iris-grib Documentation

Release 0.18.dev0

Met Office

Jun 24, 2022

CONTENTS

The library `iris-grib` provides functionality for converting between weather and climate datasets that are stored as GRIB files and `Iris cubes`. GRIB files can be loaded as Iris cubes using `iris-grib` so that you can use Iris for analysing and visualising the contents of the GRIB files. Iris cubes can be saved to GRIB files using `iris-grib`.

The contents of `iris-grib` represent the former grib loading and saving capabilities of `Iris` itself. These capabilities have been separated into a discrete library so that Iris becomes less monolithic as a library.

CHAPTER

ONE

LOADING

To use `iris-grib` to load existing GRIB files we can make use of the *load_cubes()* function:

```
>>> import os
>>> import iris_sample_data
>>> import iris_grib
>>> cubes = iris_grib.load_cubes(os.path.join(iris_sample_data.path,
                                              'polar_stereo.grib2'))
>>> print cubes
<generator object load_cubes at 0x7f69aba69d70>
```

As we can see, this returns a generator object. The generator object may be iterated over to access all the Iris cubes loaded from the GRIB file, or converted directly to a list:

```
>>> cubes = list(cubes)
>>> print cubes
[<iris 'Cube' of air_temperature / (K) (projection_y_coordinate: 200; projection_x_
↪coordinate: 247)>]
```

Note: There is no functionality in iris-grib that directly replicates `iris.load_cube` (that is, load a single cube directly rather than returning a length-one *CubeList*. Instead you could use the following, assuming that the GRIB file you have loaded contains data that can be loaded to a single cube:

```
>>> cube, = list(cubes)
>>> print cube
air_temperature / (K)               (projection_y_coordinate: 200; projection_x_
↪coordinate: 247)
     Dimension coordinates:
          projection_y_coordinate                           x                      ␣
↪ -
          projection_x_coordinate                           -                      ␣
↪ x
     Scalar coordinates:
          forecast_period: 6 hours
          forecast_reference_time: 2013-05-20 00:00:00
          pressure: 101500.0 Pa
          time: 2013-05-20 06:00:00
```

This makes use of an idiom known as variable unpacking.

CHAPTER

TWO

SAVING

To use `iris-grib` to save Iris cubes to a GRIB file we can make use of the *save_grib2()* function:

```
>>> iris_grib.save_grib2(my_cube, 'my_file.grib2')
```

Note: As the function name suggests, only saving to GRIB2 is supported.

CHAPTER

THREE

INTERCONNECTIVITY WITH IRIS

You can use the functionality provided by `iris-grib` directly within Iris without having to explicitly import `iris-grib`, as long as you have both Iris and `iris-grib` available to your Python interpreter.

For example:

```
>>> import iris
>>> import iris_sample_data
>>> cube = iris.load_cube(iris.sample_data_path('polar_stereo.grib2'))
```

Similarly, you can save your cubes to a GRIB file directly from Iris using `iris-grib`:

```
>>> iris.save(my_cube, 'my_file.grib2')
```

CHAPTER

FOUR

GETTING STARTED

To ensure all `iris-grib` dependencies, it is sufficient to have installed `Iris` itself, and ecCodes .

The simplest way to install is with conda , using the conda-forge channel , with the command

```
$ conda install -c conda-forge iris-grib
```

Development sources are hosted at https://github.com/SciTools/iris-grib .

CHAPTER

FIVE

RELEASES

For recent changes, see Release Notes .

CHAPTER

SIX

INDICES AND TABLES

Contents:

6.1 iris_grib

In this module:

- `load_cubes`
- `save_grib2`
- `load_pairs_from_fields`
- `save_pairs_from_cube`
- `save_messages`

Conversion of cubes to/from GRIB.

See: ECMWF GRIB API.

`iris_grib.`**`load_cubes`**(*filenames*, *callback=None*)

Returns a generator of cubes from the given list of filenames.

Args:

- **filenames:**
  One or more GRIB filenames to load from.

Kwargs:

- **callback:**
  Function which can be passed on to `iris.io.run_callback()`.

Returns:
A generator containing Iris cubes loaded from the GRIB files.

`iris_grib.`**`save_grib2`**(*cube*, *target*, *append=False*)

Save a cube or iterable of cubes to a GRIB2 file.

Args:

- **cube:**
  The `iris.cube.Cube`, `iris.cube.CubeList` or list of cubes to save to a GRIB2 file.
- **target:**
  A filename or open file handle specifying the GRIB2 file to save to.

Kwargs:

- **append:**
  Whether to start a new file afresh or add the cube(s) to the end of the file. Only applicable when target is a filename, not a file handle. Default is False.

iris_grib.**load_pairs_from_fields**(*grib_messages*)

Convert an iterable of GRIB messages into an iterable of (Cube, Grib message) tuples.

This capability can be used to filter out fields before they are passed to the load pipeline, and amend the cubes once they are created, using GRIB metadata conditions. Where the filtering removes a significant number of fields, the speed up to load can be significant:

```
>>> import iris
>>> from iris_grib import load_pairs_from_fields
>>> from iris_grib.message import GribMessage
>>> filename = iris.sample_data_path('polar_stereo.grib2')
>>> filtered_messages = []
>>> for message in GribMessage.messages_from_filename(filename):
...     if message.sections[1]['productionStatusOfProcessedData'] == 0:
...         filtered_messages.append(message)
>>> cubes_messages = load_pairs_from_fields(filtered_messages)
>>> for cube, msg in cubes_messages:
...     prod_stat = msg.sections[1]['productionStatusOfProcessedData']
...     cube.attributes['productionStatusOfProcessedData'] = prod_stat
>>> print(cube.attributes['productionStatusOfProcessedData'])
0
```

This capability can also be used to alter fields before they are passed to the load pipeline. Fields with out of specification header elements can be cleaned up this way and cubes created:

```
>>> from iris_grib import load_pairs_from_fields
>>> cleaned_messages = GribMessage.messages_from_filename(filename)
>>> for message in cleaned_messages:
...     if message.sections[1]['productionStatusOfProcessedData'] == 0:
...         message.sections[1]['productionStatusOfProcessedData'] = 4
>>> cubes = load_pairs_from_fields(cleaned_messages)
```

Args:

- **grib_messages:**
  An iterable of *iris_grib.message.GribMessage*.

Returns:
An iterable of tuples of (`iris.cube.Cube`, *iris_grib.message.GribMessage*).

iris_grib.**save_pairs_from_cube**(*cube*)

Convert one or more cubes to (2D cube, GRIB message) pairs. Returns an iterable of tuples each consisting of one 2D cube and one GRIB message ID, the result of the 2D cube being processed by the GRIB save rules.

Args:

- **cube:**
  A `iris.cube.Cube`, `iris.cube.CubeList` or list of cubes.

iris_grib.**save_messages**(*messages*, *target*, *append=False*)

Save messages to a GRIB2 file. The messages will be released as part of the save.

Args:

- **messages:**
  An iterable of grib_api message IDs.
- **target:**
  A filename or open file handle.

Kwargs:

- **append:**
  Whether to start a new file afresh or add the cube(s) to the end of the file. Only applicable when target is a filename, not a file handle. Default is False.

6.2 iris_grib.message

In this module:

- `GribMessage`
- `Section`

Defines a lightweight wrapper class to wrap a single GRIB message.

class iris_grib.message.**GribMessage**(*raw_message*, *recreate_raw*, *file_ref=None*)

An in-memory representation of a GribMessage, providing access to the *data()* payload and the metadata elements by section via the *sections()* property.

static messages_from_filename(*filename*)

Return a generator of *GribMessage* instances; one for each message in the supplied GRIB file.

Args:

- **filename (string):**
  Name of the file to generate fields from.

property data

The data array from the GRIB message as a dask Array.

The shape of the array will match the logical shape of the message's grid. For example, a simple global grid would be available as a 2-dimensional array with shape (Nj, Ni).

property sections

Return the key-value pairs of the message keys, grouped by containing section.

Sections in a message are indexed by GRIB section-number, and values in a section are indexed by key strings.

class iris_grib.message.**Section**(*message_id*, *number*, *keys*)

A Section of a GRIB message, supporting dictionary like access to attributes using gribapi key strings.

Values for keys may be changed using assignment but this does not write to the file.

get_computed_key(*key*)

Get the computed value associated with the given key in the GRIB message.

Args:

- **key:**
  The GRIB key to retrieve the value of.

Returns the value associated with the requested key in the GRIB message.

keys()

Return coded keys available in this Section.

6.3 iris_grib.grib_phenom_translation

In this module:

- grib1_phenom_to_cf_info
- grib2_phenom_to_cf_info
- cf_phenom_to_grib2_info
- GRIBCode

Provide grib 1 and 2 phenomenon translations to + from CF terms.

This is done by wrapping '_grib_cf_map.py', which is in a format provided by the metadata translation project.

Currently supports only these ones:

- grib1 –> cf
- grib2 –> cf
- cf –> grib2

iris_grib.grib_phenom_translation.**grib1_phenom_to_cf_info**(*table2_version*, *centre_number*, *param_number*)

Lookup grib-1 parameter –> cf_data or None.

Returned cf_data has attributes:

- standard_name
- long_name
- units : a cf_units.Unit
- set_height : a scalar 'height' value , or None

iris_grib.grib_phenom_translation.**grib2_phenom_to_cf_info**(*param_discipline*, *param_category*, *param_number*)

Lookup grib-2 parameter –> cf_data or None.

Returned cf_data has attributes:

- standard_name
- long_name
- units : a cf_units.Unit

iris_grib.grib_phenom_translation.**cf_phenom_to_grib2_info**(*standard_name*, *long_name=None*)

Lookup CF names –> grib2_data or None.

Returned grib2_data has attributes:

- discipline
- category
- number

- **units**
  [a `cf_units.Unit`] The unit represents the defined reference units for the message data.

class iris_grib.grib_phenom_translation.**GRIBCode**(*edition_or_string*, *discipline=None*, *category=None*, *number=None*)

An object representing a specific Grib phenomenon identity.

Basically a namedtuple of (edition, discipline, category, number).

Also provides a string representation, and supports creation from: another similar object; a tuple of numbers; or any string with 4 separate decimal numbers in it.

See also:

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX