

---

# **FV3Config Documentation**

*Release 0.8.0*

**Vulcan Technologies, LLC**

**May 07, 2021**



---

## Contents:

---

<b>1</b>	<b>FV3Config</b>	<b>1</b>
1.1	Basic usage . . . . .	1
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Stable release . . . . .	3
2.2	From sources . . . . .	3
<b>3</b>	<b>Usage</b>	<b>5</b>
3.1	Quickstart . . . . .	5
3.2	Shell Usage . . . . .	5
3.3	Data Caching . . . . .	6
3.4	Configuration . . . . .	6
3.5	Specifying individual files . . . . .	7
3.6	DiagTable configuration . . . . .	7
3.7	Running the model with fv3run . . . . .	9
3.8	Customizing the model execution . . . . .	10
3.9	Submitting a Kubernetes job . . . . .	10
3.10	Restart runs . . . . .	11
3.11	Nudging . . . . .	11
<b>4</b>	<b>API Reference</b>	<b>13</b>
<b>5</b>	<b>History</b>	<b>21</b>
5.1	v0.8.0 (2021-05-07) . . . . .	21
5.2	v0.7.1 (2021-03-18) . . . . .	21
5.3	v0.7.0 (2021-03-17) . . . . .	22
5.4	v0.6.1 (2021-02-23) . . . . .	22
5.5	v0.6.0 (2021-02-22) . . . . .	22
5.6	v0.5.2 (2021-02-04) . . . . .	22
5.7	v0.5.1 . . . . .	22
5.8	v0.5.0 . . . . .	23
5.9	v0.4.0 (2020-07-09) . . . . .	23
5.10	v0.3.2 (2020-04-16) . . . . .	23
5.11	v0.3.1 (2020-04-08) . . . . .	24
5.12	v0.3.0 (2020-04-03) . . . . .	24
5.13	0.2.0 (2020-01-27) . . . . .	25
5.14	0.1.0 (2019-10-11) . . . . .	25

<b>6 Indices and tables</b>	<b>27</b>
<b>Python Module Index</b>	<b>29</b>
<b>Index</b>	<b>31</b>

FV3Config is used to configure and manipulate run directories for FV3GFS.

- Free software: BSD license

## 1.1 Basic usage

```
import fv3config

with open("config.yml", "r") as f:
    config = fv3config.load(f)
fv3config.write_run_directory(config, './rundir')
```

`config` is a configuration dictionary which contains namelists, input data specifications, and other options. It can be edited just like any dictionary.

For more in-depth usage, please refer to the documentation. This can be generated with `make docs`.



### 2.1 Stable release

There is no stable release. This is unsupported, pre-alpha software: use at your own risk!

### 2.2 From sources

The sources for FV3Config can be downloaded from the [Github repo](#).

You can clone the public repository:

```
$ git clone git://github.com/VulcanClimateModeling/fv3config
```

Once you have a copy of the source, you can install it interactively with:

```
$ pip install -e .
```

The `-e` flag will set up the directory so that python uses the local folder including any modifications, instead of copying the sources to an installation directory. This is very useful for development.





## 3.1 Quickstart

The following code would write a run directory based on the contents of a yaml file:

```
import fv3config

with open("config.yml", "r") as f:
    config = fv3config.load(f)
fv3config.write_run_directory(config, './rundir')
```

`config` is a configuration dictionary which contains namelists, input data specifications, and other options, as described further below. It can be edited just like any dictionary. Namelists are specified as sub-dictionaries. An example C12 configuration dictionary is in the `tests` directory of this package.

A run directory based on a configuration can be written using `fv3config.write_run_directory()`.

## 3.2 Shell Usage

This module installs a command line interface `write_run_directory` that can be used to write the run directory from a shell. For example, if the file `config.yaml` contains a yaml-encoded configuration dictionary

```
write_run_directory config.yaml rundir
```

will write an FV3 run directory to the path `rundir`.

Two additional command line interfaces are useful for modifying configuration dictionaries in order to use them for restart runs:

```
enable_restart config.yaml /path/to/initial/conditions
```

and to provision the necessary files required for a nudged run:

```
enable_nudging config.yaml
```

Both of these shell commands will modify the given configuration dictionary in place.

This module also installs a command line interface *fv3run*, which is further detailed below.

### 3.3 Data Caching

*fv3config* can write files from local or remote locations. When remote locations are used, the package first downloads the data to a local cache directory.

If the `FV3CONFIG_CACHE_DIR` environment variable is set, the package will download and store data into `$(FV3CONFIG_CACHE_DIR)/fv3config-cache`. If unset, by default the package will use the “user cache” directory for the user’s operating system.

The download location can be retrieved using *fv3config.get\_cache\_dir()*, and set manually using *fv3config.set\_cache\_dir()*. Note that the “fv3config-cache” subdirectory will be appended to the cache directory you set. If the target is set to a directory that already contains the archive download, it will automatically start using those files. Conversely, if the target is set to an empty directory, it will be necessary to re-download the cache files.

It’s unlikely, but do not set the cache directory to a location that already contains a “fv3config-cache” subdirectory with unrelated files, or the cache files will not download until you call *fv3config.refresh\_downloaded\_data()* (which will delete any files in the subdirectory).

Automatic caching of remote files can be disabled using the *fv3config.do\_remote\_caching()* routine.

### 3.4 Configuration

The `config` dictionary must have at least the following items:

Key	Type	Description
<code>namelist</code>	<code>dict</code>	Model namelist
<code>experiment_name</code>	<code>str</code>	Name of experiment to use in output
<code>diag_table</code>	<code>str</code> or <i>DiagTable</i>	location of <code>diag_table</code> file, or one of (“default”, “grid_spec”, “no_output”), or <i>DiagTable</i> object
<code>data_table</code>	<code>str</code>	location of <code>data_table</code> file, or “default”
<code>initial_conditions</code>	<code>str</code>	location of directory containing initial conditions data
<code>forcing</code>	<code>str</code>	location of directory containing forcing data
<code>orographic_forcing</code>	<code>str</code>	location of directory containing orographic data

Paths to files or directories on the local filesystem must be given as absolute paths. If paths are given that begin with `gs://` then *fv3config* will attempt to download the specified file or files from Google Cloud Storage. For this functionality, *gsfs* must be installed and authorized to download from the specified bucket.

The `namelist` item is special in that it is explicitly stored in the `config` dictionary. For the *fv3gfs* model, individual namelists are specified for various components of the model. As an example, the vertical resolution can be accessed via `config['namelist']['fv_core_nml']['npz']`.

The `diag_table` can be either be a tag or path to a file, or it can explicitly represent the desired output diagnostics with a *DiagTable* object. See a more complete description of this object below.

By default, fv3config attempts to automatically select the `field_table` file to use for the model based on the selected microphysics scheme in the namelist. This supports Zhao-Carr or GFDL microphysics. If the user provides a `field_table` key indicating a filename in the configuration dictionary, that file will be used instead.

---

**Note:** The [Han and Bretherton \(2019\)](#) TKE-EDMF boundary layer scheme requires an additional tracer to be defined in the `field_table` for TKE. This scheme is currently not supported by default in fv3config; however for the time being one can supply a custom `field_table` for this purpose.

---

Some helper functions exist for editing and retrieving information from configuration dictionaries, like `fv3config.get_run_duration()` and `fv3config.set_run_duration()`. See the [API Reference](#) for more details.

## 3.5 Specifying individual files

More fine-grained control of the files that are written to the run-directory is possible using the “asset” representation of run-directory files. An asset is a dictionary that knows about one file’s source location/filename, target filename, target location within the run directory and whether that file is copied or linked. Asset dicts can be generated with the helper function `fv3config.get_asset_dict()`. For example:

```
>>> get_asset_dict('/path/to/filedir/', 'sample_file.nc', target_location='INPUT/')
{'source_location': '/path/to/filedir/',
 'source_name': 'sample_file.nc',
 'target_location': 'INPUT/',
 'target_name': 'sample_file.nc',
 'copy_method': 'copy'}
```

One can also add specify the asset as a python bytes object that will be written to the desired location using `fv3config.get_bytes_asset_dict()`. For example:

```
>>> get_bytes_asset_dict(b"hello_world", "hello.txt", target_location=".")
```

This is useful for storing small files in the configuration dictionary, without needing to deploy them to an external storage system.

One can set `config['initial_conditions']` or `config['forcing']` to a list of assets in order to specify every initial condition or forcing file individually.

One can use a directory to specify the initial conditions or forcing files and replace only a subset of the files within the that directory with the optional `config['patch_files']` item. All assets defined in `config['patch_files']` will overwrite any files specified in the initial conditions or forcing if they have the same target location and name.

## 3.6 DiagTable configuration

The `diag_table` specifies the diagnostics to be output by the Fortran model. See documentation for the string representation of the `diag_table` [here](#). The fv3config package defines a python representation of this object, `DiagTable`, which can be used to explicitly represent the `diag_table` within an fv3config configuration dictionary.

The `DiagTable` object can be initialized from a dict (here serialized as YAML) as follows. Suppose the following is saved within `sample_diag_table.yaml`:

```

name: example_diag_table
base_time: 2000-01-01 00:00:00
file_configs:
- name: physics_diagnostics
  frequency: 1
  frequency_units: hours
  field_configs:
  - field_name: totprcpb_ave
    module_name: gfs_phys
    output_name: surface_precipitation_rate
  - field_name: ULWRFtoa
    module_name: gfs_phys
    output_name: upward_longwave_radiative_flux_at_toa

```

Then a DiagTable object can be initialized as:

```

>>> import fv3config
>>> import yaml
>>> with open("sample_diag_table.yaml") as f:
>>>     diag_table_dict = yaml.safe_load(f)
>>> diag_table = fv3config.DiagTable.from_dict(diag_table_dict)
>>> print(diag_table) # will output diag_table in format expected by Fortran model
example_diag_table
2000 1 1 0 0 0

"physics_diagnostics", 1, "hours", 1, "hours", "time"

"gfs_phys", "totprcpb_ave", "surface_precipitation_rate", "physics_diagnostics", "all
↔", "none", "none", 2
"gfs_phys", "ULWRFtoa", "upward_longwave_radiative_flux_at_toa", "physics_diagnostics
↔", "all", "none", "none", 2

```

The same DiagTable can also be initialized programmatically as follows:

```

>>> import fv3config
>>> import datetime
>>> diag_table = fv3config.DiagTable(
>>>     name="example_diag_table",
>>>     base_time=datetime.datetime(2000, 1, 1),
>>>     file_configs=[
>>>         fv3config.DiagFileConfig(
>>>             name="physics_diagnostics",
>>>             frequency=1,
>>>             frequency_units="hours",
>>>             field_configs=[
>>>                 fv3config.DiagFieldConfig(
>>>                     "gfs_phys",
>>>                     "totprcb_ave",
>>>                     "surface_precipitation_rate"
>>>                 ),
>>>                 fv3config.DiagFieldConfig(
>>>                     "gfs_phys",
>>>                     "ULWRFtoa",
>>>                     "upward_longwave_radiative_flux_at_toa"
>>>                 ),
>>>             ],
>>>         ),
>>>     ]
>>> )

```

(continues on next page)

(continued from previous page)

```
    ]
)
```

String representations of the `diag_table` (i.e. those expected by the Fortran model) can be parsed with the `fv3config.DiagTable.from_str()` method.

If serializing an `fv3config` configuration object to yaml it is recommended to use `fv3config.dump()`. This method will convert any `DiagTable` instances to dicts (using `fv3config.DiagTable.asdict()`), which can be safely serialized.

## 3.7 Running the model with fv3run

`fv3config` provides a tool for running the python-wrapped model called `fv3run`. For example, you can run the default configuration using first:

```
$ docker pull us.gcr.io/vcm-ml/fv3gfs-python
```

to acquire the docker image for the python wrapper, followed by a call to `fv3config.run_docker()`:

```
>>> import fv3config
>>> with open("config.yml", "r") as f:
>>>     config = fv3config.load(f)
>>> fv3config.run_docker(config, 'outdir', docker_image='us.gcr.io/vcm-ml/fv3gfs-
↳python')
```

If the `fv3gfs-python` package is installed natively, the model could be run using `fv3config.run_native()`:

```
>>> fv3config.run_native(config, 'outdir')
```

The python config can be passed as either a configuration dictionary, or the name of a yaml file. There is also a bash interface for running from yaml configuration.

```
$ fv3run --help
usage: fv3run [-h] [--runfile RUNFILE] [--dockerimage DOCKERIMAGE]
             [--keyfile KEYFILE]
             config outdir

Run the FV3GFS model. Will use google cloud storage key at
$GOOGLE_APPLICATION_CREDENTIALS by default.

positional arguments:
  config                location of fv3config yaml file
  outdir               location to copy final run directory, used as run
                      directory if local

optional arguments:
  -h, --help           show this help message and exit
  --runfile RUNFILE   location of python script to execute with mpirun
  --dockerimage DOCKERIMAGE
                      if passed, execute inside a docker image with the
                      given name
  --keyfile KEYFILE   google cloud storage key to use for cloud copy
                      commands
```

(continues on next page)

(continued from previous page)

```
--kubernetes      if given, ignore --keyfile and output a yaml
                   kubernetes config to stdout instead of submitting a
                   run
```

The only required inputs are `config`, which specifies a yaml file containing the `fv3config` run directory configuration, and a final location to copy the run directory. A keyfile can be specified to authenticate Google cloud storage for any data sources located in Google cloud buckets, or the key is taken from an environment variable by default. If `dockerimage` is specified, the command will run inside a Docker container based on the given image name. This assumes the `fv3config` package and `fv3gfs` python wrapper are installed inside the container, along with any dependencies.

The python interface is very similar to the command-line interface, but is split into separate functions based on where the model is being run.

### 3.8 Customizing the model execution

The `runfile` is the python script that will be executed by `mpi`, which typically imports the `fv3gfs` module, and then performs some time stepping. The default behavior is to use a pre-packaged runfile which reproduces the behavior of Fortran model identically. For additional, flexibility a custom runfile can be specified as an argument to all the `run_` functions.

The environmental variable `FV3CONFIG_DEFAULT_RUNFILE` can be used to override the default runfile. If set, this variable should contain the path of the runfile.

---

**Note:** When using `run_docker` or `run_kubernetes`, the value of `FV3CONFIG_DEFAULT_RUNFILE` and the file it points to will be read inside the docker image where execution occurs. It will have no effect if set on the host system outside of the docker image.

---

### 3.9 Submitting a Kubernetes job

A python interface `fv3config.run_kubernetes()` is provided for submitting `fv3run` jobs to Kubernetes. Here's an example for submitting a job based on a config dictionary stored in Google cloud storage:

```
import gcsfs
import fv3config

config_location = 'gs://my_bucket/fv3config.yml'
outdir = 'gs://my_bucket/rundir'
docker_image = 'us.gcr.io/vcm-ml/fv3gfs-python'

fv3config.run_kubernetes(
    config_location,
    outdir,
    docker_image,
    gcp_secret='gcp-key' # replace with your kubernetes secret
                       # containing gcp key in key.json
)
```

The `gcp` key is generally necessary to gain permissions to read and write from google cloud storage buckets. In the unlikely case that you are writing to a public bucket, it can be ommitted.

From the command line, `fv3run` can be used to create a yaml file to submit for a kubernetes job. To create the file, add the `--kubernetes` flag to `fv3run` and pipe the result to a file. For example:

```
$ fv3run gs://bucket/config.yaml gs://bucket/outdir --dockerimage us.gcr.io/vcm-ml/fv3gfs-python:latest
--kubernetes > kubeconfig.yaml
```

The resulting file can be submitted using

```
$ kubectl apply -f kubeconfig.yaml
```

You can also modify the yaml file before submitting the job, for example to request more than one processor or a different amount of memory.

## 3.10 Restart runs

The required namelist settings for a restart run (as opposed to a run initialized from an observational analysis) can be applied to a configuration dictionary as follows:

```
config = enable_restart(config, initial_conditions)
```

## 3.11 Nudging

The `fv3gfs` model contains a module for nudging the state of the atmosphere towards GFS analysis. Two public functions are provided to ease the configuration of nudging runs.

Given the run duration and start date, `fv3config.get_nudging_assets()` returns a list of `fv3config` assets corresponding to the GFS analysis files required. Given an `fv3config` object, `fv3config.enable_nudging()` return a new `config` with the necessary assets and namelist options for a nudging run. This function requires that the `fv3config` object contains a `gfs_analysis_data` entry with corresponding `url` and `filename_pattern` items.





Top-level package for fv3config.

**exception** fv3config.**ConfigError**

Bases: ValueError

**class** fv3config.**DiagFieldConfig**(*module\_name: str, field\_name: str, output\_name: str, time\_sampling: str = 'all', reduction\_method: str = 'none', regional\_section: str = 'none', packing: fv3config.config.diag\_table.Packing = <Packing.SINGLE\_PRECISION: 2>*)

Bases: object

Object representing configuration for a field of a diagnostics file.

#### Parameters

- **module\_name** – Name of Fortran module containing diagnostic.
- **field\_name** – Name of diagnostic within Fortran code.
- **output\_name** – Name of diagnostic to use in output NetCDF.
- **time\_sampling** – Always set to 'all'.
- **reduction\_method** – One of 'none', 'average', 'min', 'max'.
- **regional\_section** – 'none' or region specification.
- **packing** – precision for output data.

```
packing = 2
```

```
reduction_method = 'none'
```

```
regional_section = 'none'
```

```
time_sampling = 'all'
```

```
class fv3config.DiagFileConfig(name: str, frequency: int, frequency_units: str, field_configs:  
                             Sequence[fv3config.config.diag_table.DiagFieldConfig],  
                             file_format: fv3config.config.diag_table.FileFormat =  
                             <FileFormat.NETCDF: 1>, time_axis_units: str = 'hours',  
                             time_axis_name: str = 'time')
```

Bases: object

Object representing a diagnostics file configuration.

#### Parameters

- **name** – Name to use for NetCDF files, not including ‘.tile?.nc’.
- **frequency** – Period between records in file.
- **frequency\_units** – One of ‘years’, ‘months’, ‘days’, ‘hours’, ‘minutes’, ‘seconds’
- **field\_configs** – Sequence of DiagFieldConfigs defining fields to save.
- **file\_format** – Always FileFormat.NETCDF.
- **time\_axis\_units** – Units for time coordinate in output files. One of ‘years’, ‘months’, ‘days’, ‘hours’, ‘minutes’, ‘seconds’.
- **time\_axis\_name** – Name for time coordinate in output files.

```
file_format = 1
```

```
time_axis_name = 'time'
```

```
time_axis_units = 'hours'
```

```
class fv3config.DiagTable(name: str, base_time: datetime.datetime, file_configs: Se-  
                          quence[fv3config.config.diag_table.DiagFileConfig])
```

Bases: object

Representation of diag\_table, which controls Fortran diagnostics manager.

---

**Note:** This implementation is based on the diag\_table specification described in

---

[https://data1.gfdl.noaa.gov/summer-school/Lectures/July16/03\\_Seth1\\_DiagManager.pdf](https://data1.gfdl.noaa.gov/summer-school/Lectures/July16/03_Seth1_DiagManager.pdf) The MOM6 documentation has a useful description as well: <https://mom6.readthedocs.io/en/latest/api/generated/pages/Diagnostics.html>.

#### Parameters

- **name** – label used as attribute in output diagnostic files. Cannot contain spaces.
- **base\_time** – time to be used as reference for time coordinate units.
- **file\_configs** – sequence of DiagFileConfig’s defining the diagnostics to be output.

```
asdict ()
```

```
classmethod from_dict (diag_table: dict)
```

```
classmethod from_str (diag_table: str)
```

Initialize DiagTable class from Fortran string representation.

```
class fv3config.FileFormat
```

Bases: enum.Enum

An enumeration.

**NETCDF = 1**

**exception** `fv3config.InvalidFileError`

Bases: `FileNotFoundError`

Raised when a specified file is invalid, either non-existent or not as expected.

**class** `fv3config.Packing`

Bases: `enum.Enum`

An enumeration.

**DOUBLE\_PRECISION = 1**

**SINGLE\_PRECISION = 2**

`fv3config.asset_list_from_path` (*from\_location*, *target\_location=""*, *copy\_method='copy'*)

Return `asset_list` from all files within a given path.

#### Parameters

- **location** (*str*) – local path or google cloud storage url.
- **target\_location** (*str*, *optional*) – `target_location` used for generated assets. Defaults to "" which is root of run-directory.
- **copy\_method** (*'copy' or 'link'*, *optional*) – whether to copy or link assets, defaults to 'copy'. If location is a google cloud storage url, this option is ignored and files are copied.

**Returns** a list of asset dictionaries

**Return type** list

`fv3config.config_from_namelist` (*namelist\_filename*)

Read a configuration dictionary from a namelist file.

Only reads the namelist configuration.

**Parameters** **namelist\_filename** (*str*) – a namelist filename

**Returns** a configuration dictionary

**Return type** `return_dict` (dict)

**Raises** `InvalidFileError` – if the specified filename does not exist

`fv3config.config_to_namelist` (*config*, *namelist\_filename*)

Write the namelist of a configuration dictionary to a namelist file.

#### Parameters

- **config** (*dict*) – a configuration dictionary
- **namelist\_filename** (*str*) – filename to write, will be overwritten if present

`fv3config.do_remote_caching` (*flag: bool*)

Set whether to cache remote files when accessed. Default is True.

`fv3config.dump` (*config: Mapping[str, Any]*, *f: TextIO*)

Serialize config to a file-like object using yaml encoding

#### Parameters

- **config** – an `fv3config` object
- **f** – the file like object to write to

`fv3config.enable_nudging` (*config: Mapping[KT, VT\_co]*) → Mapping[KT, VT\_co]

Return config object with necessary nudging file assets and associated file\_names namelist entry. Requires 'gfs\_analysis\_data' entry in fv3config object with 'url' and 'filename\_pattern' entries.

**Parameters** `config` – configuration dictionary

**Raises** `ConfigError` – if provided config does not contain “gfs\_analysis\_data” section.

---

**Note:** will delete any existing assets in 'patch\_files' that match the given filename\_pattern before new assets are added.

---

`fv3config.enable_restart` (*config, initial\_conditions*)

Apply namelist settings for initializing from model restart files.

**Parameters**

- **config** (*dict*) – a configuration dictionary
- **initial\_conditions** (*str*) – path to desired new initial conditions.

**Returns** a configuration dictionary

**Return type** dict

`fv3config.ensure_data_is_downloaded` ()

Removed, do not use.

`fv3config.get_asset_dict` (*source\_location, source\_name, target\_location=""*, *target\_name=None*, *copy\_method='copy'*)

Helper function to generate asset for a particular file

**Parameters**

- **source\_location** (*str*) – path to directory containing source file
- **source\_name** (*str*) – filename of source file
- **target\_location** (*str, optional*) – sub-directory to which file will be written, relative to run directory root. Defaults to empty string (i.e. root of run directory).
- **target\_name** (*str, optional*) – filename to which file will be written. Defaults to None, in which case source\_name is used.
- **copy\_method** (*str, optional*) – flag to determine whether file is copied ('copy') or hard-linked ('link'). Defaults to 'copy'.

**Returns** an asset dictionary

**Return type** dict

`fv3config.get_bytes_asset_dict` (*data: bytes, target\_location: str, target\_name: str*)

Helper function to define the necessary fields for a binary asset to be saved at a given location.

**Parameters**

- **data** – the bytes to save
- **target\_location** – sub-directory to which file will be written, relative to run directory root. Defaults to empty string (i.e. root of run directory).
- **target\_name** – filename to which file will be written. Defaults to None, in which case source\_name is used.

**Returns** an asset dictionary

**Return type** dict

`fv3config.get_cache_dir()`

`fv3config.get_default_config()`

Removed, do not use.

`fv3config.get_nudging_assets` (*run\_duration*: *datetime.timedelta*, *current\_date*: *Sequence[int]*, *nudge\_path*: *str*, *nudge\_filename\_pattern*: *str* = `'%Y%m%d_%HZ_T85LR.nc'`, *copy\_method*: *str* = `'copy'`, *nudge\_interval*: *datetime.timedelta* = *datetime.timedelta(seconds=21600)*) → `List[Mapping[KT, VT_co]]`

Return list of assets of nudging files required for given run duration and start time.

This method defines file paths directly from its arguments, without determining whether the files themselves are present.

**Parameters**

- **run\_duration** – length of fv3gfs run
- **current\_date** – start time of fv3gfs run as a sequence of 6 integers
- **nudge\_path** – local or remote path to nudging files
- **nudge\_filename\_pattern** – template for nudging filenames. Pattern should follow style of `datetime.strptime` and `strptime` ‘format’ argument. Defaults to `'%Y%m%d_%HZ_T85LR.nc'`.
- **copy\_method** – copy\_method for nudging file assets. Defaults to ‘copy’.
- **nudge\_interval** – time between nudging files. Must be multiple of 1 hour. Defaults to 6 hours.

**Returns** list of all assets required for nudging run

**Raises** `ConfigError` – if copy\_method is “link” and a remote path is given for nudge\_path

`fv3config.get_run_duration` (*config*)

Return a timedelta indicating the duration of the run.

**Parameters** *config* (*dict*) – a configuration dictionary

**Returns** the duration of the run

**Return type** duration (timedelta)

**Raises** `ValueError` – if the namelist contains a non-zero value for “months”

`fv3config.get_timestep` (*config*)

Get the model timestep from a configuration dictionary.

**Parameters** *config* (*dict*) – a configuration dictionary

**Returns** the model timestep

**Return type** `datetime.timedelta`

`fv3config.load` (*f*: *TextIO*) → `Mapping[str, Any]`

Load a configuration from a file-like object *f*

`fv3config.run_docker` (*config\_dict\_or\_location*, *outdir*, *docker\_image*, *runfile=None*, *keyfile=None*, *capture\_output=True*)

Run the FV3GFS model in a docker container with the given configuration.

Copies the resulting directory to a target location. Will use the Google cloud storage key at `$GOOGLE_APPLICATION_CREDENTIALS` by default. Requires the `fv3gfs-python` package and `fv3config` to be installed in the docker image.

#### Parameters

- **config\_dict\_or\_location** (*dict or str*) – a configuration dictionary, or a location (local or on Google cloud storage) of a yaml file containing a configuration dictionary
- **outdir** (*str*) – location to copy the resulting run directory
- **runfile** (*str, optional*) – Python model script to use in place of the default.
- **docker\_image** (*str, optional*) – If given, run this command inside a container using this docker image. Image must have this package and `fv3gfs-python` installed.
- **keyfile** (*str, optional*) – location of a Google cloud storage key to use inside the docker container
- **capture\_output** (*bool, optional*) – If true, then the `stderr` and `stdout` streams will be redirected to the files `outdir/stderr.log` and `outdir/stdout.log` respectively.

```
fv3config.run_kubernetes(config_location, outdir, docker_image, runfile=None, jobname=None, namespace='default', memory_gb=3.6, memory_gb_limit=None, cpu_count=1, gcp_secret=None, image_pull_policy='IfNotPresent', job_labels=None, submit=True, capture_output=True)
```

Submit a kubernetes job to perform a `fv3run` operation.

Much of the configuration must be first saved to google cloud storage, and then supplied via paths to that configuration. The resulting run directory is copied out to a google cloud storage path. This call is non-blocking, and only submits a job.

#### Parameters

- **config\_location** (*str*) – google cloud storage location of a yaml file containing a configuration dictionary
- **outdir** (*str*) – google cloud storage location to upload the resulting run directory
- **docker\_image** (*str*) – docker image name to use for execution, which has `fv3config` installed with `fv3run`
- **runfile** (*str, optional*) – location of a python file to execute as the model executable, either on google cloud storage or within the specified docker image
- **jobname** (*str, optional*) – name to use for the kubernetes job, defaults to a random `uuid.uuid4().hex`
- **namespace** (*str, optional*) – kubernetes namespace for the job, defaults to “default”
- **memory\_gb** (*float, optional*) – gigabytes of memory required for the kubernetes worker, defaults to 3.6GB
- **memory\_gb\_limit** (*float, optional*) – maximum memory allowed for the kubernetes worker, defaults to the value set by `memory_gb`
- **cpu\_count** (*int, optional*) – number of CPUs to use on the kubernetes worker
- **gcp\_secret** (*str, optional*) – name of kubernetes secret to mount containing a file `key.json` to use as the google cloud storage key.
- **image\_pull\_policy** (*str, optional*) – pull policy passed on to the kubernetes job. if set to “Always”, will always pull the latest image. When “IfNotPresent”, will only pull if no image has already been pulled. Defaults to “IfNotPresent”.

- **job\_labels** (*Mapping[str, str], optional*) – labels provided as key-value pairs to apply to job pod. Useful for grouping jobs together in status checks.
- **capture\_output** (*bool, optional*) – If True, then the stderr and stdout streams will be redirected to the files *outdir/stderr.log* and *outdir/stdout.log* respectively.

`fv3config.run_native` (*config\_dict\_or\_location, outdir, runfile=None, capture\_output: bool = True*)

Run the FV3GFS model with the given configuration.

Copies the resulting directory to a target location. Will use the Google cloud storage key at `$GOOGLE_APPLICATION_CREDENTIALS` by default. Requires the `fv3gfs-python` package.

#### Parameters

- **config\_dict\_or\_location** (*dict or str*) – a configuration dictionary, or a location (local or on Google cloud storage) of a yaml file containing a configuration dictionary
- **outdir** (*str*) – location to copy the resulting run directory
- **runfile** (*str, optional*) – Python model script to use in place of the default.
- **capture\_output** (*bool, optional*) – If true, then the stderr and stdout streams will be redirected to the files *outdir/stderr.log* and *outdir/stdout.log* respectively.

`fv3config.set_cache_dir` (*parent\_dirname*)

`fv3config.set_run_duration` (*config: dict, duration: datetime.timedelta*) → dict

Set the run duration in the configuration dictionary.

Returns a new configuration dictionary.

#### Parameters

- **config** (*dict*) – a configuration dictionary
- **duration** (*timedelta*) – the new run duration

**Returns** configuration dictionary with the new run duration

**Return type** `new_config` (dict)

**Raises** `ConfigError` – if the config dictionary is invalid

`fv3config.write_run_directory` (*config, target\_directory*)

Write a run directory based on a configuration dictionary.

#### Parameters

- **config** (*dict*) – a configuration dictionary
- **target\_directory** (*str*) – target directory, will be created if it does not exist





### 5.1 v0.8.0 (2021-05-07)

#### 5.1.1 Breaking changes:

- rename `fv3config.update_config_for_nudging` to `fv3config.enable_nudging` and make this function return a new config instead of mutating the input
- rename `update_config_for_nudging` command line entrypoint to `enable_nudging`

#### 5.1.2 Bug fixes:

- ensure `fv3config.write_run_directory` does not mutate input config
- fix bug in which `DiagTable.from_str` failed on lines that contain only whitespace

#### 5.1.3 Minor changes

- The responsibility of retrying gcsfs operations is now delegated to gcsfs.

### 5.2 v0.7.1 (2021-03-18)

#### 5.2.1 Bug Fixes:

- replace a couple instances of `yaml.dump/load` with `fv3config.dump/load`

## 5.3 v0.7.0 (2021-03-17)

### 5.3.1 Breaking changes:

- Modify the serialization APIs
- add `fv3config.load/dump`
- remove `fv3config.config_to_yaml` and `fv3config.config_from_yaml`

### 5.3.2 Bug Fixes:

- use `DiagTable` aware serialization routines inside the CLIs

## 5.4 v0.6.1 (2021-02-23)

### 5.4.1 Minor changes:

- don't specify a consistency in the `fspec.filesystem` instantiation

## 5.5 v0.6.0 (2021-02-22)

### 5.5.1 Major changes:

- add `DiagTable` class with associated `DiagFileConfig` and `DiagFieldConfig` dataclasses.
- make `fv3config.config_to_yaml` a public function.
- update `fv3config.config_to_yaml` and `fv3config.config_from_yaml` to go between `fv3config.DiagTable` and `dict` types as necessary when serializing/deserializing
- `write_run_directory` provisions necessary `patch_files` for config if the `fv_core_nml.nudge` option is set to `True`.

## 5.6 v0.5.2 (2021-02-04)

- Add logging to `write_run_directory` command

## 5.7 v0.5.1

- Fix formatting of this changelog for PyPI

## 5.8 v0.5.0

### 5.8.1 Breaking changes:

- `enable_restart` function now requires an `initial_conditions` argument. It also sets `force_date_from_namelist` to `False`.

### 5.8.2 Major changes:

- a new public function `fv3config.get_bytes_asset_dict`
- a new command line interface `write_run_directory`
- removed integration tests for `run_docker` and `run_native` which actually executed the model
- all tests are now offline, using a mocked in-memory gcsfs to represent remote communication.
- add a Dockerfile to produce a lightweight image with `fv3config` installed
- Add new public functions `fv3config.get_nudging_assets` and `fv3config.update_config_for_nudging`.
- Add CLI entry points for `enable_restart` and `update_config_for_nudging`.

### 5.8.3 Minor changes:

- updated `create_rundir` example to accept external arguments
- refactor `get_current_date` function to not require the path to the `INPUT` directory.

## 5.9 v0.4.0 (2020-07-09)

### 5.9.1 Major changes:

- the old “default” data options are removed
- `orographic_forcing` is now a required configuration key
- `get_default_config()` is removed, with a placeholder which says it was removed
- `ensure_data_is_downloaded` is removed, with a placeholder which says it was removed

## 5.10 v0.3.2 (2020-04-16)

### 5.10.1 Major changes:

- filesystem operations now manually backoff with a 1-minute max time on `RuntimeError` (which gcsfs often raises when it shouldn't) and `gcsfs.utils.HttpError`
- `put_directory` now makes use of a thread pool to copy items in parallel.

### 5.10.2 Minor changes:

- *run\_docker* now works when supplying an outdir on google cloud storage
- *put\_directory* is now marked as package-private instead of module-private

## 5.11 v0.3.1 (2020-04-08)

### 5.11.1 Major changes:

- Add *get\_timestep* and *config\_from\_yaml* functions

### 5.11.2 Minor changes:

- Allow *config\_to\_yaml* to write to remote locations
- Control whether outputs are logged to console or not in *run\_kubernetes*, *run\_native*, and *run\_docker*.

### 5.11.3 Breaking changes

- Print stderr and stdout to the console by default when using *fv3run*. Use the *-capture-output* command-line flag to enable the previous behavior.

## 5.12 v0.3.0 (2020-04-03)

### 5.12.1 Major changes:

- Added *-kubernetes* command-line flag to output a kubernetes config yaml to stdout

### 5.12.2 Minor changes:

- Added the flag `--mca btl_vader_single_copy_mechanism none` to *mpirun* in *fv3run* to *mpirun* in *fv3run*
- Add ReadTheDocs configuration file
- Do not require output dir and *fv3config* to be remote in *run\_kubernetes*
- Fix bug when submitting k8s jobs with images that have an “\_” in them

### 5.12.3 Breaking changes

- Refactored *run\_kubernetes* and *run\_docker* to call *run\_native* via a new API serializing their args/kwargs as json strings. The *fv3config* version in a docker image must be greater than or equal inside a container to outside, or a silent error will occur.
- When output location is set to a local path, the job now runs in that output location instead of in a temporary directory which then gets copied. This is done both to reduce copying time for large jobs, and to improve visibility of model behavior while running.

## 5.13 0.2.0 (2020-01-27)

- Began tagging version commits

## 5.14 0.1.0 (2019-10-11)

- Initial pre-alpha release



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





**f**

fv3config, 13



**A**

asdict() (*fv3config.DiagTable* method), 14  
 asset\_list\_from\_path() (*in module fv3config*),  
 15

**C**

config\_from\_namelist() (*in module fv3config*),  
 15  
 config\_to\_namelist() (*in module fv3config*), 15  
 ConfigError, 13

**D**

DiagFieldConfig (*class in fv3config*), 13  
 DiagFileConfig (*class in fv3config*), 13  
 DiagTable (*class in fv3config*), 14  
 do\_remote\_caching() (*in module fv3config*), 15  
 DOUBLE\_PRECISION (*fv3config.Packing* attribute), 15  
 dump() (*in module fv3config*), 15

**E**

enable\_nudging() (*in module fv3config*), 15  
 enable\_restart() (*in module fv3config*), 16  
 ensure\_data\_is\_downloaded() (*in module fv3config*), 16

**F**

file\_format (*fv3config.DiagFileConfig* attribute), 14  
 FileFormat (*class in fv3config*), 14  
 from\_dict() (*fv3config.DiagTable* class method), 14  
 from\_str() (*fv3config.DiagTable* class method), 14  
 fv3config (*module*), 13

**G**

get\_asset\_dict() (*in module fv3config*), 16  
 get\_bytes\_asset\_dict() (*in module fv3config*),  
 16  
 get\_cache\_dir() (*in module fv3config*), 17  
 get\_default\_config() (*in module fv3config*), 17  
 get\_nudging\_assets() (*in module fv3config*), 17

get\_run\_duration() (*in module fv3config*), 17  
 get\_timestep() (*in module fv3config*), 17

**I**

InvalidFileError, 15

**L**

load() (*in module fv3config*), 17

**N**

NETCDF (*fv3config.FileFormat* attribute), 14

**P**

Packing (*class in fv3config*), 15  
 packing (*fv3config.DiagFieldConfig* attribute), 13

**R**

reduction\_method (*fv3config.DiagFieldConfig* at-  
 tribute), 13  
 regional\_section (*fv3config.DiagFieldConfig* at-  
 tribute), 13  
 run\_docker() (*in module fv3config*), 17  
 run\_kubernetes() (*in module fv3config*), 18  
 run\_native() (*in module fv3config*), 19

**S**

set\_cache\_dir() (*in module fv3config*), 19  
 set\_run\_duration() (*in module fv3config*), 19  
 SINGLE\_PRECISION (*fv3config.Packing* attribute), 15

**T**

time\_axis\_name (*fv3config.DiagFileConfig* at-  
 tribute), 14  
 time\_axis\_units (*fv3config.DiagFileConfig* at-  
 tribute), 14  
 time\_sampling (*fv3config.DiagFieldConfig* at-  
 tribute), 13

**W**

write\_run\_directory() (*in module fv3config*), 19