wbpy Documentation

Release 1

Matthew Duck

March 13, 2016

Contents

NOTE: This project is not under active development.

The tests are still passing at time of writing (2016/03/16), but if the APIs have changed much over the last couple of years then it's possible that there are some rough edges or missing features.

I'm happy to look at any opened issues, but I'm no longer using the library, so I won't be finding issues myself. I'm also happy to accept pull requests, and to provide commit access (or to transfer ownership) if anybody wants to pick this up.

(This file was built from an IPython Notebook. Download README.ipynb on Github to poke around.)

A Python interface to the World Bank Indicators and Climate APIs.

- Readthedocs
- Github source
- World Bank API docs

The Indicators API lets you access a large number of world development indicators - country data on education, environment, gender, health, population, poverty, technology, etc.

The Climate API lets you access modelled and historical data for temperature and precipitation.

Why use wbpy?

- Dataset models let you access processed data and associated metadata in different formats.
- If you don't want processed data objects, you can still access the raw JSON response.
- Single method calls to do the equivalent of multiple API requests, eg. wbpy handles the specific date pairs which would otherwise be required for the Climate API.
- Works with both ISO 1366 alpha-2 and alpha-3 country codes (the web APIs mostly just support alpha-3).

Elsewhere, there is also wbdata, a wrapper for the Indicators API which supports Pandas structures and has some command-line functionality.

CHAPTER 2

Installation

`pip install wbpy`, or download the source code and `python setup.py install`.

CHAPTER 3

Indicators API

3.1 Basic use

Here's a small case where we already know what API codes to use:

```
import wbpy
from pprint import pprint

api = wbpy.IndicatorAPI()

iso_country_codes = ["GB", "FR", "JP"]
total_population = "SP.POP.TOTL"

dataset = api.get_dataset(total_population, iso_country_codes, date="2010:2012")
dataset
```

```
<wbpy.indicators.IndicatorDataset(u'SP.POP.TOTL', u'Population, total') with id: 203402188>
```

The `IndicatorDataset` instance contains the direct API response and various metadata. Use `dataset.as_dict()` to return a tidy dictionary of the data:

```
dataset.as_dict()
```

```
{u'FR': {u'2010': 65031235.0, u'2011': 65371613.0, u'2012': 65696689.0},
 u'GB': {u'2010': 62271177.0, u'2011': 62752472.0, u'2012': 63227526.0},
 u'JP': {u'2010': 127450459.0, u'2011': 127817277.0, u'2012': 127561489.0}}
```

Some examples of the metadata available:

```
dataset.api_url
```

```
'http://api.worldbank.org/countries/GBR;FRA;JPN/indicators/SP.POP.TOTL?date=2010%3A2012&format=json&p
```

```
dataset.indicator_name
```

```
u'Population, total'
```

```
dataset.indicator_topics
```

```
[{u'id': u'8', u'value': u'Health '},
 {u'id': u'19', u'value': u'Climate Change'}]
```

```
dataset.countries
```

```
{u'FR': u'France', u'GB': u'United Kingdom', u'JP': u'Japan'}
```

If you want to create your own data structures, you can process the raw API response:

```
dataset.api_response
```

```
[{u'page': 1, u'pages': 1, u'per_page': u'10000', u'total': 9},
 [{u'country': {u'id': u'FR', u'value': u'France'},
   u'date': u'2012',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'65696689'},
  {u'country': {u'id': u'FR', u'value': u'France'},
   u'date': u'2011',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'65371613'},
  {u'country': {u'id': u'FR', u'value': u'France'},
   u'date': u'2010',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'65031235'},
  {u'country': {u'id': u'GB', u'value': u'United Kingdom'},
   u'date': u'2012',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'63227526'},
  {u'country': {u'id': u'GB', u'value': u'United Kingdom'},
   u'date': u'2011',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'62752472'},
  {u'country': {u'id': u'GB', u'value': u'United Kingdom'},
   u'date': u'2010',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'62271177'},
  {u'country': {u'id': u'JP', u'value': u'Japan'},
   u'date': u'2012',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'127561489'},
  {u'country': {u'id': u'JP', u'value': u'Japan'},
   u'date': u'2011',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'127817277'},
  {u'country': {u'id': u'JP', u'value': u'Japan'},
   u'date': u'2010',
   u'decimal': u'0',
   u'indicator': {u'id': u'SP.POP.TOTL', u'value': u'Population, total'},
   u'value': u'127450459'}]]
```

3.2 Searching for indicators

We don't always know what indicators we want to use, so we can search:

```
population_indicators = api.get_indicators(search="population")
len(population_indicators)
```

```
1180
```

Ah. That's not a very manageable number. The API returns over 8000 indicator codes, and lots of them have "population" in the title. Luckily, most of those indicators don't really have much data, so we can forget about them. You can browse the indicators with the best data coverage at http://data.worldbank.org/indicator, and you can pass `common_only=True` to throw away all indicators that aren't included on that page:

```
population_indicators = api.get_indicators(search="population", common_only=True)
print "There are now only %d indicators to browse!" % len(population_indicators)
```

```
There are now only 61 indicators to browse!
```

We don't want to print that many results in the documentation, so let's filter some more. The API query string parameters are directly mapped to kwargs for each method. For the `get_indicators` method, this means we can filter by topic or source:

```
health_topic_id = 8
health_indicators = api.get_indicators(search="population", common_only=True, topic=health_topic_id)
print "We've narrowed it down to %d indicators!" % len(health_indicators)
```

```
We've narrowed it down to 18 indicators!
```

Each indicator has a variety of metadata:

```
pprint(health_indicators.items()[0])
```

```
(u'SN.ITK.DEFC.ZS',
 {u'name': u'Prevalence of undernourishment (% of population)',
  u'source': {u'id': u'2', u'value': u'World Development Indicators'},
  u'sourceNote': u'Population below minimum level of dietary energy consumption (also referred to as
  u'sourceOrganization': u'Food and Agriculture Organization, The State of Food Insecurity in the Wo
  u'topics': [{u'id': u'8', u'value': u'Health '}]})
```

That data might be useful, but it's not very friendly if you just want to grab some API codes. If that's what you want, you can pass the results to the `print_codes` method:

```
api.print_codes(health_indicators)
```

```
SH.CON.1524.FE.ZS          Condom use, population ages 15-24, female (% of females ages 15-24)
SH.CON.1524.MA.ZS          Condom use, population ages 15-24, male (% of males ages 15-24)
SH.DYN.AIDS.FE.ZS          Women's share of population ages 15+ living with HIV (%)
SH.DYN.AIDS.ZS             Prevalence of HIV, total (% of population ages 15-49)
SH.MLR.NETS.ZS             Use of insecticide-treated bed nets (% of under-5 population)
SH.STA.ACSN                Improved sanitation facilities (% of population with access)
SH.STA.ACSN.RU             Improved sanitation facilities, rural (% of rural population with acce
SH.STA.ACSN.UR             Improved sanitation facilities, urban (% of urban population with acce
SN.ITK.DEFC.ZS             Prevalence of undernourishment (% of population)
SP.POP.0014.TO.ZS          Population ages 0-14 (% of total)
SP.POP.65UP.TO.ZS          Population ages 65 and above (% of total)
SP.POP.1564.TO.ZS          Population ages 15-64 (% of total)
SP.POP.DPND                Age dependency ratio (% of working-age population)
SP.POP.DPND.OL             Age dependency ratio, old (% of working-age population)
SP.POP.DPND.YG                 Age dependency ratio, young (% of working-age population)
SP.POP.GROW                    Population growth (annual %)
SP.POP.TOTL                    Population, total
SP.POP.TOTL.FE.ZS              Population, female (% of total)
```

There are `get_` functions matching all API endpoints (countries, regions, sources, etc.), and the `search` parameter and `print_codes` method can be used on any of them. For example:

```
countries = api.get_countries(search="united")
api.print_codes(countries)
```

```
AE                             United Arab Emirates
GB                             United Kingdom
US                             United States
```

3.3 More searching

If you're not sure what to search for, just leave out the `search` parameter. By default, the `get_` methods return all API results:

```
all_regions = api.get_regions()
all_sources = api.get_sources()

print "There are %d regions and %d sources." % (len(all_regions), len(all_sources))
```

```
There are 32 regions and 28 sources.
```

The `search` parameter actually just calls a `search_results` method, which you can use directly:

```
pprint(api.search_results("debt", all_sources))
```

```
{u'20': {u'description': u'', u'name': u'Public Sector Debt', u'url': u''},
 u'22': {u'description': u'',
         u'name': u'Quarterly External Debt Statistics (QEDS) - Special Data Dissemination Standard
         u'url': u''},
 u'23': {u'description': u'',
         u'name': u'Quarterly External Debt Statistics (QEDS) - General Data Dissemination System (GI
         u'url': u''},
 u'6': {u'description': u'',
        u'name': u'International Debt Statistics',
        u'url': u''}}
```

By default, the `search` parameter only searches the title of an entity (eg. a country name, or source title). If you want to search all fields, set the `search_full` flag to `True`:

```
narrow_matches = api.get_topics(search="poverty")
wide_matches = api.get_topics(search="poverty", search_full=True)

print "%d topic(s) match(es) 'poverty' in the title field, and %d topics match 'poverty' in all fiel
```

```
1 topic(s) match(es) 'poverty' in the title field, and 7 topics match 'poverty' in all fields.
```

3.4 API options

All endpoint query string parameters are directly mapped to method kwargs. Different kwargs are available for each `get_` method (documented in the method's docstring).

- **language:** `EN`, `ES`, `FR`, `AR` or `ZH`. Non-English languages seem to have less info in the responses.
- **date:** String formats - `2001`, `2001:2006`, `2003M01:2004M06`, `2005Q2:2005Q4`. Replace the years with your own. Not all indicators have monthly or quarterly data.
- **mrv:** Most recent value, ie. `mrv=3` returns the three most recent values for an indicator.
- **gapfill:** `Y` or `N`. If using an MRV value, fills missing values with the next available value (I think tracking back as far as the MRV value allows). Defaults to `N`.
- **frequency:** Works with MRV, can specify quarterly (`Q`), monthly (`M`) or yearly (`Y`). Not all indicators have monthly and quarterly data.
- **source:** ID number to filter indicators by data source.
- **topic:** ID number to filter indicators by their assigned category. Cannot give both source and topic in the same request.
- **incomelevel:** List of 3-letter IDs to filter results by income level category.
- **lendingtype:** List of 3-letter IDs to filter results by lending type.
- **region:** List of 3-letter IDs to filter results by region.

If no date or MRV value is given, **MRV defaults to 1**, returning the most recent value.

Any given kwarg that is not in the above list will be directly added to the query string, eg. `foo="bar"` will add `&foo=bar` to the URL.

3.5 Country codes

`wbpy` supports ISO 1366 alpha-2 and alpha-3 country codes. The World Bank uses some non-ISO 2-letter and 3-letter codes for regions, which are also supported. You can access them via the `NON_STANDARD_REGIONS` attribute, which returns a dictionary of codes and region info. Again, to see the codes, pass the dictionary to the `print_codes` method:

```
api.print_codes(api.NON_STANDARD_REGIONS)
```

```
1A                              Arab World
1W                              World
4E                              East Asia & Pacific (developing only)
7E                              Europe & Central Asia (developing only)
8S                              South Asia
A4                              Sub-Saharan Africa excluding South Africa
A5                              Sub-Saharan Africa excluding South Africa and Nigeria
A9                              Africa
C4                              East Asia and the Pacific (IFC classification)
C5                              Europe and Central Asia (IFC classification)
C6                              Latin America and the Caribbean (IFC classification)
C7                              Middle East and North Africa (IFC classification)
C8                              South Asia (IFC classification)
C9                              Sub-Saharan Africa (IFC classification)
EU                              European Union
JG                              Channel Islands
```

```
KV                              Kosovo
M2                              North Africa
OE                              OECD members
S1                              Small states
S2                              Pacific island small states
S3                              Caribbean small states
S4                              Other small states
XC                              Euro area
XD                              High income
XE                              Heavily indebted poor countries (HIPC)
XJ                              Latin America & Caribbean (developing only)
XL                              Least developed countries: UN classification
XM                              Low income
XN                              Lower middle income
XO                              Low & middle income
XP                              Middle income
XQ                              Middle East & North Africa (developing only)
XR                              High income: nonOECD
XS                              High income: OECD
XT                              Upper middle income
XU                              North America
XY                              Not classified
Z4                              East Asia & Pacific (all income levels)
Z7                              Europe & Central Asia (all income levels)
ZF                              Sub-Saharan Africa (developing only)
ZG                              Sub-Saharan Africa (all income levels)
ZJ                              Latin America & Caribbean (all income levels)
ZQ                              Middle East & North Africa (all income levels)
```

CHAPTER 4

Climate API

There are two methods to the climate API - `get_modelled`, which returns a `ModelledDataset` instance, and `get_instrumental`, which returns an `InstrumentalDataset` instance. The World Bank API has multiple date pairs associated with each dataset, but a single `wbpy` call will make multiple API calls and return all the dates associated with the requested data type.

For full explanation of the data and associated models, see the Climate API documentation.

Like the Indicators API, locations can be ISO-1366 alpha-2 or alpha-3 country codes. They can also be IDs corresponding to regional river basins. A basin map can be found in the official Climate API documentation. The API includes a KML interface that returns basin definitions, but this is currently not supported by `wbpy`.

4.1 Instrumental data

The available arguments and their definitions are accessible via the `ARG_DEFINITIONS` attribute:

```
c_api = wbpy.ClimateAPI()

c_api.ARG_DEFINITIONS["instrumental_types"]
```

```
{'pr': 'Precipitation (rainfall and assumed water equivalent), in millimeters',
 'tas': 'Temperature, in degrees Celsius'}
```

```
c_api.ARG_DEFINITIONS["instrumental_intervals"]
```

```
['year', 'month', 'decade']
```

```
iso_and_basin_codes = ["AU", 1, 302]

dataset = c_api.get_instrumental(data_type="tas", interval="decade", locations=iso_and_basin_codes)
dataset
```

```
<wbpy.climate.InstrumentalDataset({'tas': 'Temperature, in degrees Celsius'}, 'decade') with id: 200
```

The `InstrumentalDataset` instance stores the API responses, various metadata and methods for accessing the data:

```
pprint(dataset.as_dict())
```

```
{'1': {'1960': 5.975941,
       '1970': 6.1606956,
       '1980': 6.3607564,
       '1990': 6.600332,
       '2000': 7.3054743},
 '302': {'1960': -12.850627,
         '1970': -12.679074,
         '1980': -12.295782,
         '1990': -11.440549,
         '2000': -11.460049},
 u'AU': {'1900': 21.078014,
         '1910': 21.296726,
         '1920': 21.158426,
         '1930': 21.245909,
         '1940': 21.04456,
         '1950': 21.136906,
         '1960': 21.263151,
         '1970': 21.306032,
         '1980': 21.633171,
         '1990': 21.727072,
         '2000': 21.741446}}
```

```
dataset.data_type
```

```
{'tas': 'Temperature, in degrees Celsius'}
```

4.2 Modelled data

`get_modelled` returns data derived from Global Glimate Models. There are various possible data types:

```
c_api.ARG_DEFINITIONS["modelled_types"]
```

```
{'ppt_days': 'Number of days with precipitation > 0.2mm',
 'ppt_days10': 'Number of days with precipitation > 10mm',
 'ppt_days2': 'Number of days with precipitation > 2mm',
 'ppt_days90th': "Number of days with precipitation > the control period's 90th percentile",
 'ppt_dryspell': 'Average number of days between precipitation events',
 'ppt_means': 'Average daily precipitation',
 'pr': 'Precipitation (rainfall and assumed water equivalent), in millimeters',
 'tas': 'Temperature, in degrees Celsius',
 'tmax_days10th': "Number of days with max temperature below the control period's 10th percentile (co
 'tmax_days90th': "Number of days with max temperature above the control period's 90th percentile (ho
 'tmax_means': 'Average daily maximum temperature, Celsius',
 'tmin_days0': 'Number of days with min temperature below 0 degrees Celsius',
 'tmin_days10th': "Number of days with min temperature below the control period's 10th percentile (co
 'tmin_days90th': "Number of days with min temperature above the control period's 90th percentile (wa
 'tmin_means': 'Average daily minimum temperature, Celsius'}
```

```
c_api.ARG_DEFINITIONS["modelled_intervals"]
```

```
{'aanom': 'Average annual change (anomaly).',
 'aavg': 'Annual average',
 'annualanom': 'Average annual change (anomaly).',
 'annualavg': 'Annual average',
 'manom': 'Average monthly change (anomaly).',
 'mavg': 'Monthly average'}
```

```
locations = ["US"]
modelled_dataset = c_api.get_modelled("pr", "aavg", locations)
modelled_dataset
```

```
<wbpy.climate.ModelledDataset({'pr': 'Precipitation (rainfall and assumed water equivalent), in mill
```

The `as_dict()` method for `ModelledDataset` takes a kwarg to specify the SRES used for future values. The API uses the A2 and B1 scenarios:

```
pprint(modelled_dataset.as_dict(sres="a2"))
```

```
{u'bccr_bcm2_0': {u'US': {'1939': 790.6361028238144,
                          '1959': 780.0266445283039,
                          '1979': 782.7526463724754,
                          '1999': 785.2701232986692,
                          '2039': 783.1710625360416,
                          '2059': 804.3092939039038,
                          '2079': 804.6334514665734,
                          '2099': 859.8239942059615}},
 u'cccma_cgcm3_1': {u'US': {'1939': 739.3362184367556,
                            '1959': 746.2975320411192,
                            '1979': 739.4449188917432,
                            '1999': 777.7889471267924,
                            '2039': 808.1474524518724,
                            '2059': 817.1428223416907,
                            '2079': 841.7569757399672,
                            '2099': 871.6962130920673}},
 u'cnrm_cm3': {u'US': {'1939': 939.7243516499025,
                       '1959': 925.6653938577782,
                       '1979': 940.2236730711822,
                       '1999': 947.5967851291585,
                       '2039': 962.6036875622598,
                       '2059': 964.4556538112397,
                       '2079': 970.7166949721155,
                       '2099': 987.7517843651068}},
 u'csiro_mk3_5': {u'US': {'1939': 779.0404023054358,
                          '1959': 799.5361627973773,
                          '1979': 796.607564873811,
                          '1999': 798.381580457504,
                          '2039': 843.0498166357976,
                          '2059': 867.6557574566958,
                          '2079': 884.6635096827529,
                          '2099': 914.4892749739001}},
 'ensemble_10': {u'US': {'1939': 666.6475434339079,
                         '1959': 665.7610790034265,
                         '1979': 667.1738791525539,
                         '1999': 670.415327533486,
                         '2039': 686.4924376146926,
                         '2059': 690.3005736391768,
                         '2079': 693.0003564697117,
                         '2099': 709.0425715268083}},
 'ensemble_50': {u'US': {'1939': 850.8566502216561,
                         '1959': 851.1821259381916,
                         '1979': 852.9435213996902,
                         '1999': 855.0129391106861,
                         '2039': 873.0523341457085,
                         '2059': 880.9922361302446,
                         '2079': 892.9013887250998,
                         '2099': 916.5180306375303}},
 'ensemble_90': {u'US': {'1939': 1020.5076048129349,
                         '1959': 1018.0491512612145,
                         '1979': 1020.2880850240846,
                         '1999': 1029.4064082957505,
                         '2039': 1048.7391596386938,
                         '2059': 1056.5504828474266,
                         '2079': 1067.6845781511777,
                         '2099': 1106.7227445303276}},
 u'gfdl_cm2_0': {u'US': {'1939': 898.1444407247458,
                         '1959': 890.578762482606,
                         '1979': 873.31199204601,
                         '1999': 890.4286021472773,
                         '2039': 884.667792836329,
                         '2059': 891.2301658572712,
                         '2079': 858.2037683045394,
                         '2099': 862.2664763719782}},
 u'gfdl_cm2_1': {u'US': {'1939': 847.0485774775588,
                         '1959': 832.6677468315708,
                         '1979': 840.3616008806812,
                         '1999': 827.3124179982142,
                         '2039': 854.7964182636986,
                         '2059': 870.5118615966802,
                         '2079': 868.5767216101426,
                         '2099': 878.4820392256858}},
 u'ingv_echam4': {u'US': {'1939': 845.4780955327558,
                          '1959': 845.2359494710544,
                          '1979': 852.7707911085288,
                          '1999': 851.9327652092476,
                          '2039': 866.0409073675132,
                          '2059': 872.7481665480419,
                          '2079': 900.9028488881945,
                          '2099': 919.2062848249728}},
 u'inmcm3_0': {u'US': {'1939': 825.6505057699028,
                       '1959': 844.9800055068362,
                       '1979': 860.5045147370352,
                       '1999': 843.0909232427455,
                       '2039': 877.4836079129254,
                       '2059': 885.5902710722888,
                       '2079': 878.6926405756873,
                       '2099': 895.3363280260298}},
 u'ipsl_cm4': {u'US': {'1939': 897.1020362453344,
                       '1959': 881.2890852171191,
                       '1979': 888.57049309408,
                       '1999': 900.6203651333254,
                       '2039': 911.0684866203087,
                       '2059': 908.9880107774133,
                       '2079': 901.9352518210636,
                       '2099': 924.6232749957305}},
 u'miroc3_2_medres': {u'US': {'1939': 815.9899280956733,
                              '1959': 820.924517871823,
                              '1979': 820.561522790526,
                              '1999': 819.1997264378206,
                              '2039': 815.5123964532938,
                              '2059': 812.3150259004544,
                              '2079': 810.515112232343,
                              '2099': 817.447065795786}},
 u'miub_echo_g': {u'US': {'1939': 815.7217424350092,
                          '1959': 819.1216945126766,
                          '1979': 816.4814506968534,
                          '1999': 836.9998036334464,
                          '2039': 841.4617194083404,
                          '2059': 847.7322521257802,
                          '2079': 880.5316551949228,
                          '2099': 920.7048218268357}},
 u'mpi_echam5': {u'US': {'1939': 932.4105818597735,
                         '1959': 930.0013750415483,
                         '1979': 921.4702739003415,
                         '1999': 941.6353488835641,
                         '2039': 969.6867904854836,
                         '2059': 990.3857663124111,
                         '2079': 1000.6110341746332,
                         '2099': 1080.5289311209049}},
 u'mri_cgcm2_3_2a': {u'US': {'1939': 728.5749928767182,
                             '1959': 720.3172590678807,
                             '1979': 732.943309679262,
                             '1999': 727.9981579483319,
                             '2039': 735.1725461582992,
                             '2059': 751.6773914898702,
                             '2079': 776.7754868580876,
                             '2099': 798.3133892715804}},
 u'ukmo_hadcm3': {u'US': {'1939': 839.9996105395489,
                          '1959': 849.9134671410114,
                          '1979': 851.505705112856,
                          '1999': 848.5821514937204,
                          '2039': 874.371671909573,
                          '2059': 877.512058895459,
                          '2079': 881.875457040721,
                          '2099': 927.3730832143624}},
 u'ukmo_hadgem1': {u'US': {'1939': 841.7922922262945,
                           '1959': 845.698748695459,
                           '1979': 834.3090961483945,
                           '1999': 831.8516144217097,
                           '2039': 866.4876927782285,
                           '2059': 864.5861500956854,
                           '2079': 882.1356350906877,
                           '2099': 907.0139017841842}}}
```

Again, various metadata is available, for example:

```
modelled_dataset.gcms
```

```
{u'bccr_bcm2_0': 'BCM 2.0',
 u'cccma_cgcm3_1': 'CGCM 3.1 (T47)',
 u'cnrm_cm3': 'CNRM CM3',
 u'csiro_mk3_5': 'CSIRO Mark 3.5',
 'ensemble_10': '10th percentile values of all models together',
 'ensemble_50': '50th percentile values of all models together',
 'ensemble_90': '90th percentile values of all models together',
 u'gfdl_cm2_0': 'GFDL CM2.0',
 u'gfdl_cm2_1': 'GFDL CM2.1',
 u'ingv_echam4': 'ECHAM 4.6',
 u'inmcm3_0': 'INMCM3.0',
 u'ipsl_cm4': 'IPSL-CM4',
 u'miub_echo_g': 'ECHO-G',
 u'mpi_echam5': 'ECHAM5/MPI-OM',
 u'mri_cgcm2_3_2a': 'MRI-CGCM2.3.2',
 u'ukmo_hadcm3': 'UKMO HadCM3',
 u'ukmo_hadgem1': 'UKMO HadGEM1'}
```

```
modelled_dataset.dates()
```

```
[('1920', '1939'),
 ('1940', '1959'),
 ('1960', '1979'),
 ('1980', '1999'),
 ('2020', '2039'),
 ('2040', '2059'),
 ('2060', '2079'),
 ('2080', '2099')]
```

Cache

The default cache function uses system temporary files. You can specify your own. The function has to take a url, and return the corresponding web page as a string.

```
def func(url):
    # Basic function that doesn't do any caching
    import urllib2
    return urllib2.urlopen(url).read()

# Either pass it in on instantiation...
ind_api = wbpy.IndicatorAPI(fetch=func)

# ...or point api.fetch to it.
climate_api = wbpy.ClimateAPI()
climate_api.fetch = func
```

CHAPTER 6

Further Contents

6.1 Indicator API

class `wbpy.IndicatorDataset` (*json_resp*, *url=None*, *date_of_call=None*)

`dates` (*use_datetime=False*)
Return list of dates used in the dataset.

Parameters **`use_datetime`** – If True, return dates as datetime.date() objects, rather than strings.

`as_dict` (*use_datetime=False*)
Return dictionary of the dataset's data.

Keys are: data[country_code][date]

Parameters **`use_datetime`** – Use datetime.date() object as the date key, rather than string.

class `wbpy.IndicatorAPI` (*fetch=None*)
Request data from the World Bank Indicators API.

You can override the default tempfile cache by passing a function `fetch`, which requests a URL and returns the response as a string.

`get_dataset` (*indicator*, *country_codes=None*, ***kwargs*)
Request a dataset from the API.

Parameters

- **`indicator`** – The API indicator code, eg. SP.POP.TOTL for total population.
- **`country_codes`** – List of ISO 1366 alpha-2 or alpha-3 country codes. If None, returns data for all countries.
- **`kwargs`** – The following map directly to the API query args: `language` `date` `mrv` `gapfill` `frequency`

Returns IndicatorDataset instance containing the dataset and metadata.

`get_indicators` (*indicator_codes=None*, *search=None*, *search_full=False*, *common_only=False*, ***kwargs*)
Request metadata on specific World Bank indicators.

Parameters

- **`indicator_codes`** – A list of codes to get metadata for, eg. [”SP.POP.GROW”]. If None, all indicators are returned (~8000)

- **`common_only`** – Many of the indicators do not have wide data coverage. If True, filter out the ~6500 indicators that do not appear on the main World Bank website (http://data.worldbank.org/indicators/all),
- **`search`** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **`kwargs`** – The following map directly to the API query args: `language` `source` `topic`

Returns Dictionary of indicators and their metadata, with their IDs as keys.

`get_countries`(*country_codes=None*, *search=None*, *search_full=False*, ***kwargs*)

Request country metadata.

eg. ISO code, coordinates, capital, income level, etc.

Parameters

- **`country_codes`** – List of alpha-2 or alpha-3 codes. If None, queries all countries.
- **`search`** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **`kwargs`** – The following map directly to the API query args: `language` `incomeLevel` `lendingType` `region`

Returns Dictionary of metadata with alpha-2 codes as keys.

`get_income_levels`(*income_codes=None*, *search=None*, *search_full=False*, ***kwargs*)

Request income categories.

Parameters

- **`income_codes`** – List of 3-letter ID codes. If None, queries all (~10).
- **`search`** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **`kwargs`** – The following map directly to the API query args: `language`

Returns Dictionary of income levels using ID codes as keys.

`get_lending_types`(*lending_codes=None*, *search=None*, *search_full=False*, ***kwargs*)

Request lending type categories.

Parameters

- **`lending_codes`** – List of lending codes. If None, queries all (4).
- **`search`** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **`kwargs`** – The following map directly to the API query args: `language`

Returns Dictionary of lending types using ID codes as keys.

`get_regions`(*region_codes=None*, *search=None*, *search_full=False*, ***kwargs*)

Request region names and codes.

Parameters

- **region_codes** – List of 3-letter codes. If None, queries all (~26).
- **search** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **kwargs** – The following map directly to the API query args: `language`

Returns Dictionary of regions, using ID codes as keys.

get_topics(*topic_codes=None*, *search=None*, *search_full=False*, ***kwargs*)
Request API topics.

All indicators are mapped to a topic, eg. Health, Private Sector. You can use the topic ID as a kwarg to `get_indicators()`.

Parameters

- **topic_codes** – List of topic IDs. If None, queries all (~20).
- **search** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **kwargs** – The following map directly to the API query args: `language`

Returns Dictionary of topics usings ID numbers as keys.

get_sources(*source_codes=None*, *search=None*, *search_full=False*, ***kwargs*)
Request API source info.

Parameters

- **source_codes** – List of source IDs. If None, queries all (~27).
- **search** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity. If `search_full` is assigned True, it will search all fields for the entity.
- **kwargs** – The following map directly to the API query args: `language`

Returns Dictionary of sources using ID numbers as keys.

print_codes(*results*, *search=None*, *search_key=None*)
Print formatted list of API IDs and their corresponding values.

Parameters

- **search** – Regexp string to filter out non-matching results. By default, this searches the main name of the entity.
- **search_key** – A second-level KEY in your dict, eg. `{foo: {KEY: val}}`. If given, will only search the value corresponding to the key. Only used if `search` is given.
- **results** – A dictionary that was returned by one of the `get` functions.

search_results(*regexp*, *results*, *key=None*)
For a given dict of `get_` results, filter out all keys that do not match the given regexp in either the key or the value. The search is not case sensitive.

Parameters

- **regexp** – The regexp string, passed to `re.search`.
- **results** – A dictionary of `get_foo()` results.

- **key** – A second-level KEY in your dict, eg. `{foo: {KEY: val}}`. If given, will only search the value corresponding to the key.

Returns The input dictionary, with non-matching keys removed.

6.2 Climate API

class `wbpy.`**`InstrumentalDataset`**`(`**args*, ***kwargs*`)`

`as_dict``(`*use_datetime=False*`)`
Return dataset data as dictionary.

Keys are: data[location][date]

Parameters **`use_datetime`** – Use datetime.date() objects for date keys, instead of strings.

class `wbpy.`**`ModelledDataset`**`(`**args*, ***kwargs*`)`

`dates``(`*use_datetime=False*`)`
Return dataset date start/end pairs.

Parameters **`use_datetime`** – If True, return dates as datetime.date() object instead of strings.

`as_dict``(`*sres='a2'*, *use_datetime=False*`)`
Return dataset data as dictionary.

Keys are: data[gcm][location][date]

Parameters

- **`sres`** – Which SRES to use for future values. The API supports A2 and B1, although not all GCMs have data for both.
- **`use_datetime`** – Use datetime.date() objects for date keys, instead of strings.

class `wbpy.`**`ClimateAPI`**`(`*fetch=None*`)`
Request data from the World Bank Climate API.

You can override the default tempfile cache by passing a function `fetch`, which requests a URL and returns the response as a string.

`get_instrumental``(`*data_type*, *interval*, *locations*`)`
Get historical data for temperature or precipitation.

Parameters

- **`data_type`** – Either `pr` for precipitation, or `tas` for temperature.
- **`interval`** – Either `year`, `month` or `decade`.
- **`locations`** – A list of API location codes - either ISO alpha-2 or alpha-3 country codes, or basin ID numbers.

`get_modelled``(`*data_type*, *interval*, *locations*`)`
Get modelled data for precipitation or temperature.

Parameters

- **`data_type`** – The data statistic ID. See `self.ARG_DEFINITIONS["modelled_types"]` for IDs and values.

- **`interval`** – The interval ID. See `self.ARG_DEFINITIONS["modelled_intervals"]` for IDs and values.
- **`locations`** – A list of API location codes - either ISO alpha-2 or alpha-3 country codes, or basin ID numbers.

CHAPTER 7

Indices and tables

- genindex
- search

Index