file for the aggregated list of variables, as well as files for each individual MIP and experiment.

Additional columns are added to list the MIPs requesting each variable, and the MIPs defining the experiment each variable is requested for (these columns are more useful in the aggregated files – they contain little useful information in files corresponding to a single MIP and a single experiment).

For files corresponding to a single MIP and experiment, there are an additional 4 columns specifying the time period needed for each variable and the grid interpolation request. Three columns for the time slice information give the number of years, and type of slice used to specify this and a list of years. The fourth column specifies a grid, of the form "native", "native:01" (a variation on "native" used by OMIP: data should be on the native grid unless it is unstructured, in which case it should be interpolated to regular), "1deg", "2deg". The value that appears here will depend on the "grid policy" (see section 4.7).

The default is to provide variables listed in MIP tables.

Alternatively, variables can be listed by realm and frequency by using the "--xfr" argument.

The variable list include structured information about each variable, including frequency and the CF cell methods attribute specifying any averaging that needs to be done.

Volume Estimate Workings

The volume estimate sheet includes a table with rows for each spatial configuration of data, and a block of 4 columns for each frequency. The 4 columns include a count of variables, the average number of years per variable, and the number of experiments, and finally a volume estimate based on this information. The names of the variables and experiments are included as comments. Sums over rows and columns give the volume associated with each spatial shape and each frequency respectively.

5.9 XML versions

To get results for a different document version:

```
drq -m HighResMIP,C4MIP --xmlVersion 01.00.21
```

The XML documents for version 01.00.21 must be placed in the directory specified by environment variable DRQ_VERSION_DIR (see section 4 above). The extensions package provides some tools to help with the management of multiple versions (section 7 below).

6 Python Library

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

6.1 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.

6.2 The index: dq.inx

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

---

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.

`dq.inx.uid` is a simple look-up table: `dq.inx.uid[thisId]` returns the record corresponding to “thisId”. This is a change from early beta releases, 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[thisId]` 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'.

6.3 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]._h.__info__()” will yield:

```
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
    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.

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

Each record contains a collection of attributes with names such as "title", "tier": the collection of attributes will be used to define the building blocks of the request. 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
```

6.5 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.volByMip2( '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.volByMip2( {'C4MIP', 'LUMIP'}, pmax=2 )
```

An example is provided in "example.py".

After a call to `sc.volByMip2`, the variable `sc.res['vu']` contains a breakdown of the volume by frequency and the CMOR name of the variable. E.g. `sc.indexedVol['mon'].['Omon.thetao']` contains the volume associated with the potential temperature.

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) with keys given by the labels in Table 1 (section 4.5).

6.6 Selection by Tier of Experiments and Priority of Variables

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 ).`

7 Extensions

In version 01.00.26 an extensions subpackage has been added. This takes some functionality from scope.py and attaches it to the objects of the core package, increasing flexibility and useability. **The methods are introduced here to explore the flexibility offered by this approach, but note that the API of these new features may change.** To avoid negative impact on the existing API, these extensions are omitted by default.

Method names have been modified between 01.00.26 and 01.00.27 to avoid problems caused by the double underscore and the start of names (python has a name mangling feature aimed at enabling a degree of privacy in variables which are declared with an initial double underscore).

7.1 The collect module

To add, execute the additional bold lines below:

```
from dreqPy import dreq
dq = dreq.loadDreq()
from dreqPy.extensions import collect
collect.add( dq )
```

This adds methods "_get__CMORvar" to records in the "mip", "requestLink" records, and "_get__expt" to records in "mip", "requestLink" and "requestItem" records. There is also a dictionary "_labelDict" added to 'var','experiment','timeSlice','mip','spatialShape', and 'structure'

records. The following code will then get a list of variables requested by C4MIP, and a list of experiments they are requesting data from:

```
c4mip = dq.coll['mip'].items[0]._labelDict['C4MIP']
vars = c4mip._get__CMORvar()
expts = c4mip._get__expt()
```

The methods take arguments "tierMax" to filter on tiers (omitting experiments with tier greater than tierMax) and "pmax" to filter on priority.

7.2 The versions module

```
from dreqPy import dreq
dq = dreq.loadDreq()
from dreqPy.extensions import collect, versions
collect.add( dq )
versions.add( dq )
```

This module adds two methods, *_fetch_* and *_conpare_*. The *_compare_* method relies on a dictionary which is created by the collect module, hence the includion of the collect.add() function call in the above.

fetch

The fetch method will fetch XML documents for a given version and save them in the directory specified by DRQ_VERSION_DIR (or the default: see section 4).

```
dq._fetch_('01.00.21')
```

compare

The compare method does a simple comparison between two versions.

```
from dreqPy import dreq
dq = dreq.loadDreq()
from dreqPy.extensions import collect, versions
collect.add( dq )
versions.add( dq )
dqb = dreq.loadDreq( xmlVersion='01.00.21')
collect.add(dqb)
dq._compare_(dqb,sect="var", lst="short")
```

The command sequence shown will print information about the records which are new in the current version, compared to version 01.00.21, and list the number of attributes changed. A shorter report is obtained with lst=None, and a longer report, giving the names of all records that have changed, is given by lst="long". As noted above, the API for these functions in the **extensions** section is likely to change.

8 Miscellaneous

8.1 Examples

The "examples" folder in the repository contains a series of example scripts. The document

"dreqExamples.pdf"[3] explains the examples, which cover both use of the python API and alternative approaches using the XML database directly.

8.2 Important caveats

- A list of issues related to the content of the XML document is given in dreqML.pdf[4]
- The request for the DECK and CMIP6 historical experiments is well reviewed, but the specifications for other experiments still need to be finalised.
- There are cases where the request contains two versions of a variable, and the data provider should select which one to provide. These choices can be explicitly listed through "varChoice" records in the database, but the content is currently incomplete.
- The qcRanges section contains information on expected data ranges: this section is still incomplete.

9 Appendix: command line arguments

```
Data Request Command line.
------------------------
      -v : print version and exit;
      --unitTest : run some simple tests;
      -m <mip>:  MIP of list of MIPs (comma separated; for objective selection
see note [1] below);
      -l <options>: List for options:
              o: objectives
              e: experiments
      -q <options>: List information about the schema:
              s: sections
              <section>: attributes for a section
              <section:attribute>: definition of an attribute.
      -h :       help: print help text;
      -e <expt>: experiment;
      -t <tier> maxmum tier;
      -p <priority>  maximum priority;
      --xls : Create Excel file with requested variables;
      --sf : Print summary of variable count by structure and frequency
[default];
      --legacy : Use legacy approach to volume estimation (deprecated);
      --xfr : Output variable lists in sheets organised by frequency and realm
instead of by MIP table;
      --SF : Print summary of variable count by structure and frequency for all
MIPs;
      --grdpol <native|1deg> :  policy for default grid, if MIPs have not
expressed a preference;
      --grdforce <native|1deg> :  force a specific grid option, independent of
individual preferences;
      --ogrdunstr : provide volume estimates for unstructured ocean grid
(interpolation requirements of OMIP data are different in this case);
      --allgrd :  When a variable is requested on multiple grids, archive all
grids requested (default: only the finest resolution);
      --unique :  List only variables which are requested uniquely by this MIP,
for at least one experiment;
      --esm :  include ESM experiments (default is to omit esm-hist etc from
volume estimates);
      --txt : Create text file with requested variables;
      --mcfg : Model configuration: 7 integers, comma separated,
'nho','nlo','nha','nla','nlas','nls','nh1'
                default: 259200,60,64800,40,20,5,100
```

3 proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/trunk/dreqPy/docs/dreqExamples.pdf

4 proj.badc.rl.ac.uk/svn/exarch/CMIP6dreq/tags/latest/dreqPy/docs/dreqML.pdf

```
      --txtOpts : options for content of text file: (v|c)[(+|-)att1[,att2[...]]]
      --xlsDir <directory> : Directory in which to place variable listing [xls];
      --printLinesMax <n>  : Maximum number of lines to be printed (default 20)
      --printVars    : If present, a summary of the variables (see
--printLinesMax) fitting the selection options will be printed
      --intersection : Analyse the intersection of requests rather than union.

NOTES
-----
[1] A set of objectives within a MIP can be specified in the command line. The
extended syntax of the "-m" argument is:
-m <mip>[:objective[.obj2[.obj3 ...]]][,<mip2]...]

e.g.
drq -m HighResMIP:Ocean.DiurnalCycle
```