dbnomics for gretl

Allin Cottrell Jack Lucchetti

June 26, 2022

1 Introduction

This package offers an interface to dbnomics for gretl. For anyone who hasn't yet caught on, dbnomics makes available, in a uniform manner, a huge number of macroeconomic data series drawn from many sources around the world—a truly admirable service!

The dbnomics website can be found at https://db.nomics.world/; interested users are encouraged to visit the site to get a better sense of what's available. The exact mechanisms whereby dbnomics makes data available are still under development so some changes can be expected in future. We will endeavor to keep our package up to date and will push out updates with gretl snapshots as required.

There are three main layers to the dbnomics "space," as follows:

- *Providers*: the various statistical agencies that are the primary sources of the data. As of this writing 62 providers are included, from the African Development Group to the WTO.
- *Datasets*: sets of related series offered by a given provider. Over 20,000 datasets are available.
- *Series*: specific time series such as the Bulgarian unemployment rate. Hundreds of millions of series are available.

A specific series in dbnomics is identified by a triplet of the form `provider/dataset/series`, for example

```
ECB/IRS/M.IT.L.L40.CI.0000.EUR.N.Z
```

Here the provider is `ECB` (the European Central Bank); the dataset is `IRS` (interest rate statistics); and the particular series is "`M.IT.L.L40.CI.0000.EUR.N.Z`," an Italian 10-year interest rate.

This package provides means of downloading a specific series if you know its identifying triplet, and also means of navigating the dbnomics space. There are three ways of accessing the functionality of the package:

- Via the gretl commands `open` and `data`, as with native gretl databases.
- By means of the gretl GUI.
- By calling the public functions of the package yourself, in command-line or scripting mode.

The following three sections expand on these methods in turn.

2 The `open` and `data` commands

To exploit this method you need to know the identifying triplet(s) for the series you want. Given that, you can initiate dbnomics access via the command

```
open dbnomics
```

From this point the `data` command will target dbnomics data until you "`open`" some other data source. So, for example, you could download the Italian 10-year interest rate mentioned above in this way:

```
data ECB/IRS/M.IT.L.L40.CI.0000.EUR.N.Z
```

Ah, but what about the name of the series within gretl? The full triplet is obviously not acceptable as a gretl series-name, and even its third component won't work since gretl identifiers cannot contain the dot character. What happens by default is that gretl takes the third portion of the triplet and squeezes out any illegal characters. In the case above this would give a name of `MITLL40CI0000EURNZ`—not very nice-looking. However, you can take charge of the naming of the imported series yourself, using the `--name` option to the `data` command, as in

```
data ECB/IRS/M.IT.L.L40.CI.0000.EUR.N.Z --name="IT_10yr"
```

In this case the series will be known to gretl as `IT_10yr`. Note that when you import a series from dbnomics its descriptive "label" starts with the full triplet so you won't lose that information of record, regardless of the naming of the series.

Note that the `data` command takes care automatically of several details, such as for example matching date or adjusting for different periodicities. For example, the following code snippet will create an empty monthly dataset and retrieve from dbnomics the monthly index of industrial production for Portugal, as reported by the "International Financial Statistics" dataset by the International Monetary Fund.

```
nulldata 220
setobs 12 2007:1
open dbnomics
data IMF/IFS/M.PT.AIP_IX --name="pt_ip"
```

The `nulldata` and `setobs` commands used in the above script will generate a dataset that starts in January 2007 and ends in April 2018, whereas the downloaded series (at the time of this writing) starts in January 1955 and ends in December 2017, but the `data` command puts the numbers in the right place. As for periodicities, suppose we import the same series into a quarterly dataset, such as AWM17:

```
open AWM17.gdt --quiet
open dbnomics
data IMF/IFS/M.PT.AIP_IX --name="pt_ip"
```

In this case, the series is also compacted to a lower frequency by averaging, which is the default behavior of the `data` command (see the *Gretl User's Guide* for more details).

3 dbnomics via the gretl GUI

GUI access to dbnomics is provided in two places (see Figure 1):

- Via the Databases item under the File menu in the gretl main window.
- From the gretl databases window (opened by the database icon on the toolbar at the foot of the main window): click the "DB" icon on the toolbar in this window.

In both cases you get a little sub-menu with entries "Browse" and "Specific series." The latter entry takes you to a dialog box in which you can enter a series triplet (see section 2). The Browse entry takes you to a window which displays the available dbnomics providers: their codes and their descriptions (see Figure 2). From here you have two options:

- Select a provider and double-click to browse the datasets it supplies.

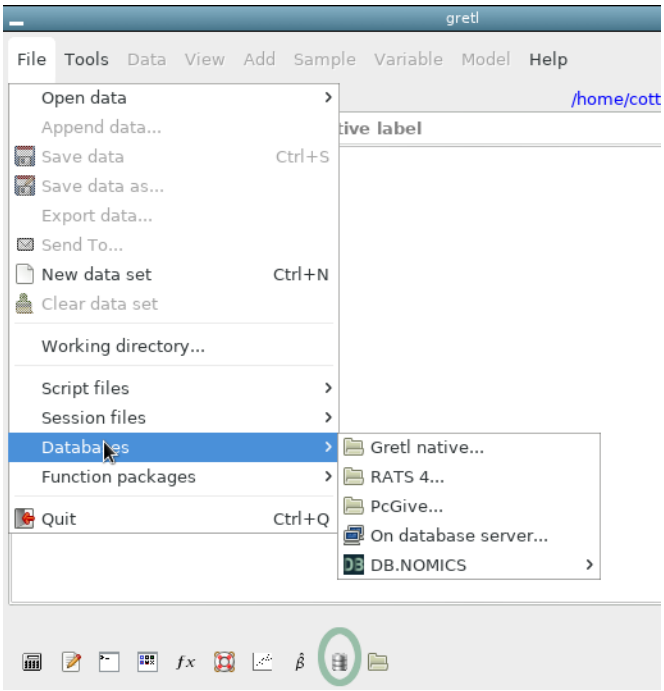

Figure 1: DB.NOMICS access via the File menu or the databases button (circled)

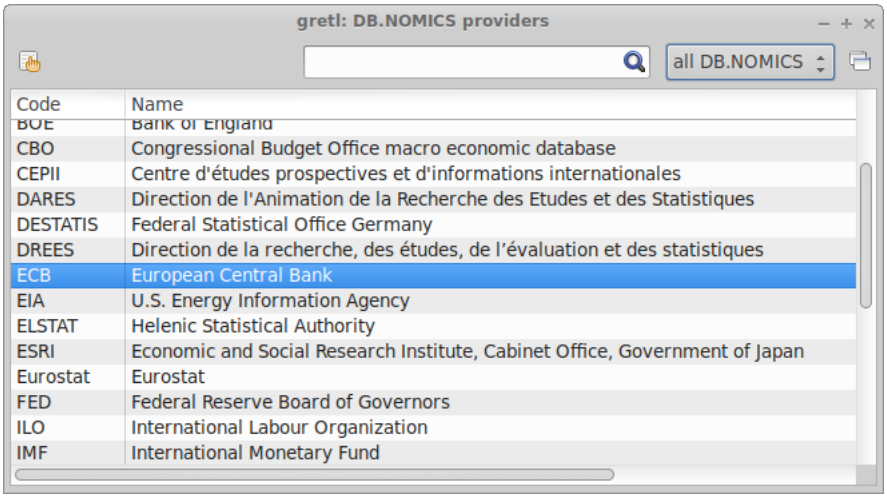

Figure 2: Listing of DB.NOMICS providers

- In the search box in the top panel of the window, enter a string and search all providers for datasets that match your specification.[1]

In each case a dbnomics datasets window will open. With some providers or searches more datasets will be found than can comfortably be displayed at once. In that case the toolbar includes buttons that let you page forward or back through the listing; you should also get a status message at the foot of the window indicating the current position in the listing.

From a datasets window two more steps are available:

- Double-click on a dataset to open a window showing the series it contains (or one "page" of its full list of series if there are too many).
- In a window showing dbnomics series, double-click to activate a particular series. This will give you detailed information on the series and allow you to display its values, create a time-series plot, or add the series to your gretl dataset.

To summarize, there are three layers to gretl's GUI representation of the dbnomics space:

1. The providers window (with global search facility)
2. Datasets window (for a given provider, or via search)
3. Series window (for a given dataset)

4 Public functions

The package contains several public functions which both subserve the modes of access described in sections 2 and 3, and can be called directly by the user.

At this point we just offer an "as is" listing of the signatures of these functions with brief commentary. The finer points are subject to change; we can expand on them later if there's sufficient interest.

However, the general guiding principle is that, when you download some information from dbnomics, be it a single series or more complex objects, the metadata are going to be as important as the data themselves. Therefore, in most cases what you get from the functions provided by this package are bundles, or arrays of bundles.

For example, the `dbnomics_get_series` function takes as its first mandatory argument a series code and returns a bundle containing both the data and the metadata: take the series `Q.AU.C.A.M.USD.A`, from the "long series on total credit" dataset, itself from the Bank for International Settlements. The code for the dataset is `BIS/CNFS`. Therefore, this code snippet

```
b = dbnomics_get_series("BIS/CNFS/Q.AU.C.A.M.USD.A")
print b
```

produces the following:

```
bundle b, created by dbnomics:
  frequency = 4
  series_name (string, 127 bytes)
  dimensions (bundle)
  dataset_name = long series on total credit
  period = array of strings, length 120
  error = 0
  series_code = Q.AU.C.A.M.USD.A
  indexed_at = 2018-12-13T17:38:15.031Z
```

[1] If you wish to use this facility to find a string in the providers window itself, first select "this window" to the right of the search box, the default being "all DB.NOMICS" as shown in Figure 2.

```
  dataset_code = CNFS
  provider_code = BIS
  has_data = 1
  period_start_day = array of strings, length 312
  T = 120
  @frequency = quarterly
  value (matrix: 120 x 1)
```

The actual data are stored as a column vector, under the key `value`; however, there is much more information available to you: for example, the `frequency` key equals 4, thus indicating that the data are quarterly, and so on. If you want to have this information printed in a more readable way, you'll want to use the `dbnomics_bundle_print` function, that yields (long lines broken for readability):

```
Series: Q.AU.C.A.M.USD.A
Provider: BIS
Dataset: CNFS (long series on total credit)
Identifier: BIS/CNFS/Q.AU.C.A.M.USD.A
Name: Quarterly - Australia - Non financial sector - Adjusted for
  breaks - Market value - US Dollar - Adjusted for breaks
Dimensions:
  Frequency: 'Q' (Quarterly)
  Borrowers' country: 'AU' (Australia)
  Borrowing sector: 'C' (Non financial sector)
  Lending sector: 'A' (Adjusted for breaks)
  Valuation: 'M' (Market value)
  Type of adjustment: 'A' (Adjusted for breaks)
  Unit type: 'USD' (US Dollar)
------------------------
pd = 4; 120 observations, 1988-Q2 - 2018-Q1
```

Some functions can be used for retrieving multiple series at once; therefore, what they return is an *array* of bundles. The following example fetches two series for two countries from the "AMECO" dataset as provided by the European Central Bank.

```
set verbose off
include dbnomics.gfn

bundle spec = null
spec.AME_ITEM = defarray("UBLGE", "OVGD")
spec.AME_REF_AREA = defarray("AUT", "BEL")
bs = dbnomics_get_multiple("ECB", "AME", 20, 0, spec)
dbnomics_bundles_print(bs)
```

Once the series have been downloaded, a short description of the resulting array of bundles is obtained by the `dbnomics_bundles_print` (note the plural) function, and this is the output (with long lines broken for readability, again):

```
Contents of bs:

     Provider  Code                    Description
  1: ECB/AME   A_AUT_1_0_0_0_OVGD      Austria - Gross domestic product at 2...
                                       61 observations (pd = 1) [1960:2020]
  2: ECB/AME   A_AUT_1_0_319_0_UBLGE   Austria - Net lending (+) or net borr...
                                       26 observations (pd = 1) [1995:2020]
  3: ECB/AME   A_BEL_1_0_0_0_OVGD      Belgium - Gross domestic product at 2...
                                       61 observations (pd = 1) [1960:2020]
  4: ECB/AME   A_BEL_1_0_319_0_UBLGE   Belgium - Net lending (+) or net borr...
                                       26 observations (pd = 1) [1995:2020]
```

Note that this package contains several test scripts that exemplify calls to the functions listed below; this can be found in the `examples` subdirectory of the installation directory. Likely locations for this are as follows (though the paths may differ by locale and otherwise):

Linux	`/usr/share/gretl/functions/dbnomics`
Windows	`C:\Program Files\gretl\functions\dbnomics`
Mac	`/Applications/Gretl.app/Contents/Resources/share/gretl/functions/dbnomics`

List of public functions (in alphabetical order)

```
scalar dbnomics_bundle_get_data (const bundle b,
                                 series *x,
                                 bool verbose[0])
```

Given a bundle obtained by `dbnomics_get_series`, writes the actual data values (and description) into the series `x`, which must exist already and be given in "pointer" form. Returns zero on success, non-zero on error.

```
# example
bundle b = dbnomics_get_series("ECB/IRS/M.IT.L.L40.CI.0000.EUR.N.Z")
series IT_10yr = NA
dbnomics_bundle_get_data(b, &IT_10yr)
```

```
void dbnomics_bundle_print (const bundle b,
                            bool print_data[0])
```

Displays the content of a bundle obtained by `dbnomics_get_series`. Give a non-zero value for the second argument to print the actual values, otherwise just the metadata is shown.

```
void dbnomics_bundles_print (const bundles bs)
```

Displays a concise description of the series contained in an array of bundles, such as the ones you get from function like `dbnomics_get_multiple`. For a more detailed printout of the individual bundles, use `dbnomics_bundle_print`.

```
list dbnomics_bundles_to_list(bundles bs, string key[null])
```

Creates a list of series from an array of bundles containing series data, as returned by functions such as `dbnomics_get_multiple`. By default the names of the series will be constructed automatically but the second, optional string argument can be used to impose a chosen naming scheme: for each bundle, if it contains a string value under the specified `key`, that string will be used as the name of the corresponding series.

```
bundle dbnomics_category_tree (const string provider,
                               bool verbose[0])
```

Returns a bundle containing a representation of the "category tree" for the specified provider. For some providers this tree just amounts to a list of datasets but for others it is a hierarchy in which related datasets are grouped under one or more levels of headings. Each bundle in the tree will have members `code` and `name`; those that represent groups of datasets rather then datasets proper will in addition have a `children` member.

```
# example
bundle b = dbnomics_category_tree("BLS")
print b --tree
```

```
bundle dbnomics_dsets_for_provider (const string provider,
                                    bool verbose[0])
```

Returns a bundle containing basic information on the datasets associated with a given provider, namely two arrays of strings holding the codes and names of the datasets respectively.

```
# example
bundle b = dbnomics_dsets_for_provider("AMECO")
```

```
series dbnomics_fetch (const string datacode,
                       bool verbose[0])
```

This is just a convenience wrapper for `dbnomics_get_series` followed by `dbnomics_bundle_get_data`.

```
bundles dbnomics_get_cart (const string URL)
```

This is a convenience function that you can use to select the series you want via the "cart" facility provided by the DB.nomics website. After choosing the series you want, you have to select the "Copy API link" entry in the "Download" menu.

At that point, you can just paste the result into a gretl string, and use that as the argument of this function. See the file `get_cart_example.inp` for an example.

```
bundles dbnomics_get_dataset_content (const string provider,
                                      const string dset,
                                      int limit[0::100],
                                      int offset[0])
```

Returns an array of bundles each containing information on a series contained in the dataset specified by the `provider` and `dset` codes. The `limit` and `offset` arguments allow "paging": retrieve so many results, starting at a given offset into the full listing.

```
# example
bundles B = dbnomics_get_dataset_content("ECB", "IRS", 50, 100)
```

```
bundles dbnomics_get_dataset_dimensions (const string provider,
                                         const string dset,
                                         bool verbose[0])
```

Returns an array of bundles, with all the "dimensions" for a given dataset, and prints it out if the `verbose` argument is nonzero. The dimensions typically contain lists of the different periodicities of the series contained in the datasets, the geographical units they refer to, and so on.

Therefore, each resulting bundle will have a key called `code`, which identifies the dimension, and an array of bundles called `values`, describing each dimension via the keys `code` and `label`. For example, the following code

```
set verbose off
include dbnomics.gfn

dims = dbnomics_get_dataset_dimensions("ECB", "AME")
code = dims[2].code
vals = dims[2].values
printf "%s\n\n", code
loop i = 1..4 --quiet
    printf "%s - %s\n", vals[i].code, vals[i].label
endloop
```

returns

```
AME_REF_AREA

AUT - Austria
BEL - Belgium
BGR - Bulgaria
HRV - Croatia
```

Note: This may not work with some providers.

```
bundles dbnomics_providers (bool verbose[0])
```

Returns an array of bundles, one per provider. Each bundle contains basic info about the provider, notably its dbnomics code under the `code` key and its full name under the `name` key.

```
bundles dbnomics_get_multiple (const string provider,
                               const string dset,
                               int limit[0::50],
                               int offset[0],
                               bundle spec[null])
```

Returns an array of bundles (defaulting to a maximum of 50), each of which contains information (data + metadata) on a series from dataset `provider/dset`.

The bundle `spec`, if present, can be used to limit the query to certain dimensions. There are two ways to to this:

- you may put a `mask` key into the bundle, which contains a string specially tailored to the specifics of that particular dataset. For example, the string "`Q.FR+DE+BE.PCPIFBT_IX`" in the context of the `IMF/CPI` dataset, corresponds to the quarterly price indices for "Alcoholic Beverages, Tobacco, and Narcotics" in France, Germany and Belgium (see the example file `get_multiple_example_via_mask`);

- alternatively, you may put into the `spec` bundle one or more string array with the "dimensions" for that dataset (see the example file `get_multiple_example`). In order to find the dimensions available for a given dataset, use the function `dbnomics_get_dataset_dimensions()`.

```
bundle dbnomics_get_series (const string datacode,
                            bool verbose[0])
```

Returns a bundle containing information on the series specified by `datacode`, which must be a dbnomics triplet as described above.

```
# example
bundle b = dbnomics_get_series("ECB/IRS/M.IT.L.L40.CI.0000.EUR.N.Z")
```

The `verbose` switch, if true, prints out the actual URL that the function sends to the dbnomics website, and can be used for debugging purposes.

```
bundles dbnomics_search (const string key,
                         const string dset[null],
                         int limit[0::100],
                         int offset[0],
                         bool verbose[0])
```

The behavior of this function is dictated by the second parameter `dset`.

If `dset` is null, or an empty string, the function returns an array of bundles, each holding information on a dataset which matches (in some way or other) the `key` string. If, conversely, `dset` contains a valid dataset representation (eg "`AMECO/ZUTN`"), then the query will be limited to that particular dataset, and the bundles returned will contain the series matching the query. See section 5 below for further details.

The `limit` and `offset` argument should in principle work to allow paging but as of this writing the `offset` argument has no effect due to a dbnomics bug.

```
# example
bundles B = dbnomics_search("interest rates", null, 50)
```

5 Searching dbnomics

Given the vast size of the dbnomics space, it's important to have effective search tools. This is work in progress, but in this section we illustrate the current state of play. The example below shows a two-stage search. We first search for relevant datasets across the whole population of providers then we home in on a particular dataset and search for relevant series, in each case requesting verbose results.

```
set verbose off
include dbnomics.gfn

# target of search
key = "remittances Iraq"

# search all providers for up to 10 relevant databases
bundles generic = dbnomics_search(key, null, 10, 0, 1)

# search the "WDI" dataset of the World Bank for up to
```

```
# 10 relevant series
dataset_code = "WB/WDI"
bundles specific = dbnomics_search(key, dataset_code, 10, 0, 1)
```

The example produces the following output (long lines broken for readability):

```
Datasets containing "remittances Iraq" (1-5 of 5):

  1: Eurostat.bop_rem6 (42 series)
  2: WB.WDI (5 series)
  3: IMF.BOP (54 series)
  4: CEPII.BOP (54 series)
  5: ECB.BOP (54 series)

Dataset WB/WDI, matching series 1-5 of 5:

BM.TRF.PWKR.CD.DT-IQ: Personal remittances, paid (current US$) -- Iraq
BX.TRF.PWKR.CD.DT-IQ: Personal remittances, received (current US$) -- Iraq
BX.TRF.PWKR.DT.GD.ZS-IQ: Personal remittances, received (% of GDP) -- Iraq
SI.RMT.COST.IB.ZS-IQ: Average transaction cost of sending remittances
  to a specific country (%) -- Iraq
SI.RMT.COST.OB.ZS-IQ: Average transaction cost of sending remittances
  from a specific country (%) -- Iraq
```

The same search facilities are also available through the GUI: if you go back to Figure 2, you will notice a search text box at the top. By default, any term you insert will trigger a search for that term on the whole dbnomics space, like the `dbnomics_search` function with a `null` second argument. If you want to restrict the search to a particular dataset instead, you will have to "click to" that particular data set and use the search text box there, like in Figure 3.

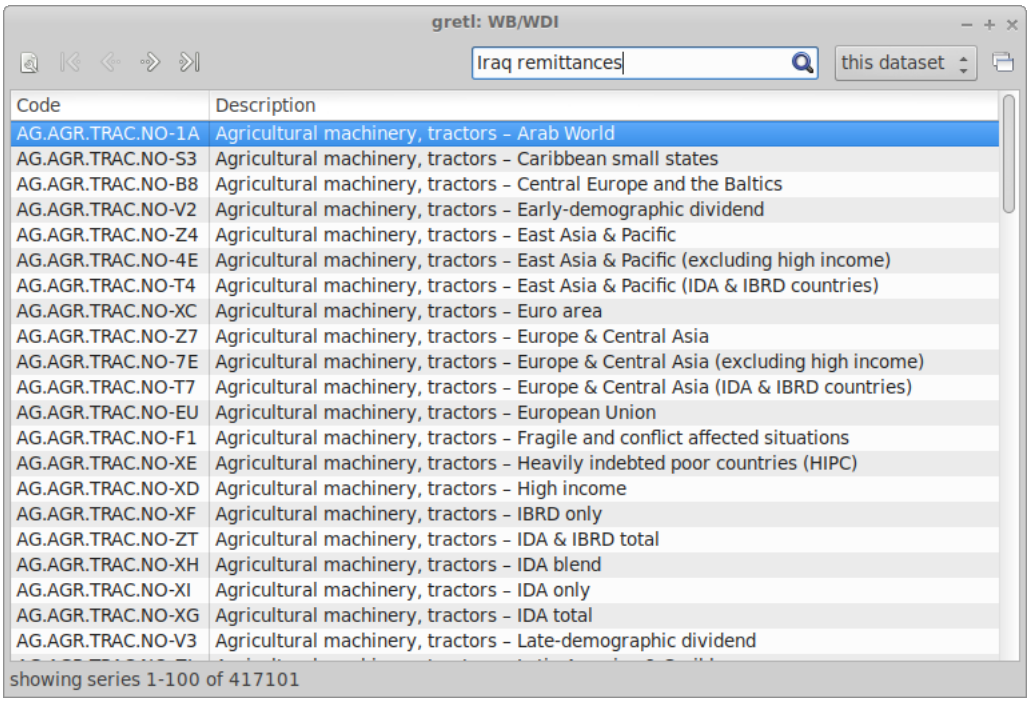

Figure 3: Search within a particular dataset

6 Change log

We show below a brief history of changes in the gretl dbnomics package. Details can be found at https://sourceforge.net/p/gretl/git/ci/master/tree/addons/dbnomics/.

2022-06-26	ensure deletion of temporary data files
2022-01-12	update for absence of `series_name`
2021-10-21	fix handling of quarterly data with gaps
2021-03-29	cut out waste of time downloading metadata
2021-03-10	update for presence of metadata switch in API
2021-01-06	update for absence of `complete_missing_periods`
2020-10-22	update for absence of `dimensions_labels` in many datasets
2020-02-27	add `dbnomics_printer` function
2020-01-25	add the `commit_missing_periods` flag to our dbnomics requests to ensure we get a full data calendar
2019-07-01	more work on handling nested JSON arrays
2019-06-26	work around nested arrays in dbnomics "dimensions" info
2019-03-03	another fix in light of API switch
2019-02-28	fix minor breakage due to API switch
2019-01-17	update dbnomics URL
2018-12-23	switch to version 22 of dbnomics API
2018-11-28	support downloading of multiple series bundles
2018-06-27	initial entry as gretl addon