CAMPS Documentation

version 1.2.0

CAMPS Team

April 23, 2021

Contents

An Introduction to the Community Atmospheric Model Post-processing System (CAMPS)

The Community Atmospheric Model Post-processing System (CAMPS) is a Python-based software package that supports Statistical Post-Processing (StatPP) of atmospheric data and is maintained as community code. This package includes frameworks for metadata standards, and tools for data representation.

The original vision for CAMPS put priority on achieving the functionality to reproduce a station-based Model Output Statistics (MOS) forecast development. In its current form, CAMPS has the capability to produce a limited MOS forecast for the variables: Temperature, Dewpoint, Daytime Maximum Temperature, and Nighttime Minimum Temperature.

While replicating a *complete* "MOS forecast development" is still a long term goal, the main focus of CAMPS going forward has shifted towards supporting the National Blend of Models (NBM).

As CAMPS matures and makes progress towards fully supporting the NBM, updates will be added to these documentation pages accordingly.

Current Scope of CAMPS:

- Provide structured metadata for encoding Statistical Post Processing (StatPP) data.
- Provide capability to convert observations and model output to netCDF.
- Provide capability to apply select procedures to data (ex: smoothing, interpolations, etc…).
- Perform a multiple linear regression (MLR), based on the legacy MOS-2000 software.
- Generate forecasts based on MLR output.
- Foster community involvement.
- Provide accessible, documented, robust, and portable code.
- Allow for user created utilities to be easily incorporated into the software (Ex: statistical post-processing techniques, interpolations, atmospheric variable calculations, etc).
- Utilize NetCDF-Linked Data to expand on metadata within CAMPS output files by pointing directly to metadata ontology documentation, as well as codes registries.

User's Guide

It is important to note that CAMPS is actively under development and new features are added frequently. The user is advised to check the GitHub repository and these documentation pages frequently for software and documentation updates.

CAMPS can be utilized in one of two ways; through an interactive python environment, or by running each driver script in succession. The latter is referred to as a "CAMPS forecast development" (for those familiar with MOS, this is similar in nature to a MOS "development").

First, regardless of your use case, ensure CAMPS and the necessary dependencies are properly installed on your machine.

Installation

Python 3.6 or newer is required to run CAMPS. Anaconda3-5.3.1 (or newer) is highly recommended for running CAMPS, as many of the additional required packages are included within an Anaconda distribution.

Required Packages (minimum versions):

Language	Version
Python	3.6

Packages	Version

netCDF4	1.7.3
pyproj	1.9.6
pygrib	2.0.4
metpy	0.12.0
scipy	1.3.1
pandas	0.23.4
seaborn	0.9.0
PyYaml	3.13
numpy	1.17.3
matplotlib	3.1.1

Standard Installation

If you have the correct permissions, you may install CAMPS directly into your system's main Python distribution (either Anaconda or a custom library with all necessary dependencies met for CAMPS). This should make the CAMPS package available for anyone using that Python distribution, just like any other Python package.

To install CAMPS into an existing Python distribution, perform the following:

```
$ cd /path/to/CAMPS/repository_clone
$ python3 setup.py install
```

When installing CAMPS into an existing Python installation, console scripts will be installed to the same bin/ directory as the python executable you are invoking.

Custom Installation

CAMPS can be installed into a separate directory (outside of an existing Python installation). A situation where this might occur is if you are working on a shared computing system. Given an installation path prefix (<install-path>), Python packages will install packages using the following directory structure:

```
$ mkdir -p <install-path>/lib/pythonX.Y/site-packages
```

Note: <install-path> should be the absolute path.

Issue the following commands to set the necessary environment variables and install CAMPS:

```
$ cd /path/to/CAMPS/repository_clone
$ export PYTHONPATH=<install-path>/lib/pythonX.Y/site-packages
$ python3 setup.py install --prefix=<install-path>
$ export PATH=$PATH:<install-path>/bin
```

When installing CAMPS outside of a Python distribution, if the installation path does not exist, and/or the installation directory is not in your PYTHONPATH environment variable, then setup.py will fail. Append the CAMPS installation bin/ directory to PATH. This allows CAMPS console scripts to be available in your shell environment.

Alternatively, you can install CAMPS into your own user space via the following command:

```
$ python3 setup.py install --user
```

which will install into *$HOME/.local* on most modern Linux/UNIX systems.

Initializing CAMPS

After successfully installing CAMPS, it is necessary to open a python session, and import the software at least once. This first import of the CAMPS module will create a hidden directory, `$HOME/.camps/control`, which will contain all the control and configuration file templates.

```
$  python
$  import camps
$  quit()
```

CAMPS Package Structure

camps/camps/core

Contains code that defines a Camps_data object. Also stored here are composite classes such as Time and Location, along with I/O modules. Additionally, the directory with data conversion modules resides within core.

camps/camps/core/data_conversion/grib2_to_nc

Grib2 to netCDF conversion code.

camps/camps/core/data_conversion/marine_to_nc

Marine buoy observations to netCDF code.

camps/camps/core/data_conversion/metar_to_nc

Metar observations to netCDF code. Also includes QC.

camps/camps/scripts

This contains all of the driver scripts. Running all driver scripts constitutes a “CAMPS development”. All scripts require a control file to be used as an argument when running the script.

There are six driver scripts in a CAMPS development:

1. grib2_to_nc_driver.py
2. Metar_driver.py
3. Marine_driver.py
4. Mospred_driver.py
5. equations_driver.py
6. forecast_driver.py

camps/camps/registry

Contains all the example control files for the various drivers. Also, contains several configuration files.

The registry directory in CAMPS can be broken down into 4 categories of files:

1. **Control files for each driver script**
    a. metar_control.yaml
    b. marine_control.yaml
    c. grib2_to_nc_control.yaml
    d. mospred_control.yaml
    e. equations_control.yaml
    f. forecast_control.yaml
    g. graphs.yaml - this controls the graphing functions held within the GUI directory. This feature is currently being updated.
2. **Configuration files, aiding in various aspects of the CAMPS development process**
    a. netcdf.yaml - Serves a comprehensive list of accepted variables within CAMPS. A user must modify to this configuration file using the correct formatting, to add a new supported variable to CAMPS.
    b. constants.py - A collection of constants and unit information used within CAMPS.
    c. marine_to_nc.yaml - key/value pairs that decode marine txt files for the desired variables.

d. nc_to_metar.yaml - key/value pairs that decode metar txt files for the desired variables.

e. pred.yaml - Configuration file that informs CAMPS which predictors, predictands and leadtimes to process, and what procedures to apply.

f. predictands_graphs.yaml - This configures graph features for predictands, this is part of the graphing feature that is currently being updated.

g. predictors_graphs.yaml - This configures graph features for predictors, this is part of the graphing feature that is currently being updated.

h. procedures.yaml - A comprehensive list of accepted procedures within CAMPS. A user must modify this configuration file, using the correct formatting, to add a new supported procedure to CAMPS.

3. List and table files, with station information, which may or may not be of use to the user. Each user may decide which station information they wish to use. For example, the file “short.lst” is a small subset of stations often used by CAMPS developers for testing. Alldevsites.lst is a grouping of all stations from alldevsites.tbl. A user could develop their own desired station list, as was done with short.lst. As long as the station information exists in the .tbl file specified in the driver script control file, then it should work.

4. Util.py is a utility module that aids the CAMPS software in interfacing with yaml files.

camps/camps/StatPP/regression

Contains the modules used for regressions. Currently there is only one multiple linear regression module.

camps/camps/gui

Various modules used to display data are here. The modules in this directory are currently broken and will be fixed in a subsequent release of the software.

camps/camps/mospred

Contains modules that aid in the processing, and calculation, of predictors and predictands (ie. mospred_driver.py).

Examples of mospred support modules would be smooth.py and interp.py. These modules contain procedures for smoothing and interpolating gridded model data onto stations.

> ***Note***
>
> For those familiar with MOS-2000, the functionality of this driver script is essentially equivalent to u201.

camps/camps/libraries/mathlib

Contains modules used when calculating new predictors/predictands. Modules contained here are largely used during the mospred (u201 equivalent) step in a CAMPS development.

CAMPS Forecast Development - start to finish

To run a CAMPS development, aside from the initial import step, there is no need to interact directly with Python. CAMPS makes use of a feature called “console scripts”, which allows a user to run the driver scripts directly from the command line.

Each CAMPS driver script has a subsequent console script:

- metar_driver.py –> *camps_metar_to_nc*
- marine_driver.py –> *camps_marine_to_nc*
- grib2_to_nc_driver.py –> *camps_grib2_to_nc*
- mospred_driver.py –> *camps_mospred*
- equations_driver.py –> *camps_equations*
- forecast_driver.py –> *camps_forecast*

Input files for driver scripts have metadata and format requirements. When using CAMPS interactively, the same metadata requirements still exist, but input file requirements could be circumvented. If a file is not read using the

internal CAMPS reader module, the user could still instantiate a Camps_data object, populate it with data, and add the required metadata.

CAMPS driver scripts require formatted input files. While the sources of input data are currently somewhat limited, as the software matures, additional sources and flexibility of input data will be added. In the spirit of being a community, open source, software package, CAMPS is being designed so that users can easily add their own features if desired. This could include additional data conversion methods.

Every driver script in CAMPS will have an accompanying control file(s) that configures certain aspects of the associated driver script. Without the appropriate control file(s) passed as an argument, CAMPS software will error out.

After installation, template control files can be found in the automatically generated directory: `$HOME/.camps/control`

They can also be viewed in the CAMPS Github Repository in the registry directory.

A CAMPS forecast development can be broken into 4 steps:

Step 1: Data conversion

To start a CAMPS development, you'll need to convert observation and model data into CAMPS compliant NetCDF files, and apply quality control where necessary.

> ***Note***
>
> There is no order required for converting data.

METAR observations:

MDL Hourly Table Text Files (METAR) are ASCII text, fixed-width, and colon-delimited. The data are decoded from METAR reports and a first-pass quality control is performed.

Go to `$HOME/.camps/control/metar_conrol.yaml` to view an example control file.

Then execute the console script, providing the control file as standard input.

```
$ camps_metar_to_nc /path/to/controlfile/metar_control.yaml
```

> ***Note***
>
> For more information on METAR reports, see sections A-D (pages 13.1 to 13.13) of chapter 13 in the following office note document: https://www.weather.gov/media/mdl/TDL_OfficeNote00-1.pdf

Marine observations:

NBDC QC Marine Observations Text Files are ASCII text , fixed-width, and colon-delimited. These files are from the National Buoy Data Center (NBDC) and are quality controlled by NBDC.

Go to `$HOME/.camps/control/marine_conrol.yaml` to view an example control file.

Then execute the driver script:

```
$ camps_marine_to_nc /path/to/controlfile/marine_control.yaml
```

Model data:

Model data will also need to be processed and converted to NetCDF. The input file must only contain GRIB2 Messages that the user wishes to convert to NetCDF. For now, grib2_to_nc_driver only accepts GRIB2 data that has already been filtered by level, and the GRIB2 Messages Grid Definition Template must be one of the acceptable coordinate systems.

Currently accepted projections/grids:

a. Lambert Conformal

b. Polar Stereographic

c. Mercator

d. Regular Latitude/Longitude

We recommend wgrib2 (https://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/) to filter and interpolate GRIB2 Messages outside of CAMPS.

The ability to do filtering and grid-to-grid interpolation internally, within CAMPS, is currently under development.

Once again, create your control file based on the provided template. `$HOME/.camps/control/grib2_to_nc_control.yaml`

Run the console script:

```
$ camps_grib2_to_nc /path/to/controlfile/grib2_to_nc_control.yaml
```

Step 2: Data Processing

During this step, the user will choose what predictors and predictands to process and/or derive, and what procedures to apply. Input files must be in NetCDF-CAMPS format, and have the necessary metadata. The source of this input data will more than likely be the output files produced by grib2_to_nc_driver, Metar_driver, and Marine_driver.

Currently CAMPS only interpolates from select map projections, to stations. The ability to process gridded predictors is being developed for a future release. Mospred_driver can be run for processing predictors and/or predictands. Be sure to specify which (or both) you want to run for in the control file.

Note

Reminder that for a linear regression, predictand is the dependent variable, and the predictor is the independent variable.

Inside the control file for this driver script, paths to other input and control files should be provided. The template control file itself `$HOME/.camps/control/mospred_conrol.yaml` should provide adequate instruction for proper configuration.

Note

In mospred_control.yaml, of unique importance is the “pred_file” declaration. The pred_file provided should contain information about the predictors and predictands for the current development. See the template control file “pred.yaml” for more information on configuration.

```
$ camps_mospred /path/to/controlfile/mospred_control.yaml
```

Step 3: Generate equations

Now we’re ready for the regression. The control file $HOME/.camps/control/equation_conrol.yaml will give an example of how to tune regression parameters. You will again specify the predictors and predictands you want to generate equations for by specifying the path to your “pred_file”.

Note

Required input files are the output files from Mospred_driver.

Execute the driver script:

```
$ camps_equations path/to/controlfile/equations_control.yaml
```

Step 4: Generate Forecast

Finally, we will want to generate some forecast output and apply basic consistency checks.

> ***Note***
>
> Required input files are the output files from Mospred_driver and Equations_driver.

The template control file to follow:

```
$HOME/.camps/control/forecast_control.yaml
```

And the driver,

```
$ camps_forecast /path/to/controlfile/forecast_control.yaml
```

Finished!

That's it! That is all you need for a CAMPS forecast development! All output files are saved in NetCDF format and can be found in the output paths specified inside each driver script control file.

Interactive CAMPS

CAMPS can also be used via an interactive python session. The user can import CAMPS, instantiate a Camps_data object, apply procedures, and read/write data.

When instantiating a Camps_data object, two arguments are accepted: **name** and **autofill**. The default for autofill is set to True. When autofill is set to True, name must match a variable defined within the netcdf.yaml configuration file. This can be through a direct match, or via a predefined alias. Once identified, the associated metadata will be automatically added to the object. If the name doesn't match any variable within netcdf.yaml, an error will be raised.

For example, In netcdf.yaml the following entry exists for wind speed:

```
wind_speed_instant :  &WindSpd
  data_type : float32
  attribute :
    SOSA__observedProperty : "StatPP__Data/Met/Moment/WindSpd"
    long_name : "horizontal wind speed"
    standard_name : "wind_speed"
    comment : "Wind speed is set to -9 if winds are variable."
```

A Camps_data object can then be instantiated as follows:

```
>>> from camps.core.Camps_data import Camps_data
>>> import numpy
>>> wspd = Camps_data('wind_speed_instant') # Initializes object
```

Notice that next to the variable name there is a pre-defined alias "WindSpd". At the bottom of netcdf.yaml there is a list of alternative variable names for some variables, which are linked to the desired alias. These aliases act as "pointers" to allow for CAMPS to recognize alternative variable names, and use the metadata pre-defined in netcdf.yaml.

For example, perhaps your wind speed variable is called "wind_speed_value". At the bottom of the netcdf.yaml file one simply would need to add the following: `"wind_speed_value" : *WindSpd` to the list of pointers. This would link the variable name "wind_speed_value" to the assigned metadata for "wind_speed_instant".

If autofill is instead set to False, name can be whatever string a user decides. However, all metadata will need to be added manually by the user. It is highly recommended that any new variables be added to a users netcdf.yaml file, following the formatting layout contained in that file. Otherwise, there may be issues with reading the resulting output using other CAMPS utilities.

Instantiating a CAMPS data object provides you with essentially an empty container, with some basic metadata, either autofilled or manually entered. The next steps would be to add actual data to the object, along with associated data such as time or location information. This can be added to the Camps_data object in the following way:

```
>>> wspd.data = numpy.random.rand(10,10)*50 # Assign data
>>> ptime = Time.PhenomenonTime(start_time='19910518', end_time='20180810')
>>> wspd.time.append(ptime)
```

Perhaps the user has a piece of metadata they would like to add that is unique to their data. Metadata attributes can always be added to a Camps_data object in the following way:

```
>>> wspd.metadata['my_important_attribute'] = 42
```

As the user works with their data, they will likely want to add information about what procedures have been performed on their data. To add a procedure to your Camps_data object, a Process object is instantiated, and appended to the Camps_data object.

Using the procedure ‘BiLinInterp’ as an example (predefined in procedures.yaml), there are two ways to add a Process object to your Camps_data object:

```
>>> wspd.add_process('BiLinInterp')
```

Another option is:

```
>>> p = camps.core.Process.Process(‘BiLinInterp’)
>>> wspd.processes.append(p)
```

Similar to netcdf.yaml, metadata for predefined procedures are stored in procedures.yaml. If the user wishes to take advantage of the “add_process” function, they need to add their procedure to that control file, following the format outlined there. Otherwise “empty” procedure objects can be instantiated, and their metadata manually added.

Lastly, since we’re writing to netcdf, dimensions of the variable data must be added to the Camps_data object. Dimensions can generally be named anything, but there are a few dimensions that have special properties, that act slightly differently. These are found in netcdf.yaml in the dimensions section. To add dimensions to the Camps_data object:

```
>>> wspd.add_dimensions('x','y')
```

The Camps_data object would now look as follows:

```
***** wind_speed ******
*
* dtype               : float64
* processes           : ( )
* dimensions          : ['x', 'y']
  Metadata:
* comment             : Wind speed is set to -9 if winds are variable.
* SOSA__observedProperty: StatPP__Data/Met/Moment/WindSpd
* name                : wind_speed
* valid_min           : 0.0
* coordinates         : elev
* long_name           : horizontal wind speed
* standard_name       : wind_speed
* my_important_attribute: 42
* valid_max           : 75.0
Shape:
(4, 3)
Data:
[[ 44.32559503  29.6
29957  48.87075532]]
```

The user will likely want to write their data to an output file. Once the Camps_data object has been instantiated and populated with data and metadata, writing output is simple:

```
>>> from camps.core.writer import write
>>> write(wspd, 'CAMPS_output.nc')
```

The CDL output from the newly created file would look like this:

```
$ ncdump output.nc

netcdf CAMPS_output {
dimensions:
    x = 4 ;
    y = 3 ;
variables:
    double WindSpd_instant(x, y) ;
            WindSpd_instant:_FillValue = 9999. ;
            WindSpd_instant:SOSA__observedProperty = "StatPP__Data/Met/Moment/WindSpd" ;
            WindSpd_instant:long_name = "horizontal wind speed" ;
            WindSpd_instant:valid_max = 75. ;
            WindSpd_instant:standard_name = "wind_speed" ;
            WindSpd_instant:comment = "Wind speed is set to -9 if winds are variable." ;
            WindSpd_instant:units = "m/s" ;
            WindSpd_instant:my_important_attribute = 42LL ;
            WindSpd_instant:coordinates = "latitude longitude" ;
            WindSpd_instant:ancillary_variables = "" ;
            WindSpd_instant:missing_value = 9999. ;
            WindSpd_instant:PROV__wasInformedBy = "( )" ;
            WindSpd_instant:SOSA__usedProcedure = "( )" ;

// global attributes:
            :institution = "NOAA/National Weather Service" ;
            :Conventions = "CF-1.7 CAMPS-1.2" ;
            :version = "CAMPS-1.2" ;
            :history = "" ;
            :references = "" ;
            :organization = "NOAA/MDL" ;
            :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
            :primary_variables = "WindSpd_instant" ;
data:

  WindSpd_instant =
    9.36115709400316, 40.0805602441551, 4.00665537148,
    40.9072218935792, 16.2853803224382, 23.8285486619925,
    2.557393430461, 18.4436592224694, 26.3832006293729,
    29.9961875737658, 17.9772550713641, 33.5965850685218 ;

group: prefix_list {

  // group attributes:
            :PROV__ = "http://www.w3.org/ns/prov/#" ;
            :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
            :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Known Issues

Graphing

The graphing functionality within the GUI section of the code base is currently broken. Naming schemes within CAMPS have been under active development, which has caused this functionality to become slightly out of date. Efforts are underway to update this feature so that it can be available for use again soon.

NWS Codes Registry

CAMPS is currently making use of NetCDF Linked-Data to reference the NWS codes-registry web resource. The codes-registry is also under active development. The look, purpose, and content of the NWS codes-registry will change as CAMPS matures as a software package.

Technical Description

Network Common Data Format (NetCDF) will be used as the primary CAMPS file format, as it supports the creation, access, and sharing of array-oriented scientific data. In addition, NetCDF also offers a self-describing data format with more flexibility than other widely used table driven formats such as TDLPACK or Gridded Binary (GRIB). Additionally, NetCDF has more widespread support within the StatPP community than other self-describing data formats like Hierarchical Data Format (HDF).

Metadata

A key motivation in establishing the NetCDF-CAMPS metadata structure, has been to develop a well-defined template for incorporating descriptive, well established, controlled vocabularies for StatPP output. Here CAMPS introduces an application profile, which utilizes 3 controlled vocabularies: NetCDF Climate and Forecast (CF) Conventions, PROV Family of Documents (PROV-O), and Sensor, Observation, Sample, and Actuator (SOSA). These data models and ontologies are already in use outside the U.S., and are hosted on the Web (supported by XML). CAMPS also makes use of NetCDF Linked-Data, which allows for the linkage of a given attribute, to an external descriptive source.

There are several different “types” of variables in a NetCDF-CAMPS file. Some contain numerical data, while others contain only descriptive metadata. The main types of variables are:

- **Primary variable**: All “primary_variables” are listed as a string, with space delimiters, under the “global attributes” section of a NetCDF-CAMPS file. These variables should satisfy CF-Convention rules, and be represented as a Camps_data object, with numerical data.
- **Metadata variable**: This type of variable contains no numerical data, only metadata attributes that help describe a procedure or some important characteristic of a primary variable.
- **Coordinate variable**: Our definition of a coordinate variable matches the definition outlined in NetCDF CF-Conventions.
    - **Time variable**: Any variable containing time information. Examples would be: phenomenonTime, forecast_reference_time, and lead_time. Most time variables are going to either be coordinate variables, or auxiliary coordinate variables, following CF-Conventions.
    - **Location variable**: These are variables which signify the horizontal axis of a primary variable. These will either be coordinate variables or auxiliary coordinate variables, following CF-Conventions.
- **Vertical coordinate variable**: These variables will contain the numerical data, and metadata, necessary to describe the vertical axis information for a primary variable. This information is attached to a primary variable via the vertical_coord metadata attribute, and is also included in the ancillary_variables attribute list.

By establishing this application profile, our goal is to improve the interoperability within the StatPP and Atmospheric Science community. This will also ensure that CAMPS data files contain sufficient metadata to be self-describing for relatively long periods of time. We introduce the shorthand NetCDF-CAMPS to describe a file that conforms to this application profile. The metadata properties outlined in this documentation suffice for the intended purposes of StatPP within MDL. However, any user of CAMPS is welcome, and even encouraged, to use any additional metadata properties necessary for their specific use case.

NetCDF CF-Conventions

NetCDF Climate and Forecast (CF) Conventions are meant to encourage the exchange of data created with the NetCDF API. These conventions provide a controlled vocabulary (standard names), and a controlled data structure. This allows descriptive metadata, and spatial and temporal properties, to be encoded for each variable within a file. These conventions are gaining in popularity in the atmospheric science community, with many new programs and applications using them as a standard. CAMPS aims to be fully NetCDF CF compliant, while including additional metadata ontologies, and unique CAMPS terminology where necessary.

While NetCDF CF-Conventions is widely used and a stable resource, it is also actively maintained. If a specific standard name, or method of representing data, is not possible under the current convention rules, there is a discussion page where new proposals may be submitted. If approved, they are published in a subsequent version of the CF Standard Name Table. The CAMPS team has, and will continue to, publish new standard names, as necessary. Additionally, there are guidelines for construction of CF Standard Names, which provides background for how CF Standard Names are constructed, and how new standard names should be developed. Included in this description are transformations of existing standard names, which should be utilized, if possible, before new

standard names are proposed. If no CF standard name, or transformation, adequately describes the given StatPP variable, method, or process, the most appropriate standard name should be used, while a proposal for a new standard name should be initiated.

Primary Variables: While not an official CF-Convention term, a primary variable within CAMPS should satisfy CF-Convention rules. A primary variable should contain the following CF-Conventions metadata attributes:

- standard_name - Should use the most appropriate standard name from the CF standard name table.
- long_name - A human readable definition of the variable.
- coordinates - Following CF conventions, this attribute specifies the auxiliary coordinate variables for the primary variable.
- ancillary_variables - Following CF conventions, this attribute specifies a variable, or procedure, that contains data, or information, closely associated with the given primary variable. The value should be a blank separated list of variable or procedure names, encoded as a single string.

CF-Convention Ancillary Variables: These attributes aid in describing the primary variable, and may or may not contain numerical data.

CF-Convention Coordinate Variables: A CF-Convention coordinate variable has a very strict definition. The variable can only have a single dimension, and that dimension must be the same name as the variable itself (ex: latitude(latitude)). The data must be numeric in type and be either monotonically increasing or decreasing, without missing values.

CF-Convention Auxiliary Coordinate Variables: A CF-Convention auxiliary coordinate variable helps make up for the flexibility that a coordinate variable lacks. These variables must contain coordinate data, but can be multi-dimensional, and there are no restrictions on the naming of the variable.

Globals: Every NetCDF-CAMPS file must contain the appropriate string representing the version of CF it follows.

Example:

```
// global attributes:
            :Conventions = "CF-1.7" ;
```

SOSA

The Semantic Sensor Network Ontology [SSN], developed by the World Wide Web Consortium (W3C), also includes a simple core ontology named SOSA (Sensor, Observation, Sample, and Actuator). This ontology introduces a data model and a controlled vocabulary that is useful for the description of both observations and forecasts of a given StatPP variable. Attributes can be easily recognized and the terminology is designed to apply across disciplines.

The most fundamental metadata concept in CAMPS is the **Observation** class. SOSA defines Observation as: “Act of carrying out an (*Observation*) *Procedure* to estimate or calculate a value of a property of a FeatureOfInterest. Links to a *Sensor* to describe what made the Observation and how; links to an *ObservableProperty* to describe what the result is an estimate of, and to a *FeatureOfInterest* to detail what that property was associated with.”

The procedure that estimates a temperature value at some point on the earth could be a human reading a mercury-in-glass thermometer, an automated observing platform reading a voltage from a sensor, or an numerical weather prediction system running on a supercomputer. All three are procedures; all can estimate temperature values. Thus, one can consider the difference between an observation and a forecast to be a difference in the estimating procedure used.

The following SOSA properties are necessary to fully comply and describe NetCDF-CAMPS data (italics represent related SOSA classes and properties the user should familiarize themselves with). Every primary_variable should contain these SOSA metadata attributes:

- observedProperty: “Relation linking an Observation to the property that was observed. The ObservableProperty should be a property of the FeatureOfInterest (linked by hasFeatureOfInterest) of this Observation.”

  Within a NetCDF-CAMPS file, every unique primary variable will have an observedProperty. For example: 500 mb temperature has an observedProperty of temperature. The value of the observedProperty attribute will resolve to a URI, usually an entry in the NWS Codes Registry. This URI will refer to a clear description of the primary_variable’s property. The URI need not describe any procedures performed to obtain the variable.

- phenomenonTime: “The time that the Result of an Observation, Actuation, or Sampling applies to the FeatureOfInterest. Not necessarily the same as the resultTime. May be an interval or an instant, or some other compound temporal entity [owl-time].”

  The unique definition of phenomenonTime, within the SOSA ontology, allows for a standardized time scheme to persist across all driver scripts that make up a CAMPS development. However, there are slight, but important, differences between the phenomenonTime for forecast model output vs. observations.

  The phenomenonTime for forecast model output is: forcast_reference_time (initialization date and time) + forecast_period (lead time). This effectively gives the time the phenomenon is forecast to occur.

  There is no forecast_reference_time, or forecast_period for observational data. Instead, the phenomenonTime is simply the time at which the phenomenon occurred.

- usedProcedure: “A relation to link to a reusable Procedure used in making an Observation, an Actuation, or a Sample, typically through a Sensor, Actuator or Sampler.”

  To expand on the idea of a Procedure within CAMPS, we look to the PROV-O ontology. This ontology allows for categorization of procedures. Those being applied to a dataset by the current user, or procedures which were previously applied to a dataset, either by a different user, or the current user at an earlier point in time. This is specifically relevant when performing a full CAMPS development, as a user executes each subsequent driver script.

  Within a NetCDF-CAMPS object, usedProcedures should be encoded in the same order that the procedures were/are performed. Each usedProcedure is encoded as a separate metadata variable within the file, containing no numerical data, whose attributes describe the usedProcedure.

PROV-O

The PROV Family of Documents (PROV) provides a model for provenance information (entities, activities, production of data). This information can be used to form assessments about the quality of the data. PROV allows for the exchange of such information using formats such as RDF and XML.

The PROV Ontology (PROV-O), “provides a set of classes, properties, and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts. It can also be specialized to create new classes and properties to model provenance information for different applications and domains.”

NetCDF-CAMPS utilizes this ontology to specify data sources, procedures, and inherited information about a given dataset. The following PROV-O properties are used to encode NetCDF-CAMPS output:

- entity- Is used exclusively for encoding statistics and coefficients within NetCDF-CAMPS equation output files. This attribute should resolve to a web based URI with a more robust description of the entity.
- activity - This attribute should be included for all procedure metadata variables. It should resolve to a web based URI providing more detailed information on the given procedure. Likely a NWS codes registry entry.
- used - Represents the entity that is being used by the given procedure. Where applicable, provides the source of the data for the procedure. Not required.
- wasInformedBy - The value of this attribute will contain strings representing the procedures performed on this data prior to the current application. This metadata attribute should be present for all primary_variables, but can be empty if appropriate.
- wasDerivedFrom - The value of this attribute will contain strings representing the individual pieces of information (often other primary variables) that went into deriving the given primary variable. This attribute will only be included for primary variables that have been computed during the current application.
- hadPrimarySource- The originating source of the data for a primary variable. Should be a predefined string representing source data recognized by CAMPS (Ex: METAR). New sources can easily be added.
- specializationOf- Used in NetCDF-CAMPS to link a very specific time variable to a broader time concept.

NetCDF Classic Linked-Data

NetCDF-CAMPS makes use of Linked Data (LD) for encoding and publishing data wherever possible. One key aspect of netCDF-LD that CAMPS takes advantage of is the mapping of global and variable attributes within a NetCDF-CAMPS file to URIs, using prefixes. Following the NetCDF-LD guidelines will allow NetCDF-CAMPS files to be represented within the Resource Description Framework (RDF). This furthers the versatility goal of CAMPS by allowing the use of other linked data technologies.

For example, CAMPS uses the SOSA__ prefix to designate concepts that had their origin in SOSA. In order to fully adhere to the Netcdf-CAMPS structure, there will be multiple prefixes in any given CAMPS file. The user should include NetCDF group attributes which identify the necessary prefixes within the file. The 'double underscore' character pair: __ is used as an identifier and as the termination of the prefix; the double underscore is part of the prefix.

CDL output showing the prefixes from a NetCDF-CAMPS file:

::

```
group: prefix_list {
// group attributes:
:StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
:SOSA__ = "http://www.w3.org/ns/sosa/" ;
:PROV__ = "http://www.w3.org/ns/prov/#" ;
} // group prefix_list
}
```

CDL output showing how a prefix is used to link a variable's metadata to external web resources:

::

short Temp_instant_2m(phenomenonTime, stations) ;

Temp_instant_2m:_FillValue = 9999s ; Temp_instant_2m:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ; Temp_instant_2m:long_name = "dry bulb temperature" ; Temp_instant_2m:valid_min = -80s ; Temp_instant_2m:valid_max = 130s ; Temp_instant_2m:standard_name = "air_temperature" ; Temp_instant_2m:units = "degF" ; Temp_instant_2m:vertical_coord = "elev0" ; Temp_instant_2m:PROV__hadPrimarySource = "METAR" ; Temp_instant_2m:coordinates = "phenomenonTime latitude longitude" ; Temp_instant_2m:ancillary_variables = "elev0 phenomenonTime DecodeBUFR METARQC " ; Temp_instant_2m:missing_value = 9999s ; Temp_instant_2m:PROV__wasInformedBy = "( )" ; Temp_instant_2m:SOSA__usedProcedure = "(DecodeBUFR METARQC)" ;

In the CDL output above SOSA__observedProperty is set to the path StatPP__Data/Met/Temp/Temp. StatPP__ has been given the prefix definition of http://codes.nws.noaa.gov/StatPP/. Thus if one replaces the prefix with it's definition they get the full URI to the codes registry entry for temperature (http://codes.nws.noaa.gov/StatPP/Data/Met/Temp/Temp). This same technique is also applied to other prefixes followed by double underscores in CDL output above "SOSA__observedProperty". By replacing SOSA__ with it's prefix definition, the user will be directed to http://www.w3.org/ns/sosa/observedProperty, which is the documentation describing SOSA observedProperty.

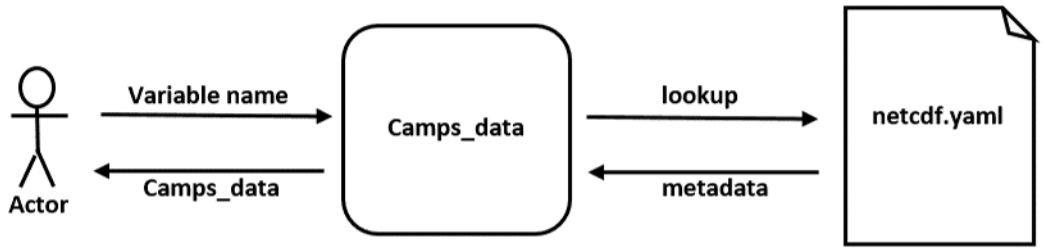
Data Format

Camps_data object:

The core data structure of CAMPS is the Camps_data object. This is defined as: a class containing components which fully describe a given variable. Many of these components can be written to a netcdf file independently of the parent object. Classes like Time, Location, and Process all share a common interface, *nc_writable*, which enforces the definition of the method *write_to_nc*.

The data contained in a Camps_data object can be gridded or vector, it may or may not have a lead time (forecast period) associated with it, and can be n-dimensional. For example, assume there is a variable that is gridded and includes lead times at multiple forecast reference times. This would have 4 dimensions; time, lead time, y, and x, and is a snapshot of a single time. Another type of variable could be a vector of stations. This variable could be one-dimensional as a snapshot of a single time, two-dimensional covering multiple forecast reference times, or three-dimensional and include multiple lead times.These dimensions would be number of stations, lead time, and time. It is also possible to have a Camps_data object with no numerical data, only metadata.

When instantiating a Camps_data object, the variable name provided is used to look up pre-defined metadata from the control file *netcdf.yaml*. This metadata is added to the Camps_data object for the variable. The name provided must match a variable listed in netcdf.yaml for the autofill feature to work. Alternative names for existing variables can be added to the bottom of the netcdf.yaml control file.



Dimensions:

Standard dimension names are located in netcdf.yaml in the 'Dimensions' section. Allowing for the flexibility of the object, the number of dimensions of the data are unlimited.

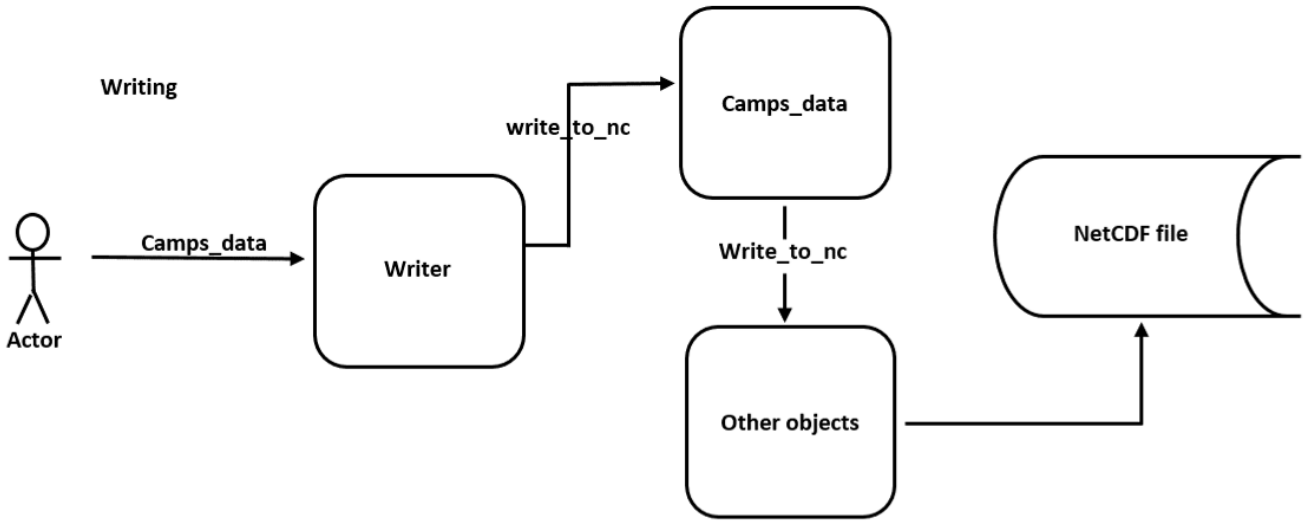
Reading:

When reading from a NetCDF input file, either a unique variable name should be provided to the camps.core.reader.read_var() function, or specific metadata should be provided via the template control file pred.yaml. If no name is provided, the metadata is used to search for the desired primary variable from the input file, by matching specific metadata attributes.. If a phenomenonTime or lead_time is also given, CAMPS will slice the retrieved primary variable based on those parameters.

Writing

Writing a variable to a NetCDF file is achieved by invoking the Camps_data object's write_to_nc function. This will call the appropriate write_to_nc function for any composite object of the main Camps_data object.

Camps_data.write_to_nc() uses a core metadata structure to generate unique variable names when writing to NetCDF. Additionally, this function helps format and filter the metadata written to the output file.



Time Coordinates:

NetCDF-CAMPS makes use of 4 distinct classes to handle the differences. They are; ForecastReferenceTime, LeadTime, phenomenonTime, and phenomenonTimePeriod.

The following table provides explanations of these time variables:

Phenomenon Time

The SOSA property phenomenonTime can be of type instant or period. This makes phenomenonTime appropriate for weather elements valid at both points in time (e.g., temperature, wind speed) and spans of time (e.g., event probabilities, precipitation accumulations).

A phenomenonTime variable is linked to the appropriate primary variable by including it in the coordinates attribute, and the ancillary_variables attribute list. This follows CF-Conventions by making this multi-dimensional time variable an auxiliary coordinate variable. We recognize that in many applications a combination of forecast reference time and lead time can convey the same content as phenomenonTime. However, in NetCDF-CAMPS, it is necessary to create an auxiliary coordinate variable explicitly for this purpose and assign it the appropriate attribute. The purpose is to limit implicit metadata wherever possible.

In NetCDF-CAMPS, a period of time has an instant when it begins, an instant when it ends, and a duration. Generally knowledge of any two of the three will be sufficient to compute the third.

CF-conventions describes a method for identifying vertices on a time axis which can then be designated period bounds. While this method is useful for many applications, we note a number of StatPP applications where it is inadequate. (E.g., this method cannot describe overlapping time periods.) Instead we provide an alternative which is more general and better-suited to StatPP.

For NetCDF-CAMPS time period variables are encoded using the syntax phenomenonTimePeriod. The PROV-O concept specializationOf can take on multiple values. In this example, the variable phenomenonTimePeriod is a specialization of the SOSA concept phenomenonTime and it is also a specialization of the concept of the NetCDF-CAMPS time bounds syntax called “BeginEnd”.

Example:

```
|  SOSA__phenomenonTimePeriod:PROV__specializationOf =
|  "StatPP__concepts/TimeBoundsSyntax/BeginEnd", "SOSA__phenomenonTime";
```

If there are multiple phenomenonTimePeriod variables in a single file, the duration (in hours) is appended to the end of the variable name. These variables should include a dimension (the one that varies fastest) of 2. That dimension can be interpreted as either a duration, beginning, or ending.

In order to properly associate the phenomenonTimePeriod variable with the primary variable it represents, it should be added to the coordinates attribute of the primary variable, as well as the ancillary_variables attribute list. By adding the phenomenonTimePeriod variable to the coordinates attribute, we are declaring it an auxiliary coordinate variable, according to CF-Conventions.

Forecast Reference Time

The concept of forecast_reference_time is defined in NetCDF CF-Conventions as “The ‘data time’, the time of the analysis from which the forecast was made”. The forecast_reference_time is an instant in time. It is frequently used to define the epoch of the time coordinate for variables associated with numerical weather prediction output. For data where this variable is appropriate, the syntax used for this variable is FcstRefTime, and will have the standard name forecast_reference_time. In order to link this time variable to the appropriate primary variable, it should be included in the ancillary_variables attribute list.

Lead Time

The term “lead time” is often used to describe a duration of time that is measured from a forecast_reference_time to the time when some phenomenon is observed or forecast to occur. There are a number of expressions that are commonly used to describe this concept (e.g., forecast period, forecast lead, time projection). In NetCDF-CAMPS, the syntax used to encode a lead time variable is lead_time. If there are multiple lead time variables in a single file, an identifying number will be appended to the end of the variable name. According to CF-Conventions, the standard name for lead time is forecast_period. The standard name should be included as an attribute for any lead time variable. .In NetCDF-CAMPS datasets, where this concept is meaningful, a variable of appropriate dimensionality should be defined and contain lead time values. This variable should have the attribute PROV__specializationOf declared with a value that expresses the concept of lead time. Ideally this value will use a URI and be machine-readable.

This variable should be encoded as a CF-Conventions coordinate variable, when appropriate. Otherwise, it should be included in the ancillary_variables attribute list for the appropriate primary variable.

Output Examples

Sample metar_driver Output

The example CDL output, from a metar_driver output file, is shown for 2 meter temperature, for 384 hours, and 2584 stations. Quality Control was applied to the data, and the metadata variables reflect that procedure application. The necessary time and location variables are also included in the output.

```
netcdf camps_metar_output {
dimensions:
      stations = 2584 ;
      phenomenonTime = 384 ;
      level = 1 ;
      nv = 2 ;
variables:
      int64 DecodeBUFR ;
              DecodeBUFR:PROV__activity = "StatPP__Methods/Ingest/DecodeBUFR" ;
              DecodeBUFR:PROV__used = "StatPP__Data/Source/NCEPSfcObsMETAR" ;
              DecodeBUFR:long_name = "Ingest BUFR encoded METAR observations from NCEP repository" ;
              DecodeBUFR:standard_name = "source" ;
      int64 METARQC ;
              METARQC:PROV__activity = "StatPP__Methods/QC/METARQC" ;
              METARQC:PROV__wasInformedBy = "DecodeBUFR" ;
              METARQC:long_name = "Apply MDL METAR Quality Control procedure" ;
              METARQC:standard_name = "source" ;
      string stations(stations) ;
              string stations:_FillValue = "_" ;
              stations:long_name = "ICAO METAR call letters" ;
              stations:standard_name = "platform_id" ;
              stations:comment = " Only currently archives reports from stations only if the first letter of the ICAO ID is \'K\', \'P\', \'M\', \'C\', or \'T\'. " ;
              stations:coordinates = "latitude longitude" ;
              stations:missing_value = 9999LL ;
      byte station_type(stations) ;
              station_type:_FillValue = 15b ;
              station_type:standard_name = "platform_id" ;
              station_type:long_name = "station type" ;
              station_type:PROV__hadPrimarySource = "METAR" ;
              station_type:coordinates = "latitude longitude" ;
              station_type:missing_value = 9999LL ;
      double latitude(stations) ;
              latitude:_FillValue = 9999. ;
              latitude:long_name = "latitude" ;
              latitude:units = "degrees_north" ;
              latitude:valid_min = -90. ;
              latitude:valid_max = 90. ;
              latitude:standard_name = "latitude" ;
              latitude:PROV__hadPrimarySource = "METAR" ;
              latitude:coordinates = "latitude longitude" ;
              latitude:missing_value = 9999. ;
      double longitude(stations) ;
              longitude:_FillValue = 9999. ;
              longitude:long_name = "longitude" ;
              longitude:units = "degrees_west" ;
              longitude:valid_min = -180. ;
              longitude:valid_max = 180. ;
              longitude:standard_name = "longitude" ;
              longitude:PROV__hadPrimarySource = "METAR" ;
              longitude:coordinates = "latitude longitude" ;
              longitude:missing_value = 9999. ;
      int64 phenomenonTime(phenomenonTime) ;
              phenomenonTime:_FillValue = 9999LL ;
              phenomenonTime:calendar = "gregorian" ;
              phenomenonTime:units = "seconds since 1970-01-01 00:00:00.0" ;
              phenomenonTime:standard_name = "time" ;
              phenomenonTime:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      short Temp_instant_2m(phenomenonTime, stations) ;
              Temp_instant_2m:_FillValue = 9999s ;
              Temp_instant_2m:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
              Temp_instant_2m:long_name = "dry bulb temperature" ;
              Temp_instant_2m:valid_min = -80s ;
              Temp_instant_2m:valid_max = 130s ;
              Temp_instant_2m:standard_name = "air_temperature" ;
              Temp_instant_2m:units = "degF" ;
              Temp_instant_2m:vertical_coord = "elev0" ;
              Temp_instant_2m:PROV__hadPrimarySource = "METAR" ;
              Temp_instant_2m:coordinates = "phenomenonTime latitude longitude" ;
              Temp_instant_2m:ancillary_variables = "elev0 phenomenonTime DecodeBUFR METARQC " ;
              Temp_instant_2m:missing_value = 9999s ;
              Temp_instant_2m:PROV__wasInformedBy = "( )" ;
              Temp_instant_2m:SOSA__usedProcedure = "( DecodeBUFR METARQC )" ;
// global attributes:
              :institution = "NOAA/National Weather Service" ;
              :Conventions = "CF-1.7 CAMPS-1.2" ;
              :version = "CAMPS-1.2" ;
              :history = "" ;
              :references = "" ;
              :organization = "NOAA/MDL" ;
              :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
              :primary_variables = "Temp_instant_2m longitude latitude stations station_type" ;
  // group attributes:
              :PROV__ = "http://www.w3.org/ns/prov/#" ;
              :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
              :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample marine_driver Output

Marine_driver output is extremely similar to Metar_driver output. The difference is the procedures, stations, and source of information. Everything else is effectively the same, for the same time range and variables.

```
netcdf camps_marine_output {
dimensions:
      stations = 328 ;
      phenomenonTime = 384 ;
      level = 1 ;
      nv = 2 ;
variables:
      string stations(stations) ;
              string stations:_FillValue = "_" ;
              stations:long_name = "NDBC station identifiers" ;
              stations:standard_name = "platform_id" ;
              stations:comment = "NDBC stations consist of buoy, C-MAN, and platform drilling sites" ;
              stations:PROV__hadPrimarySource = "NDBC" ;
              stations:coordinates = "latitude longitude" ;
              stations:missing_value = 9999LL ;
      byte station_type(stations) ;
              station_type:_FillValue = 15b ;
              station_type:standard_name = "platform_id" ;
              station_type:long_name = "station type" ;
              station_type:PROV__hadPrimarySource = "NDBC" ;
              station_type:coordinates = "latitude longitude" ;
              station_type:missing_value = 9999LL ;
      double latitude(stations) ;
              latitude:_FillValue = 9999. ;
              latitude:long_name = "latitude" ;
              latitude:units = "degrees_north" ;
              latitude:valid_min = -90. ;
              latitude:valid_max = 90. ;
              latitude:standard_name = "latitude" ;
              latitude:coordinates = "latitude longitude" ;
              latitude:missing_value = 9999. ;
      double longitude(stations) ;
              longitude:_FillValue = 9999. ;
              longitude:long_name = "longitude" ;
              longitude:units = "degrees_west" ;
              longitude:valid_min = -180. ;
              longitude:valid_max = 180. ;
              longitude:standard_name = "longitude" ;
              longitude:coordinates = "latitude longitude" ;
              longitude:missing_value = 9999. ;
      int64 elev0(level) ;
              elev0:long_name = "height above surface" ;
              elev0:units = "m" ;
              elev0:standard_name = "height" ;
              elev0:positive = "up" ;
              elev0:axis = "Z" ;
      int64 phenomenonTime(phenomenonTime) ;
              phenomenonTime:_FillValue = 9999LL ;
              phenomenonTime:calendar = "gregorian" ;
              phenomenonTime:units = "seconds since 1970-01-01 00:00:00.0" ;
              phenomenonTime:standard_name = "time" ;
              phenomenonTime:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      int64 ProcMarine ;
              ProcMarine:PROV__activity = "StatPP__Methods/Ingest/DecodeTabularText" ;
              ProcMarine:PROV__used = "StatPP__Data/Source/NDBC" ;
              ProcMarine:long_name = "Decode tabular text data" ;
```

```
              ProcMarine:standard_name = "source" ;
       int64 MarineQC ;
              MarineQC:PROV__activity = "StatPP__Methods/QC/MarineQC" ;
              MarineQC:PROV__wasInformedBy = "ProcMarine" ;
              MarineQC:long_name = "Marine Observation Quality Control" ;
              MarineQC:standard_name = "source" ;
       short Temp_instant_2m(phenomenonTime, stations) ;
              Temp_instant_2m:_FillValue = 9999s ;
```

```
              Temp_instant_2m:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
              Temp_instant_2m:long_name = "dry bulb temperature" ;
              Temp_instant_2m:valid_min = -80s ;
              Temp_instant_2m:valid_max = 130s ;
              Temp_instant_2m:standard_name = "air_temperature" ;
              Temp_instant_2m:units = "degF" ;
              Temp_instant_2m:vertical_coord = "elev0" ;
              Temp_instant_2m:PROV__hadPrimarySource = "NDBC" ;
              Temp_instant_2m:coordinates = "phenomenonTime latitude longitude" ;
              Temp_instant_2m:ancillary_variables = "elev0 phenomenonTime ProcMarine MarineQC " ;
              Temp_instant_2m:missing_value = 9999s ;
              Temp_instant_2m:PROV__wasInformedBy = "( )" ;
              Temp_instant_2m:SOSA__usedProcedure = "( ProcMarine MarineQC )" ;
// global attributes:
              :institution = "NOAA/National Weather Service" ;
              :Conventions = "CF-1.7 CAMPS-1.2" ;
              :version = "CAMPS-1.2" ;
              :history = "" ;
              :references = "" ;
              :organization = "NOAA/MDL" ;
              :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
              :primary_variables = "Temp_instant_2m longitude latitude stations station_type" ;
group: prefix_list {
  // group attributes:
              :PROV__ = "http://www.w3.org/ns/prov/#" ;
              :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
              :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample grib2_to_nc_driver output

This example is CDL output from grib2_to_nc_driver, for 2 meter temperature. Primary variables from this driver script will include lead_times in the dimensions, unlike the other driver scripts which process model data. This makes encoding time for grib2_to_nc_driver output look slightly different from all the other driver scripts.

```
netcdf camps_grib2_output {
dimensions:
      level = 1 ;
      lead_times = 33 ;
      phenomenonTime = 10 ;
      y = 169 ;
      x = 297 ;
      nv = 2 ;
variables:
      int64 elev0(level) ;
              elev0:long_name = "height above surface" ;
              elev0:units = "m" ;
              elev0:standard_name = "height" ;
              elev0:positive = "up" ;
              elev0:axis = "Z" ;
      int64 phenomenonTimes(lead_times, phenomenonTime) ;
              phenomenonTimes:_FillValue = 9999LL ;
              phenomenonTimes:calendar = "gregorian" ;
              phenomenonTimes:units = "seconds since 1970-01-01 00:00:00.0" ;
              phenomenonTimes:standard_name = "time" ;
              phenomenonTimes:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      int64 FcstRefTime(phenomenonTime) ;
              FcstRefTime:_FillValue = 9999LL ;
              FcstRefTime:calendar = "gregorian" ;
              FcstRefTime:units = "seconds since 1970-01-01 00:00:00.0" ;
              FcstRefTime:standard_name = "forecast_reference_time" ;
              FcstRefTime:PROV__specializationOf = "( StatPP__Data/Time/FcstRefTime )" ;
      int64 lead_times(lead_times) ;
              lead_times:_FillValue = 9999LL ;
```

```
              lead_times:units = "seconds" ;
              lead_times:standard_name = "forecast_period" ;
              lead_times:PROV__specializationOf = "( StatPP__Data/Time/LeadTime )" ;
              lead_times:firstLeadTime = "P0H" ;
              lead_times:PeriodicTime = "P3H" ;
              lead_times:lastLeadTime = "P96H" ;
      int64 FilterGRIB2 ;
              FilterGRIB2:PROV__activity = "StatPP__Methods/Ingest/FilterGRIB2" ;
              FilterGRIB2:PROV__used = "StatPP__Data/Source/GFS13" ;
              FilterGRIB2:long_name = "Filter GRIB2-encoded forecasts" ;
              FilterGRIB2:standard_name = "source" ;
      int64 ResampleGRIB2 ;
              ResampleGRIB2:PROV__activity = "StatPP__Methods/Ingest/ResampleGRIB2" ;
              ResampleGRIB2:PROV__wasInformedBy = "FilterGRIB2" ;
```

```
            ResampleGRIB2:long_name = "Resampling of GRIB2 data onto a new grid" ;
            ResampleGRIB2:standard_name = "source" ;
      float Temp_instant_2m_00Z(phenomenonTime, lead_times, y, x) ;
            Temp_instant_2m_00Z:_FillValue = 9999.f ;
            Temp_instant_2m_00Z:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
            Temp_instant_2m_00Z:long_name = "temperature instant" ;
            Temp_instant_2m_00Z:standard_name = "air_temperature" ;
            Temp_instant_2m_00Z:units = "K" ;
            Temp_instant_2m_00Z:PROV__hadPrimarySource = "GFS" ;
            Temp_instant_2m_00Z:FcstTime_hour = 0LL ;
            Temp_instant_2m_00Z:grid_mapping = "polar_stereographic_grid" ;
            Temp_instant_2m_00Z:vertical_coord = "elev1" ;
            Temp_instant_2m_00Z:coordinates = "phenomenonTimes latitude longitude" ;
            Temp_instant_2m_00Z:ancillary_variables = "elev1 phenomenonTimes FcstRefTime lead_times FilterGRIB2 ResampleGRIB2 " ;
            Temp_instant_2m_00Z:missing_value = 9999.f ;
            Temp_instant_2m_00Z:PROV__wasInformedBy = "( FilterGRIB2 ResampleGRIB2 )" ;
            Temp_instant_2m_00Z:SOSA__usedProcedure = "( )" ;
      double polar_stereographic_grid ;
            polar_stereographic_grid:_FillValue = 9999. ;
            polar_stereographic_grid:grid_mapping_name = "polar_stereographic" ;
            polar_stereographic_grid:straight_vertical_longitude_from_pole = 255. ;
            polar_stereographic_grid:latitude_of_projection_origin = 90. ;
            polar_stereographic_grid:standard_parallel = 60. ;
            polar_stereographic_grid:scale_factor_at_projection_origin = 1LL ;
            polar_stereographic_grid:PROJ_string = "+a=6371229 +b=6371229 +proj=stere +lat_ts=60.0 +lat_0=90.0 +lon_0=255.0 +x_0=8001120.943743923 +y_0=8001120.943743925" ;
            polar_stereographic_grid:coordinates = "" ;
// global attributes:
            :institution = "NOAA/National Weather Service" ;
            :Conventions = "CF-1.7 CAMPS-1.2" ;
            :version = "CAMPS-1.2" ;
            :history = "" ;
            :references = "" ;
            :organization = "NOAA/MDL" ;
            :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
            :primary_variables = "Temp_instant_2m_00Z latitude longitude x y polar_stereographic_grid"" ;
group: prefix_list {
  // group attributes:
            :PROV__ = "http://www.w3.org/ns/prov/#" ;
            :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
            :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample mospred_driver output (predictors)

Mospred_driver will have two potential output files. This example is the CDL output from processing predictors only. Notice that lead_times is no longer a dimension of our primary variable, 2 meter temperature. Instead, primary variables will be broken into separate variables based on procedures, level, AND lead_time.

```
netcdf camps_predictor_output {
dimensions:
      phenomenonTime = 10 ;
      stations = 20 ;
      level = 1 ;
      lead_times = 1 ;
variables:
      int64 elev0(level) ;
            elev0:long_name = "height above surface" ;
            elev0:units = "m" ;
            elev0:standard_name = "height" ;
            elev0:positive = "up" ;
            elev0:axis = "Z" ;
      int64 phenomenonTimes(phenomenonTime) ;
            phenomenonTimes:_FillValue = 9999LL ;
            phenomenonTimes:calendar = "gregorian" ;
            phenomenonTimes:units = "seconds since 1970-01-01 00:00:00.0" ;
            phenomenonTimes:standard_name = "time" ;
            phenomenonTimes:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      int64 FcstRefTime(phenomenonTime) ;
            FcstRefTime:_FillValue = 9999LL ;
            FcstRefTime:calendar = "gregorian" ;
            FcstRefTime:units = "seconds since 1970-01-01 00:00:00.0" ;
            FcstRefTime:standard_name = "forecast_reference_time" ;
            FcstRefTime:PROV__specializationOf = "( StatPP__Data/Time/FcstRefTime )" ;
      int64 lead_times(lead_times) ;
            lead_times:_FillValue = 9999LL ;
            lead_times:units = "seconds" ;
            lead_times:standard_name = "forecast_period" ;
            lead_times:PROV__specializationOf = "( StatPP__Data/Time/LeadTime )" ;
            lead_times:firstLeadTime = "P0H" ;
            lead_times:PeriodicTime = "P3H" ;
            lead_times:lastLeadTime = "P96H" ;
```

```
      int64 FilterGRIB2 ;
            FilterGRIB2:PROV__activity = "StatPP__Methods/Ingest/FilterGRIB2" ;
            FilterGRIB2:PROV__used = "StatPP__Data/Source/GFS13" ;
            FilterGRIB2:long_name = "Filter GRIB2-encoded forecasts" ;
            FilterGRIB2:standard_name = "source" ;
      int64 ResampleGRIB2 ;
            ResampleGRIB2:PROV__activity = "StatPP__Methods/Ingest/ResampleGRIB2" ;
            ResampleGRIB2:PROV__wasInformedBy = "FilterGRIB2" ;
            ResampleGRIB2:long_name = "Resampling of GRIB2 data onto a new grid" ;
            ResampleGRIB2:standard_name = "source" ;
      int64 LinSmooth ;
            LinSmooth:PROV__activity = "StatPP__Methods/Arith/LinSmooth" ;
            LinSmooth:long_name = "Linear Smoothing" ;
      int64 BiLinInterp ;
            BiLinInterp:PROV__activity = "StatPP__Methods/Geosp/BiLinInterp" ;
            BiLinInterp:long_name = "Bilinear Interpolation" ;
      byte station_type(stations) ;
```

```
            station_type:_FillValue = 15b ;
            station_type:standard_name = "platform_id" ;
            station_type:long_name = "station type" ;
            station_type:PROV__hadPrimarySource = "METAR" ;
            station_type:coordinates = "latitude longitude" ;
            station_type:missing_value = 9999LL ;
      string stations(stations) ;
            string stations:_FillValue = "_" ;
            stations:long_name = "NDBC station identifiers" ;
            stations:standard_name = "platform_id" ;
            stations:comment = "NDBC stations consist of buoy, C-MAN, and platform drilling sites" ;
            stations:PROV__hadPrimarySource = "NDBC" ;
            stations:coordinates = "latitude longitude" ;
            stations:missing_value = 9999LL ;
      double Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp(phenomenonTime, stations) ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:_FillValue = 9999. ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:long_name = "temperature instant" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:standard_name = "air_temperature" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:units = "K" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:PROV__hadPrimarySource = "GFS" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:FcstTime_hour = 0LL ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:vertical_coord = "elev0" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:coordinates = "phenomenonTimes latitude longitude" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:ancillary_variables = "elev0 phenomenonTimes FcstRefTime lead_times FilterGRIB2 ResampleGRIB2 LinSmooth BiLinInterp " ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:missing_value = 9999. ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:PROV__wasInformedBy = "( FilterGRIB2 ResampleGRIB2 )" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:smooth = "25" ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:leadtime = 12LL ;
            Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp:SOSA__usedProcedure = "( LinSmooth BiLinInterp )" ;
// global attributes:
            :institution = "NOAA/National Weather Service" ;
            :Conventions = "CF-1.7 CAMPS-1.2" ;
            :version = "CAMPS-1.2" ;
            :history = "" ;
            :references = "" ;
            :organization = "NOAA/MDL" ;
            :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
            :primary_variables = "Temp_instant_2m_00Z_12hr_LinSmooth25_BiLinInterp longitude latitude stations station_type" ;
group: prefix_list {
  // group attributes:
            :PROV__ = "http://www.w3.org/ns/prov/#" ;
            :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
            :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample mospred_driver output (predictands)

Mospred_driver will have two potential output files. This example is the CDL output from processing predictands only. The main difference between mospred predictor output and predictand output is how time is encoded. For predictands, there is no lead_time or forecast_reference_time. Instead, there is only a phenomenonTime. Predictands are still split out by procedure and level.

```
netcdf camps_predictand_output {
dimensions:
      phenomenonTime = 10 ;
      stations = 20 ;
      level = 1 ;
variables:
      int64 elev0(level) ;
            elev0:long_name = "height above surface" ;
            elev0:units = "m" ;
            elev0:standard_name = "height" ;
            elev0:positive = "up" ;
            elev0:axis = "Z" ;
      int64 phenomenonTime(phenomenonTime) ;
            phenomenonTime:_FillValue = 9999LL ;
            phenomenonTime:calendar = "gregorian" ;
            phenomenonTime:units = "seconds since 1970-01-01 00:00:00.0" ;
            phenomenonTime:standard_name = "time" ;
            phenomenonTime:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      int64 DecodeBUFR ;
            DecodeBUFR:PROV__activity = "StatPP__Methods/Ingest/DecodeBUFR" ;
            DecodeBUFR:PROV__used = "StatPP__Data/Source/NCEPSfcObsMETAR" ;
            DecodeBUFR:long_name = "Ingest BUFR encoded METAR observations from NCEP repository" ;
            DecodeBUFR:standard_name = "source" ;
      int64 METARQC ;
            METARQC:PROV__activity = "StatPP__Methods/QC/METARQC" ;
            METARQC:PROV__wasInformedBy = "DecodeBUFR" ;
            METARQC:long_name = "Apply MDL METAR Quality Control procedure" ;
            METARQC:standard_name = "source" ;
      double Temp_instant_2m(phenomenonTime, stations) ;
            Temp_instant_2m:_FillValue = 9999. ;
            Temp_instant_2m:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
```

```
            Temp_instant_2m:long_name = "dry bulb temperature" ;
            Temp_instant_2m:valid_min = 210.927777777778 ;
            Temp_instant_2m:valid_max = 327.594444444444 ;
            Temp_instant_2m:standard_name = "air_temperature" ;
            Temp_instant_2m:units = "K" ;
            Temp_instant_2m:vertical_coord = "elev0" ;
            Temp_instant_2m:coordinates = "phenomenonTime latitude longitude" ;
            Temp_instant_2m:ancillary_variables = "elev0 phenomenonTime DecodeBUFR METARQC " ;
            Temp_instant_2m:missing_value = 9999. ;
            Temp_instant_2m:PROV__wasInformedBy = "( DecodeBUFR METARQC )" ;
            Temp_instant_2m:PROV__hadPrimarySource = "METAR NDBC" ;
            Temp_instant_2m:SOSA__usedProcedure = "( )" ;
      byte station_type(stations) ;
            station_type:_FillValue = 15b ;
```

```
            station_type:standard_name = "platform_id" ;
            station_type:long_name = "station type" ;
            station_type:PROV__hadPrimarySource = "METAR" ;
            station_type:coordinates = "latitude longitude" ;
            station_type:missing_value = 9999LL ;
      string stations(stations) ;
```

```
             string stations:_FillValue = "_" ;
             stations:long_name = "ICAO METAR call letters" ;
             stations:standard_name = "platform_id" ;
             stations:comment = " Only currently archives reports from stations only if the first letter of the ICAO ID is \'K\', \'P\', \'M\', \'C\', or \'T\'. " ;
             stations:coordinates = "latitude longitude" ;
             stations:missing_value = 9999LL ;
     double latitude(stations) ;
             latitude:_FillValue = 9999. ;
             latitude:long_name = "latitude" ;
             latitude:units = "degrees_north" ;
             latitude:valid_min = -90. ;
             latitude:valid_max = 90. ;
             latitude:standard_name = "latitude" ;
             latitude:coordinates = "latitude longitude" ;
             latitude:missing_value = 9999. ;
     double longitude(stations) ;
             longitude:_FillValue = 9999. ;
             longitude:long_name = "longitude" ;
             longitude:units = "degrees_west" ;
             longitude:valid_min = -180. ;
             longitude:valid_max = 180. ;
             longitude:standard_name = "longitude" ;
             longitude:coordinates = "latitude longitude" ;
             longitude:missing_value = 9999. ;

// global attributes:
             :institution = "NOAA/National Weather Service" ;
             :Conventions = "CF-1.7 CAMPS-1.2" ;
             :version = "CAMPS-1.2" ;
             :history = "" ;
             :references = "" ;
             :organization = "NOAA/MDL" ;
             :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
             :primary_variables = "Temp_instant_2m stations station_type latitude longitude" ;

group: prefix_list {

  // group attributes:
             :PROV__ = "http://www.w3.org/ns/prov/#" ;
             :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
             :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample equations_driver output

Regression coefficients, and the *Equation_Constant*, are saved as a single variable (*MOS_Equations*), dimensioned by the number of stations, the number of coefficients (including the *Equation_Constant*) and the number of Predictands.

The ancillary_variables attribute, for the *MOS_equations* variable, should contain a reference to the non-data-bearing variables *MOS_Predictor_Coeffs* and *Equation_Constant*. These variables contain the appropriate metadata description to identify the type of information a *MOS_equations* variable contains.

The auxiliary coordinate variable attribute (coordinates), for a *MOS_equations* variable, references an *Equations_List* and *Predictand_List* variable. These variables provide ordered lists that apply to the dimensions of the *MOS_equations* variable.

Equations_List corresponds to the dimension *max_eq_terms*, which stores an ordered list of the predictors that provide input to the set of equations, along with the equation constant, as a character array. The ordered input list references the non-data-bearing predictor variables that appear elsewhere in the file. The attributes assigned to each of those non-data-bearing variables, provide all the metadata needed to access and use those predictors contained in the input file to equations_driver.

Predictand_List provides an ordered character output array, which plays a complementary role by identifying the predictands that are forecast by the equations.

In addition to the MOS_equations variable, ordered input and output lists, and non-data-bearing predictor and predictand variables, there are a number of variables that contain diagnostic information about the MOS development process. Each of these variables is stored individually, as primary variables, within an equations output file.

```
netcdf camps_equations_output {
dimensions:
       stations = 20 ;
       number_of_predictands = 1 ;
       max_eq_terms = 4 ;
       num_char_predictors = 51 ;
       num_char_predictands = 20 ;
       level = 1 ;
       phenomenonTime = 10 ;
       lead_times = 1 ;
variables:
       char Equations_List(max_eq_terms, num_char_predictors) ;
```

```
            Equations_List:PROV__entity = "StatPP__Methods/Stat/OrdrdInpt" ;
            Equations_List:long_name = "Ordered List of Equation Terms" ;
      int64 PolyLinReg ;
            PolyLinReg:PROV__Activity = "StatPP__Methods/Stat/PolyLinReg" ;
            PolyLinReg:long_name = "Polynomial Linear Regression" ;
            PolyLinReg:feature_of_interest = "no" ;
      float Reduction_of_Variance(stations, number_of_predictands) ;
            Reduction_of_Variance:PROV__entity = "StatPP__Methods/Stat/MOS/OutptParams/MOSRedOfVar" ;
            Reduction_of_Variance:standard_name = "source" ;
            Reduction_of_Variance:long_name = "MOS Reduction of Variance" ;
            Reduction_of_Variance:coordinates = "station Predictand_List" ;
            Reduction_of_Variance:SOSA__usedProcedure = "( PolyLinReg )" ;
            Reduction_of_Variance:units = 1LL ;
      float Multiple_Correlation_Coefficient(stations, number_of_predictands) ;
            Multiple_Correlation_Coefficient:PROV__entity = "StatPP__Methods/Stat/MOS/OutptParams/MOSMultiCorCoef" ;
            Multiple_Correlation_Coefficient:standard_name = "source" ;
            Multiple_Correlation_Coefficient:long_name = "MOS Multiple Correlation Coefficient" ;
            Multiple_Correlation_Coefficient:coordinates = "station Predictand_List" ;
            Multiple_Correlation_Coefficient:SOSA__usedProcedure = "( PolyLinReg )" ;
            Multiple_Correlation_Coefficient:units = 1LL ;
      float Predictand_Average(stations, number_of_predictands) ;
            Predictand_Average:PROV__entity = "StatPP__Methods/MOS/Arith/Mean" ;
            Predictand_Average:standard_name = "source" ;
            Predictand_Average:long_name = "Predictand Average" ;
            Predictand_Average:coordinates = "station Predictand_List" ;
            Predictand_Average:SOSA__usedProcedure = "( Mean )" ;
            Predictand_Average:units = 1LL ;
      float MOS_Equations(stations, max_eq_terms, number_of_predictands) ;
            MOS_Equations:PROV__entity = "StatPP__Methods/Stat/MOS/MOSEqn" ;
            MOS_Equations:standard_name = "source" ;
            MOS_Equations:long_name = "MOS Equation Coefficients and Constants" ;
            MOS_Equations:coordinates = "station Equations_List Predictand_List" ;
            MOS_Equations:SOSA__usedProcedure = "( PolyLinReg )" ;
            MOS_Equations:ancillary_variables = "( MOS_Predictor_Coeffs Equation_Constant )" ;
            MOS_Equations:units = 1LL ;
// global attributes:
            :PROV__entity = "StatPP__Data/Source/MOSEqnFile" ;
            :PROV__wasGeneratedBy = "StatPP__Methods/Stat/MOSDev" ;
            :primary_variables = "MOS_Equations Standard_Error_Estimate Reduction_of_Variance Multiple_Correlation_Coefficient Predictand_Average" ;
            :season = "warm" ;
            :StatPPTime__SeasBeginDay = "0401" ;
            :StatPPTime__SeasEndDay = "0410" ;
            :StatPPTime__SeasDayFmt = "StatPP__Data/Time/SeasDayFmt/MMDD" ;
            :StatPPTime__FcstCyc = "00" ;
            :StatPPTime__FcstCycFmt = "StatPP__Data/Time/FcstCycFmt/HH" ;
            :StatPPSystem__Status = "StatPP__Methods/System/Status/Dev" ;
            :institution = "NOAA/National Weather Service" ;
            :Conventions = "CF-1.7 CAMPS-1.2" ;
            :version = "CAMPS-1.2" ;
            :PROV__wasAttributedTo = "StatPP__Data/Source/MDL" ;
            :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
            :PROV__generatedAtTime = "2021-02-25T19:00:00" ;

group: prefix_list {

  // group attributes:
            :PROV__ = "http://www.w3.org/ns/prov/#" ;
            :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
            :StatPPTime__ = "http://codes.nws.noaa.gov/StatPP/Data/Time/" ;
            :StatPPSystem__ = "http://codes.nws.noaa.gov/StatPP/Methods/System/" ;
            :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Sample forecast_driver output

The CDL output for forecast_driver should look very similar to that of mospred_driver, for predictors. The main exception being the “MOS” identifier prepended to the start of every variable. There are also consistency check variables included in forecast_driver output, when appropriate.

```
netcdf camps_forecast_output {
dimensions:
      phenomenonTime = 10 ;
      stations = 20 ;
      level = 1 ;
      lead_times = 1 ;
variables:
      int64 elev0(level) ;
            elev0:long_name = "height above surface" ;
            elev0:units = "m" ;
            elev0:standard_name = "height" ;
            elev0:positive = "up" ;
            elev0:axis = "Z" ;
      int64 phenomenonTime(phenomenonTime) ;
            phenomenonTime:_FillValue = 9999LL ;
            phenomenonTime:calendar = "gregorian" ;
            phenomenonTime:units = "seconds since 1970-01-01 00:00:00.0" ;
            phenomenonTime:standard_name = "time" ;
```

```
              phenomenonTime:PROV__specializationOf = "( SOSA__phenomenonTime )" ;
      int64 FcstRefTime(phenomenonTime) ;
              FcstRefTime:_FillValue = 9999LL ;
              FcstRefTime:calendar = "gregorian" ;
              FcstRefTime:units = "seconds since 1970-01-01 00:00:00.0" ;
              FcstRefTime:standard_name = "forecast_reference_time" ;
              FcstRefTime:PROV__specializationOf = "( StatPP__Data/Time/FcstRefTime )" ;
      int64 lead_times(lead_times) ;
              lead_times:_FillValue = 9999LL ;
              lead_times:units = "seconds" ;
              lead_times:standard_name = "forecast_period" ;
              lead_times:PROV__specializationOf = "( StatPP__Data/Time/LeadTime )" ;
      int64 MOS_Method ;
              MOS_Method:PROV__activity = "StatPP__Methods/Stat/MOS" ;
              MOS_Method:long_name = "Model Output statistical method: Multiple Linear Regression " ;
      double MOS_Temp_instant_2m_00Z_12hr(phenomenonTime, stations) ;
              MOS_Temp_instant_2m_00Z_12hr:_FillValue = 9999. ;
              MOS_Temp_instant_2m_00Z_12hr:SOSA__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
              MOS_Temp_instant_2m_00Z_12hr:long_name = "dry bulb temperature" ;
              MOS_Temp_instant_2m_00Z_12hr:valid_min = 210.927777777778 ;
              MOS_Temp_instant_2m_00Z_12hr:valid_max = 327.594444444444 ;
              MOS_Temp_instant_2m_00Z_12hr:standard_name = "air_temperature" ;
              MOS_Temp_instant_2m_00Z_12hr:units = "K" ;
              MOS_Temp_instant_2m_00Z_12hr:vertical_coord = "elev0" ;
              MOS_Temp_instant_2m_00Z_12hr:coordinates = "phenomenonTime latitude longitude" ;
              MOS_Temp_instant_2m_00Z_12hr:ancillary_variables = "elev0 phenomenonTime FcstRefTime lead_times MOS_Method " ;
              MOS_Temp_instant_2m_00Z_12hr:missing_value = 9999. ;
              MOS_Temp_instant_2m_00Z_12hr:PROV__wasInformedBy = "( )" ;
              MOS_Temp_instant_2m_00Z_12hr:PROV__hadPrimarySource = "GFS13" ;
              MOS_Temp_instant_2m_00Z_12hr:leadtime = 12LL ;
              MOS_Temp_instant_2m_00Z_12hr:FcstTime_hour = 0LL ;
              MOS_Temp_instant_2m_00Z_12hr:SOSA__usedProcedure = "( MOS_Method )" ;
      string stations(stations) ;
              string stations:_FillValue = "_" ;
              stations:long_name = "ICAO METAR call letters" ;
              stations:standard_name = "platform_id" ;
              stations:comment = " Only currently archives reports from stations only if the first letter of the ICAO ID is \'K\', \'P\', \'M\', \'C\', or \'T\'. " ;
              stations:coordinates = "latitude longitude" ;
              stations:missing_value = 9999LL ;
      double latitude(stations) ;
              latitude:_FillValue = 9999. ;
              latitude:long_name = "latitude" ;
              latitude:units = "degrees_north" ;
              latitude:valid_min = -90. ;
              latitude:valid_max = 90. ;
              latitude:standard_name = "latitude" ;
              latitude:coordinates = "latitude longitude" ;
              latitude:missing_value = 9999. ;
      double longitude(stations) ;
              longitude:_FillValue = 9999. ;
              longitude:long_name = "longitude" ;
              longitude:units = "degrees_west" ;
              longitude:valid_min = -180. ;
              longitude:valid_max = 180. ;
              longitude:standard_name = "longitude" ;
              longitude:coordinates = "latitude longitude" ;
              longitude:missing_value = 9999. ;
// global attributes:
              :institution = "NOAA/National Weather Service" ;
              :Conventions = "CF-1.7 CAMPS-1.2" ;
              :version = "CAMPS-1.2" ;
              :history = "" ;
              :references = "" ;
              :organization = "NOAA/MDL" ;
              :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~camps/" ;
              :file_id = "0e4181eb-9683-4e17-bcd1-a0096fca9b1f" ;
              :primary_variables = "MOS_Temp_instant_2m_00Z_12hr stations latitude longitude" ;
group: prefix_list {

  // group attributes:
              :PROV__ = "http://www.w3.org/ns/prov/#" ;
              :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
              :SOSA__ = "http://www.w3.org/ns/sosa/" ;
  } // group prefix_list
}
```

Statistical Post-Processing Background

Statistical post-processing (StatPP) refers to the adjustment of current real-time forecast guidance using the discrepancies noted between past forecasts and observations/analyses. Past experience has shown that StatPP is capable of modifying real-time NWP guidance that is biased, somewhat unskillful, and unreliable into guidance that is unbiased, much more skillful, downscaled to local conditions, and highly reliable, thus making it suitable for use in decision support with little or no manual modification by forecasters.” StatPP can also ameliorate deficiencies due to finite ensemble size and infer forecasts for weather elements that are not directly forecast by the NWP system. (Cf. Hamill and Peroutka, 2016: High-Level Functional Requirements for Statistical Post-Processing in NOAA.)

It is a truism among StatPP developers that they spend 10% of their time in science, 10% in statistics, and 80% in bookkeeping. Indeed, metadata storage and use are key aspects to any successful StatPP project. Daunting amounts of data characterize the training phase of many techniques. Some techniques defer these challenges to the production phase.

For many StatPP techniques, the **Training Phase** or **Development Phase** is a set of processes and software that notes discrepancies between past forecasts and observations/analyses and distills them into a set of parameters. The Model Output Statistics (MOS) and Kalman Filter techniques both have distinct training phases. Most bias-correction techniques, however, do not.

All StatPP techniques have a **Production Phase** or **Implementation Phase**, which is the set of processes and software that creates output forecasts.

The **Proxy for Truth** is the set of observations/analyses that guides the StatPP process. The name recognizes the biases and errors that afflict our best observing platforms and analytical techniques. The proxy for truth is generally accepted to be adequate for the task of StatPP.

Numerical Weather Prediction (NWP) generally begins with some form of **Data Assimilation (DA)** which is followed by one or more runs of a NWP system. Additional steps may be required to breed perturbed inputs to facilitate an ensemble of NWP runs. The final step of an NWP run is named the Model Post; this step generally converts output from the specialized coordinate reference systems used in NWP (e.g., spherical harmonics and sigma levels) to more standard coordinate reference systems. StatPP applications generally work with these standard outputs.

Many StatPP applications use some form of **Statistical Pre-processing** step where **NWP output** from multiple runs is captured in a **StatPP Archive**. This Statistical Pre-processing captures the data needed for the Training Phase. Often, NWP output is transformed in ways that facilitate statistical training. In general, if a Statistical Pre-processing step is required in the Training Phase, that same step will also appear in the Production Step.

A Training Phase, if present, will use one or more **Statistical Development Engines** to note discrepancies between past NWP output and a selected Proxy for Truth. These discrepancies are then captured in a set of **StatPP Parameters** which can be used in the Production Phase.

Figure 1, below, attempts to capture some of these concepts in a data flow diagram.

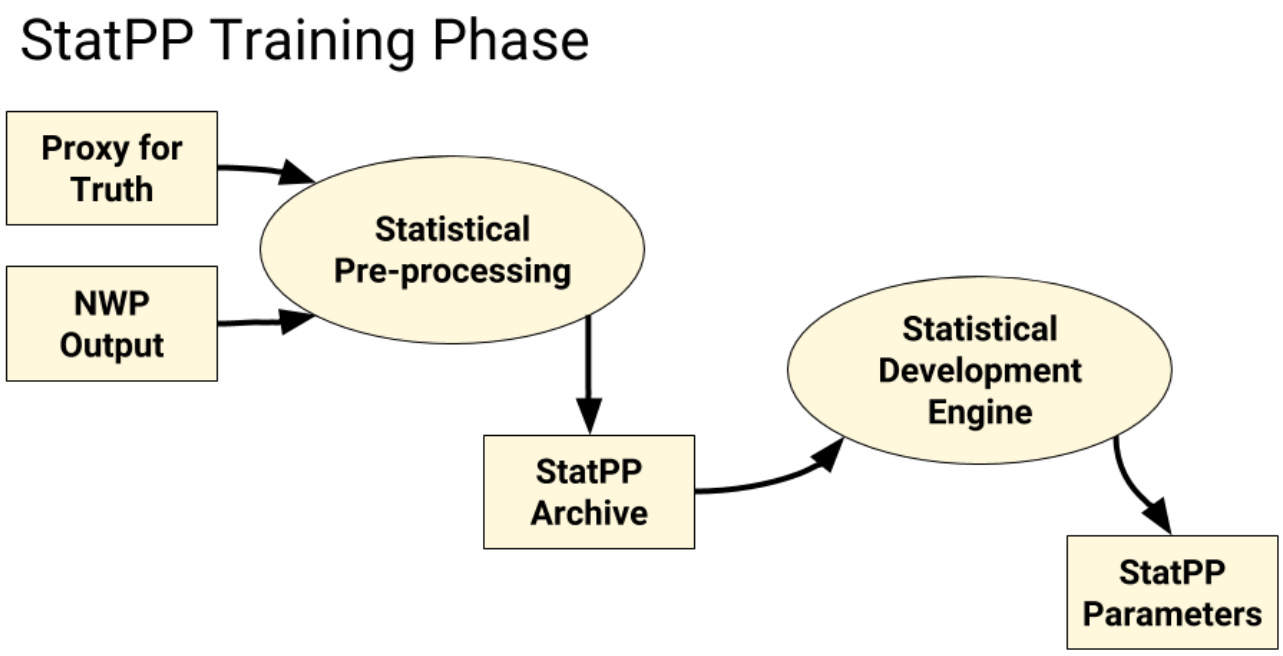
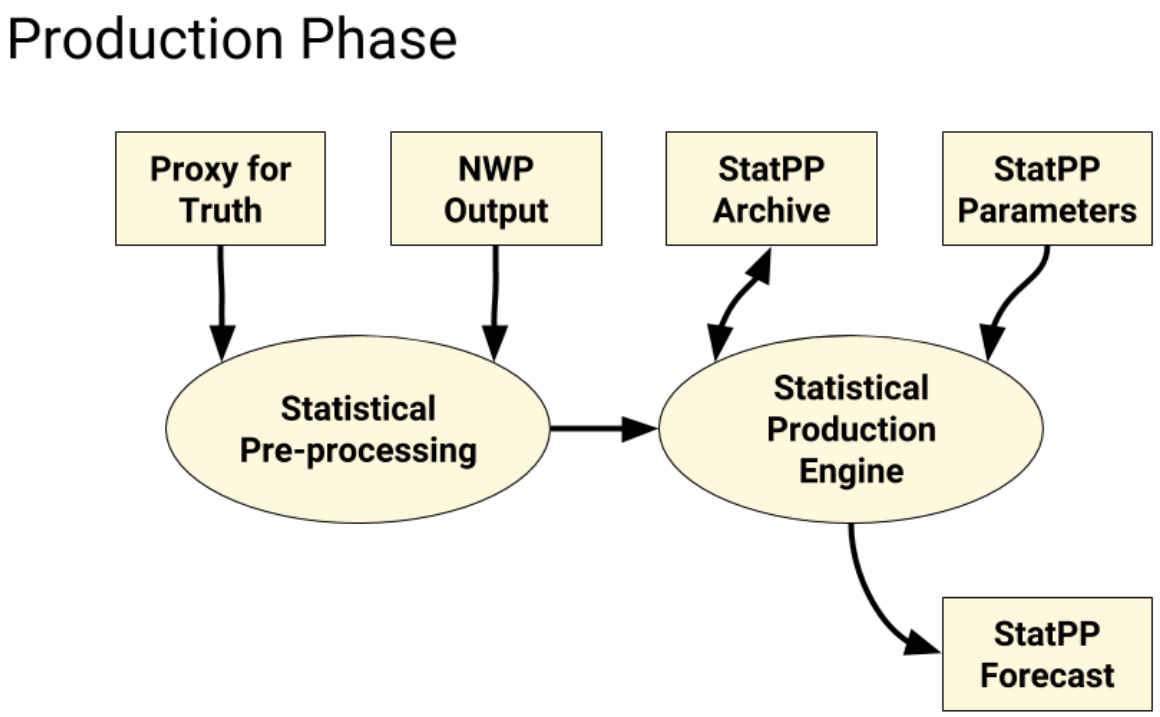


Figure 2, below, also captures these concepts, but applies them to the **Production Phase**.



The NWS Codes Registry: Using linked data to support CAMPS

Introduction to the NWS Codes Registry

The NWS Codes Registry (https://codes.nws.noaa.gov) supports a number of projects, including XML-encoding of aviation products and the Community Atmospheric Model Post-processing System (CAMPS). Most of the techniques described here apply to all of them. The technical documentation for the codes registry API can be found at https://github.com/UKGovLD/registry-core/wiki/Api.

Code Registry Basics

Upon navigating to the NWS Codes Registry website, users are directed to the Registry's main page (Fig. 1). From there, one may search directly for a specific entry or group of entries by typing text into the search box at the top of the page (red numeral "1", Fig. 1), or may browse through the registry by using the URI navigation bar (2) and the links for the various "collections" of entries found in the table of contents at the middle of the page (3). The Registry may be traversed as a nested file system by using the links of the desired collections to move to the desired collection and level of the registry tree.

Conversely, the navigation bar can be used to navigate quickly back *up* the Registry "tree". The various path segments of the URI are separate, clickable links, which will take the user to the Registry contents page at that level. (*Note:* In addition to clicking on the link for the "root" level in the navigation bar, the "Browse" button at the top of the page may also be used to return quickly to the main page of the Registry.)

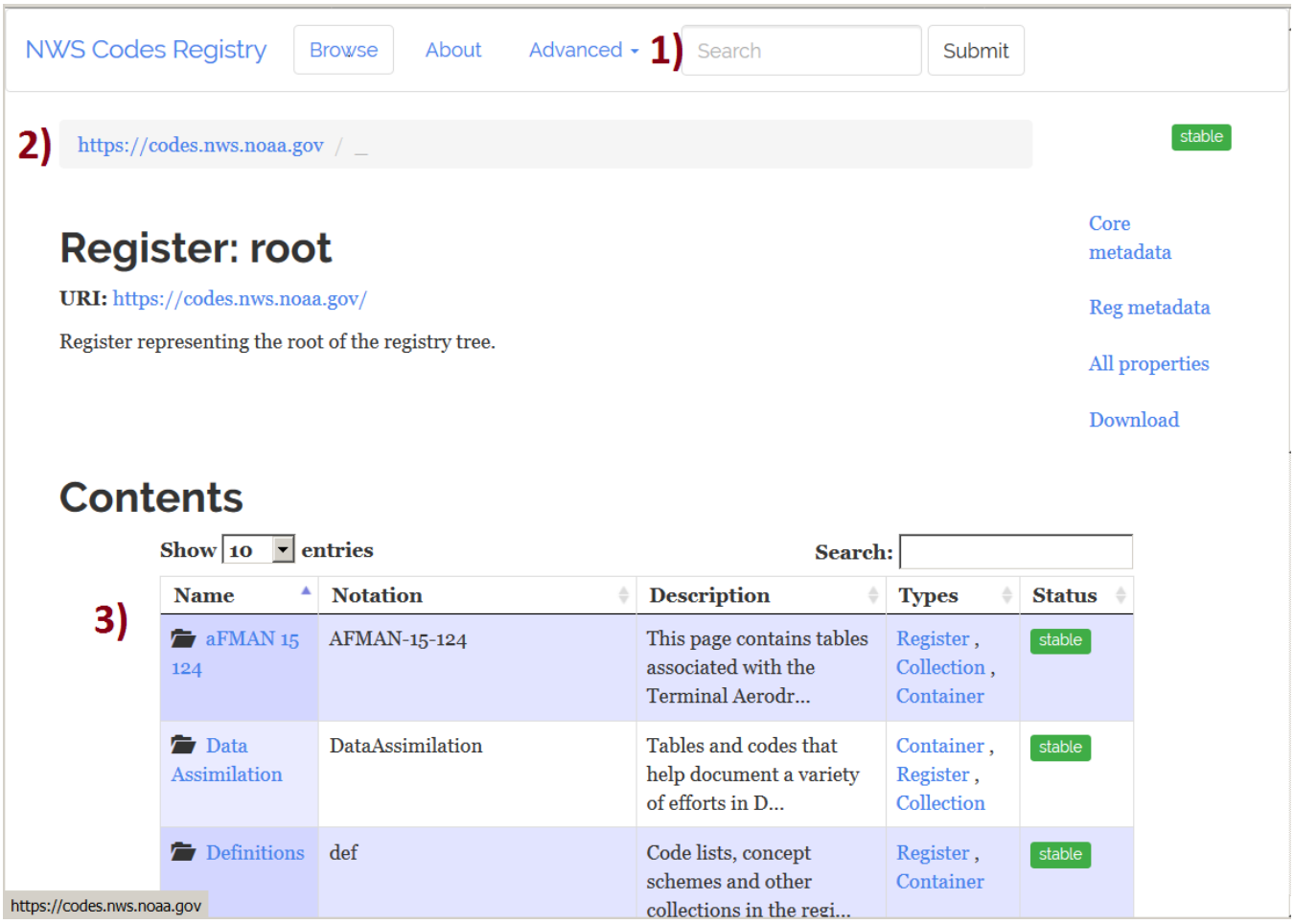


Figure 1: Main or “Home” page of Registry at root level

Once at the desired level of the Registry, the user can obtain metadata for specific “concept” entries by clicking on their links in the table of contents. Most of the entries that will be of interest to StatPP practitioners are found under the collection named “Statistical Post Processing”.

Figure 2 is a screenshot of the GUI display after navigating to the entry for “Potential temperature” **URI:** https://codes.nws.noaa.gov/StatPP/Data/Met/Temp/PotTemp in the “/statPP/Data/Met/Temp” collection of the Registry. Note: 1) the navigation bar with links to all levels above the current Registry level, 2) the entry name and link to the full URI near the center of the page, and 3) a further set of links near the upper right of the page which can be used to view or download metadata and history information for this entry. The “core” metadata for the entry is displayed at the bottom portion of the page under the heading “Definition” (4). The core metadata includes a number of standard properties (“description”, “label”, “notation”, “references”, “type”, etc.) which are used to describe the entry

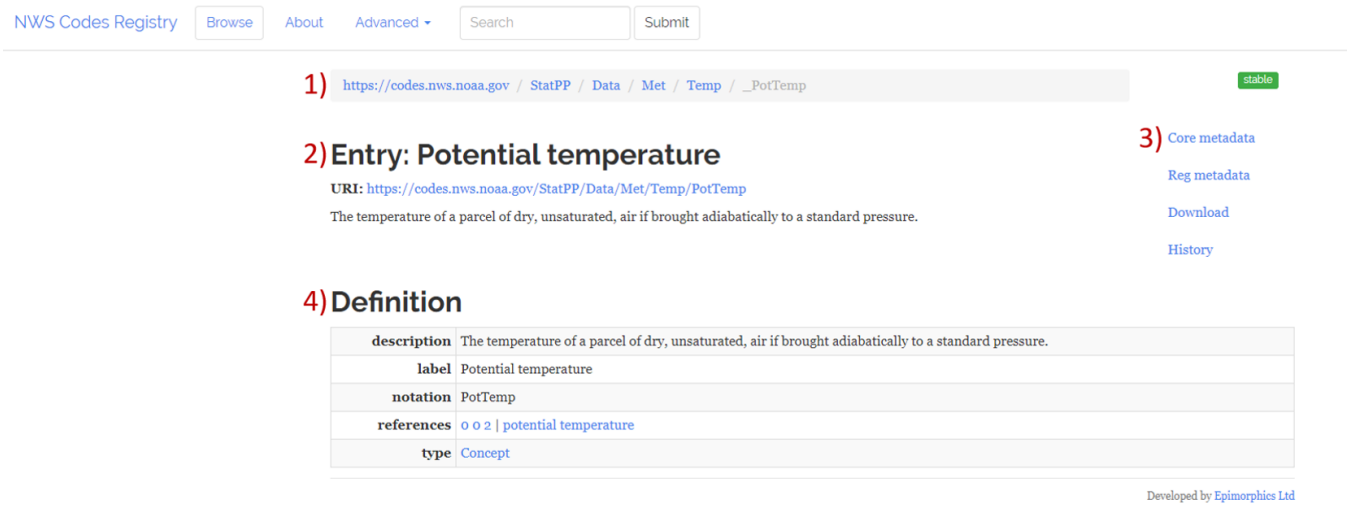


Figure 2: Registry GUI display for “Potential Temperature” under /StatPP/Data/Met/Temp

Note

In this case, the “description” is quite generic, however this field will often contain more extensive and detailed language. Further, the “references” and “type” fields may contain links to other URIs which more completely describe the particular concept or collection. In this case our “type” is a Concept and we have two “references”, one points to the WMO Codes Registry entry for Potential Temperature (0 0 2) and the other the AMS Glossary definition of Potential Temperature.

How to submit new entries to the NWS Codes Registry

Entries for statistical post-processing

Entries for statistical post-processing were drawn originally from concepts, variables, and procedures used in the MOS-2000 statistical post-processing system, augmented with various WMO resources. Requests for new entries should be submited as a .csv (or similar spreadsheet format) file with specific formatting the user should follow which is outlined in Figures 3a, 3b, and 3c.

Submissions of new items and their metadata should be in exactly the same format as the example entries. New entries will be added to the Registry only after their metadata are determined to be in good order and the entries have been approved by the CAMPS Metadata Review Team via an internal review process.

Note

See the “Definitions and References” (Fig. 3c) for further explanation of the metadata properties required for entry into the Registry and their format.

Developers proposing new entries for the Registry should submit their requests via the CAMPS github page (more to come on that soon!) A member of the CAMPS team will initiate the internal review process. The CAMPS team will work to ensure that the requested entries are necessary (i.e. that functionally similar entries do not already exist elsewhere in the Registry which could be used to describe the proposed Concept), and conform to CAMPS metadata standards for software and netCDF datasets.

Once the CAMPS Metadata Review Team is satisfied that the metadata request is well-posed, properly organized for the Registry, and conforms to CAMPS standards, the new entry will be added to the registry.

Collections under StatPP				
@ID	**rdfs:LABEL**	**skos:NOTATION**	**dct:DESCRIPTION**	**dct:REFERENCES**
https://codes.nws.noaa.gov/StatPP/Methods/Arith	Arithmetic	Arith	Arithmetic calculations	
https://codes.nws.noaa.gov/StatPP/Data/Met/Temp	Temperature	Temp	Parameter category of temperature	http://codes.wmo.int/grib2/codeflag/4.1/_0-0
https://codes.nws.noaa.gov/StatPP/Uncertainty/DichProbEvents	Dichotomous Events	DichProbEvents	A collection of dichotomous probabilistic events (i.e., they either happen or they don't)	
Fill in proposed entries below, following the above examples:				
https://codes.nws.noaa.gov/StatPP/				

Figure 3a: Metadata Request Form for Codes Regsitry Collections

Concepts under StatPP				
@ID	**rdfs:LABEL**	**skos:NOTATIO**	**dct:DESCRIPTION**	**dct:REFERENCES**
https://codes.nws.noaa.gov/StatPP/Methods/Geosp/ElevA	Adjustment for elevation	ElevAdj	Adjustment for elevation to a property	http://codes.wmo.int/bufr4/b/25/_066
https://codes.nws.noaa.gov/StatPP/Data/Met/Moment/Abs	Absolute vorticity	AbsVort	Absolute vorticity	http://codes.wmo.int/grib2/codeflag/4.2/_0-2-10
https://codes.nws.noaa.gov/StatPP/Uncertainty/PrcntlRnk	Percentile Rank	PrcntlRnk	A value (often predetermined) that specifies an ordered quantile of a) the cumulative distribution function of a random variable or b) the observations in a sample in units of percent.	
Fill in proposed entries below, following the above examples:				
https://codes.nws.noaa.gov/StatPP/				

Figure 3b: Metadata Request Form for Codes Registry Concepts

Definitions and References		**MDL guidelines and conventions**
id:	A token appended to the end of a URI to create a new URI.	Includes full URI, which generally should be confined to alphanumeric characters only. Users should follow established URI conventions. (See: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier for a quick reference.) **Caution**: We have made limited use of "_" and/or "." where it seems appropriate, but these may not upload properly to Registry without further human intervention.
label:	The human-readable label assigned to the term.	Short-form label, "human readable" for the most part, but short enough to serve as column entry in MDL Codes Registry. Again, for the most part, the "label" should be confined to alphanumeric characters only. Some limited use of special characters (".", "_", "*") seems possible, but these must be protected and may not upload properly to Registry without further human intervention. (For further guidance, see: https://github.com/UKGovLD/registry-core/wiki/Api#csv-format)
notation:	A notation is a string of characters such as "T58.5" or "303.4833" used to uniquely identify a concept within the scope of a given concept scheme. A notation is different from a lexical label in that a notation is not normally recognizable as a word or sequence of words in any natural language. This property is used to assign a notation as a typed literal. (See https://www.w3.org/TR/skos-reference/#notations)	For MDL purposes, this usually matches the final position in URI tree under "id" above, but it doesn't have to in all given concept schemes.
description:	A statement that represents the concept and essential nature of the term.	This should be a human-readable, long form description of the entry. It's acceptable for this to match the label entry, above, for the simplest items. For others, it is acceptable to include a more complete description or summary.
references:	Authoritative documentation related to the term. For our purposes, we will use this tag to indicate our adoption of a concept defined by WMO or some other relevant authority.	Link to site hosted by WMO, MDL, or other pertinent "authority" containing more complete documentation of the concept or method described.

Figure 3c: Definitions and References

Entries for other projects

Code Registry entries for projects other than StatPP should follow the procedures established elsewhere by the Administrators of those particular projects. These will generally not be reviewed by the CAMPS team for adherence to CAMPS StatPP metadata standards, but occasionally may need to be added to the Registry by a CAMPS Administrator or superuser.