CAMPS Documentation

version 1.0.0

CAMPS Team

August 20, 2020

Contents

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

CAMPS is a software infrastructure that supports Statistical Post-Processing and is maintained as community code. This infrastructure includes standards and tools for data representation as well as software repositories that facilitate the use of these standards. This document is intended to describe the goals, standards, and structure for the development and maintenance of CAMPS. It will also serve as a link to a number of other documents that will capture considerably more detail about aspects of CAMPS.

Overall Goals

- Follow NOAA's standards for Statistical Post-processing (StatPP) of output from Numerical Weather Prediction (NWP) systems.
- Maintain all components of CAMPS as community code. This implies the following:
    - CAMPS is a free and shared resource with distributed development and centralized support.
    - Ongoing development of CAMPS is maintained under version control
    - Periodic releases, which include the latest in developments of new capabilities and techniques, are made available to the user community.
    - Insofar as policy permits, major organizations (i.e. NOAA) that participate in the CAMPS community and perform StatPP operationally will maintain the currently operational code in a public repository
- Foster community involvement, experimentation, and improvement.
- Promote accessible, documented, robust, and portable code.

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.

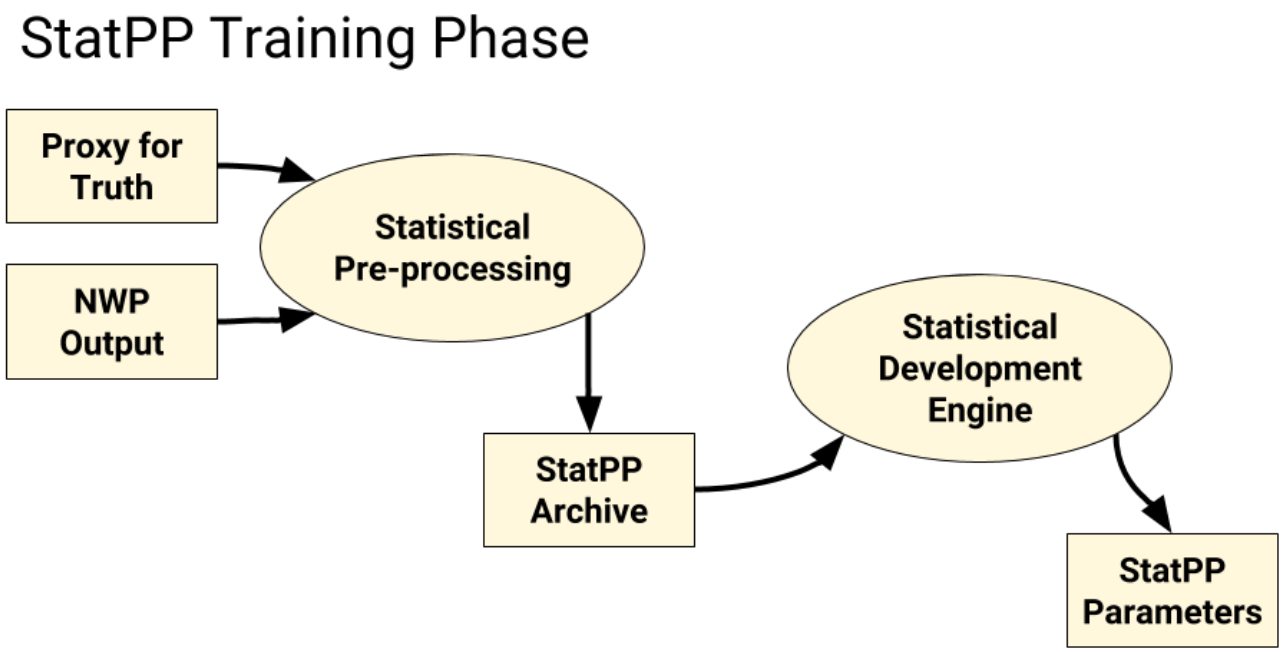
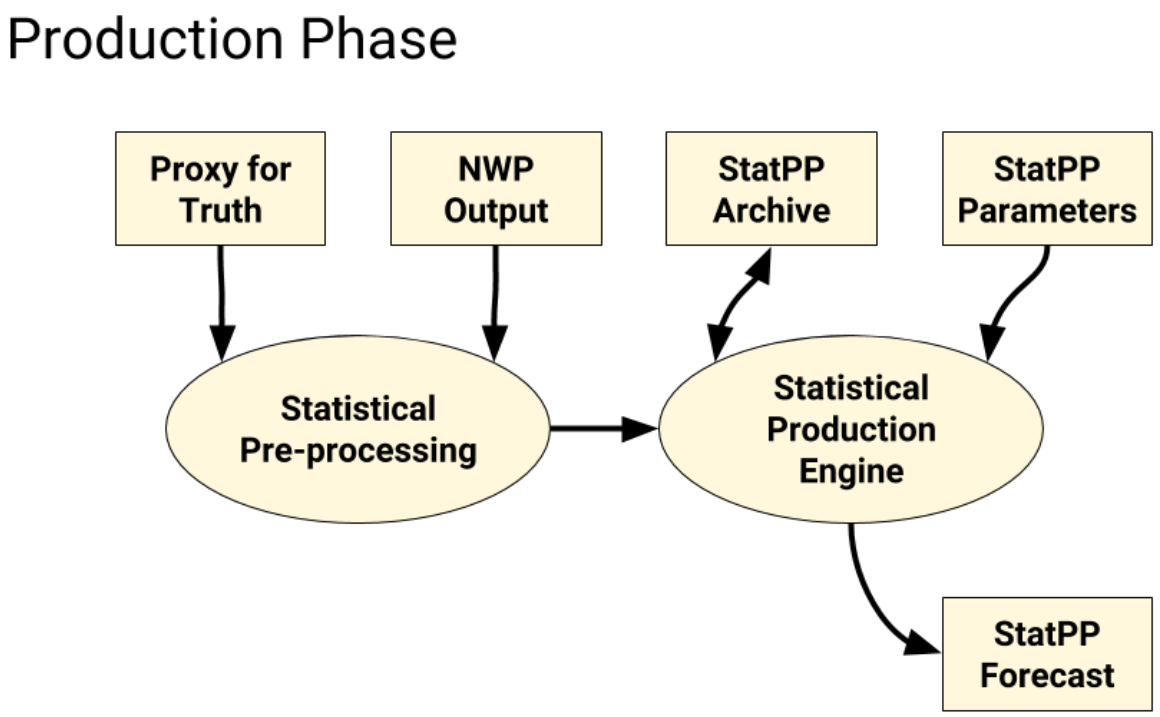


Figure 2, below, also captures these concepts, but applies them to the **Production Phase**.



Towards a new NetCDF-CAMPS Standard

Primary File Format

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). Also, NetCDF seems to have more widespread support within the StatPP community than other self-describing data formats like Hierarchical Data Format (HDF).

Use of the CF-Convention

Within the netCDF family, CAMPS has adopted the NetCDF Climate and Forecast (netCDF-CF) conventions format. NetCDF-CF shares common goals with CAMPS, including the processing and sharing of netCDF files and facilitating data exchange. NetCDF-CF also provides a controlled vocabulary that guides the specification of metadata, as well as other spatial and temporal properties of variables commonly used in StatPP datasets.

First and foremost, a netCDF-CAMPS file must comply with both the netCDF standard and the associated CF conventions (http://cfconventions.org/). **The CF conventions recommend the file extension ".nc" and imposes no other restrictions on file names.** Similarly, netCDF-CAMPS imposes no further restrictions on variable names or dimensions beyond those established for CF.

Similar to the CF convention, most of the requirements for **netCDF-CAMPS** are focused on the use of attributes, ancillary variables, and auxiliary coordinate variables to adequately describe the metadata associated with StatPP variables. We also recommend the use of a global attribute "Conventions" which should include the string "CAMPS-1.0" to indicate compliance with this profile. Since all files that follow the CAMPS application profile also follow the netCDF-CF Conventions, the conventions attribute should also contain the appropriate CF string (Ex: Conventions = "CF-1.7 CAMPS 1.0").

Guidelines for Construction of CF Standard Names <http://cfconventions.org/Data/cf-standard-names/docs/guidelines.html>`_ provides background for how CF Standard Names are constructed and how new prototype standard names should be developed. Included in this description are Transformations which are Standard Names constructed via other Standard Names. There are cases where there are no CF Standard Names or Transformations that adequately describe the given StatPP variable,

method, or process. In CAMPS when development of a new Transformation or Standard Name is necessary, we declare an attribute named **prototype_standard_name** as well as provide the most relevant CF Standard Name to the given variable, method, or process. the. This conveys our intent while maintaining conformity with the current standard.

Use of NetCDF-Classic-LD

NetCDF-CAMPS makes use of Linked Data (LD) for encoding and publishing data wherever possible. We follow the netCDF Classic Linked Data standard (netCDF-LD) (http://docs.opengeospatial.org/DRAFTS/19-002.html – see section 6). 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 Uniform Resource Identifiers (URIs), using prefixes and aliases. 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.

An example of how NetCDF-CAMPS utilizes the prefix and alias aspect of netcdf-LD follows. The user should specify group attributes which specify the necessary prefixes within the file.

Prefix definition and use: (from section 6.3.3 (http://docs.opengeospatial.org/DRAFTS/19-002.html).

“A prefix is declared using a name-value-pair that associates a short name (e.g. cf, bald), with a URI. A single prefix declaration is an attribute and a value: the attribute name is the prefix name and the attribute value is the full URI for that prefix.”

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.

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

```
|  group: prefix_list {
|   // group attributes:
|             :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
|             :OM2__ = "http://codes.nws.noaa.gov/StatPP/" ;
|             :SOSA__ = "http://www.w3.org/ns/sosa/" ;
|             :OM__ = "http://www.w3.org/ns/sosa/" ;
|             :PROV__ = "http://www.w3.org/ns/prov/#" ;
|             :StatppUncertainty__ = "http://codes.nws.noaa.gov/StatPP/Uncertainty" ;
|   } // group prefix_list
|  }
```

These prefixes are utilized to link metadata within a variable to web documentation. The following is CDL output for a NetCDF-CAMPS file that shows how a prefix can be used to link a variables metadata to a NWS codes registry entry.

```
|double Temp_instant_700_21600(default_time_coordinate_size, number_of_stations, level) ;
|         Temp_instant_700_21600:_FillValue = 9999. ;
|         Temp_instant_700_21600:OM__observedProperty = "StatPP__Data/Met/Temp/Temp" ;
|         Temp_instant_700_21600:grid_mapping = "polar_stereographic" ;
|         Temp_instant_700_21600:SOSA__usedProcedure = "( GFSModProcStep1 GFSModProcStep2 LinSmooth BiLinInterp )" ;
|         Temp_instant_700_21600:long_name = "temperature instant at 2m" ;
|         Temp_instant_700_21600:filepath = "/scratch3/NCEPDEV/mdl/Emily.Schlie/inputfiles/greg_test/gfs0020160700.nc" ;
|         Temp_instant_700_21600:ancillary_variables = "OM__phenomenonTimeInstant OM__resultTime ValidTime FcstRefTime LeadTime GFSModProcStep1 GFSModProcStep2 LinSmooth BiLinInterp " ;
|         Temp_instant_700_21600:coordinates = "plev0 station" ;
|         Temp_instant_700_21600:FcstTime_hour = 21600LL ;
|         Temp_instant_700_21600:standard_name = "air_temperature" ;
|         Temp_instant_700_21600:units = "K" ;
```

In the CDL output above OM__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 can also be applied to the other part of the bolded CDL output above “OM__observedProperty”. By replacing OM__ with it’s prefix definition, the user will be directed to http://www.w3.org/ns/sosa/observedProperty, which is the documentation describing O&M observedProperty.

The user is encouraged to read through the entirety of section 6 within the “OGC Encoding Linked Data Graphs in NetCDF Classic Files” documentation and apply these techniques whenever possible when creating and processing Netcdf-CAMPS data files.

CAMPS Metadata Standards

A number of attributes that will be used in netCDF-CAMPS have their origins in vocabularies that are external to netCDF and the CF conventions. One of the key motivations in establishing netCDF-CAMPS has been to develop a well-defined template for incorporating these very rich external vocabularies into files that conform to netCDF-CF. Since netCDF-CAMPS is intended to support StatPP specifically, we focus on vocabularies that have been useful for this work in the past. Sections 3.2.0 through 3.2.3 outline the vocabularies utilized in CAMPS and our justification for including them.

As a starting point, CAMPS has adopted the use of the International Organization of Standards (ISO) standard for Observation and Measurements (O&M), ISO 19156. This introduces a data model and a controlled vocabulary that is useful for the description of both observations and forecasts of a given StatPP variable within NetCDF-CF. Many elements can be easily recognized and the terminology is designed to apply across disciplines.

CAMPS will also make use of other international standards, which include additional data models and vocabularies that are web-accessible. The inclusion of these standards will help simplify atmospheric science applications and promote more widespread acceptance of CAMPS as “community” code. These standards include a data model for the ontology of provenance information (PROV-O; [PROV]) and the Semantic Sensor Network Ontology [SSN] developed by the World Wide Web Consortium (W3C). SSN also includes a simple core ontology named SOSA (Sensor, Observation, Sample, and Actuator). Many of the key concepts found in O&M have been integrated into SSN in formats that are hosted on the web (https://www.w3.org/2015/spatial/wiki/Alignment_to_O%26M). In many cases, the W3C vocabularies express these key concepts in a simpler, more compact format. In addition, the W3C models also seem a bit more stable and robust than O&M.

ISO standards

The International Organization of Standards (ISO) is the world’s largest developer of international standards, they facilitate world trade by providing common standards between nations. Today, some 20,000 standards have been set by ISO. Of particular importance, the WMO, OGC, and ICAO have all adopted the use of ISO standards for Observations and Measurements (ISO 19156) and Metadata (TC211) for key applications. These standards are very relevant for weather data, so it seems appropriate for us to illustrate the use of these concepts in netCDF attribute names.

OM_Observation:

It may appear counterintuitive to apply the O&M standard when encoding data that are forecasts. Despite its name, the O&M class (OM_Observation) is remarkably well suited to the representation of forecasts as well as observations. Meteorological examples of forecasts include NWP and StatPP output and Terminal Aerodrome Forecasts (TAFs). Meteorological examples of observations include analyses, METARs, and Pilot Reports

The **OM_Observation** class provides a framework to describe the event of an observation or measurement. In O&M, an observation is defined as: *“An event that results in an estimation of the value (the result) of a property (the observed property) of some entity (the feature of interest) using a specified procedure”.*

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 NWP 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.

Several model elements used in O&M are defined in other ISO geographic information standards. With the exception of basic data type classes, the names of Unified Modeling Language (UML) classes (within ISO/TC 211) include a two or three letter prefix that identifies the relevant international standard and the UML package in which the class is defined. (Ex: “OM” would be the international standard and “ObservedProperty” would be the class. Together this is OM_ObservedProperty). UML can be thought of as thesuccessor to object-oriented (OO) analysis and design. An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and also forms a hierarchy to model the real-world system.

W3C standards

The World Wide Web Consortium (W3C) is an international community of members that work together to develop web standards. The W3C has developed two standards that are useful for StatPP applications. They are the **ontology for provenance information (PROV-O; [PROV])** and the **Semantic Sensor Network Ontology [SSN]**. SSN includes a core ontology named **SOSA (Sensor, Observation, Sample, and Actuator)**. Many of the key concepts found in O&M have been integrated into SSN in formats that are hosted on the web.

Many of the attributes used in netcdf-CAMPS are rooted in PROV-O and SSN/SOSA. There are a number of reasons why netCDF-CAMPS makes use of both SSN/SOSA and PROV-O, they are:

- Helps promote a more widespread acceptance of our “community” code by incorporating other data models already in use outside the U.S. and which are hosted on the Web (supported by XML).
- The W3C vocabularies incorporate many key concepts of OM in a simpler, more compact format.
    - This simplicity is especially useful when describing dataset provenance and lineage; OM constructs (LI,LE classes) are very cumbersome when trying to describe multi-step processes.
    - Furthermore, some essential properties that are necessary to accurately describe meteorological and StatPP variables are not available within the O&M vocabulary, specifically with regards to time variables.
- The W3C models seem a bit more stable and robust than O&M.

Integration of user-defined concepts

Data producers who use NetCDF-CAMPS should give preference to concepts published by the WMO and other Standards Development Organizations wherever possible. However, in many cases, data producers will need to identify concepts that fall exclusively within their provenance. Examples include data processing techniques that are not widely used, weather element definitions that are not widely accepted, etc. For these cases, the data producer should define and publish, in some manner, a number of URIs that identify and document these concepts. The preferred solution would be for the data producer to make this information web-accessible and machine readable. The WMO Code Registry (http://codes.wmo.int) and the NWS Code Registry (https://codes.nws.noaa.gov) both provide these capabilities.

This document introduces an application profile for CAMPS utilizing the netCDF-CF conventions, and other data models, and controlled vocabularies that improve interoperability within the StatPP community. By following this application profile, we ensure that CAMPS compliant 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.

NetCDF-CAMPS tries to standardize the nomenclature and its representation in metadata by using pre-existing standards from the International Organization of Standards (ISO), World Wide Web (W3), Open Geospatial Consortium (OGC), and the World Meteorological Organization (WMO). The reader is encouraged to download and review **Guidelines on Data Modelling for WMO Codes** (available in English only from http://wis.wmo.int/metce-uml), especially section 1.6 and chapters 4 and 5. These foundational concepts will be applied to the development of metadata within CAMPS data files. Chapter 5 describes the WMO Codes Registry. We note that the NWS has created a Codes Registry at https://codes.nws.noaa.gov, and the NWS plans to use that registry to support local applications within CAMPS. Further, the WMO hosts a similar Codes Registry at https://codes.wmo.int. Both registries are considered authoritative.

Coincidentally, the OGC’s netCDF Standards Working Group (SWG) has been developing a NetCDF Classic Linked Data encoding standard (NetCDF-LD) (http://docs.opengeospatial.org/DRAFTS/19-002.html), which combines netCDF with linked data, the binary-array-ld (bald; https://github.com/binary-array-ld). Where possible, CAMPS 1.0 complies with this emerging standard.

NetCDF-CAMPS Structure

Attribute Names

In netCDF-CAMPS, attributes based on the ISO and other concepts will have names that concatenate the prefix, two underscore characters, and the model element name. The double underscore allows CAMPS to integrate the features of netCDF-Classic-LD. For example, the attribute based on the model element phenomenonTime (in the Observations and Measurements package) is named OM__phenomenonTime. Also, the attribute based on the ontology element prov:Activity (in the PROV-O ontology) is named PROV__Activity.

In at least one case, the SSN/SOSA vocabulary deviates slightly from OM (i.e., usedProcedure vs. procedure). The CAMPS encodings described here will use the OM__ prefix to designate concepts that had their origin in OM. In most cases, the associated attribute name or value will resolve to a concept hosted within SSN or SOSA. A different prefix (OM2__ or SSN__ or SOSA__) will be used when a single CAMPS-encoded file must reference two or more of these resources.

Primary and Ancillary Variables

NetCDF files often contain a large quantity of variables. These variables sometimes depend upon other variables, often referred to as **ancillary variables**. Because of this, NetCDF-CAMPS requires a global attribute called **primary_variables** to be declared. By declaring primary variables we make it easy to differentiate between the main variables from a large data file vs the ancillary variables that are mostly meant to describe aspects of the primary variables.

Primary_variables is a list, separated by white space, of variable identifiers. These primary variables make accessing data of interest more streamlined, especially when a file contains a large number of ancillary variables.

Example:

```
|  //  global attributes:
|
|     :primary_variables = "mos_temperature mos_pop06";
```

Section 4 describes a number of requirements and best practices that involve ancillary variables and auxiliary coordinate variables. Most netCDF-CAMPS files will contain many ancillary variables that are used to fully describe the complexities of time, properties, and procedures used. This makes the use of the primary_variables global attribute especially useful.

Observed Property and Used Procedure

NetCDF-CAMPS requires the two attributes **standard_name** and **long_name**, which are introduced in CF-conventions, to standardize the concepts of what is observed/forecast and how it was processed.

Standard_name is a controlled vocabulary (http://cfconventions.org/Data/cf-standard-names/65/build/cf-standard-name-table.html).

Long_name is intended to be human-readable.

NetCDF-CAMPS also introduces two additional concepts; **OM__observedProperty** and **SOSA__usedProcedure**. Our intent is to facilitate data discovery, provide explicit linkages to widely-accepted standards outside the atmospheric/oceanic/space weather domains, and use widely-accepted standards to enhance our metadata. StatPP applications frequently apply complex procedures to datasets, and this increases the importance of these metadata entries.

Recall that in O&M an observation is an event that results in an estimation of the value (the result) of a property (*OM__observedProperty*) of some entity (the feature of interest) using a specified procedure (*SOSA__usedProcedure*).

In simple cases, the OM__observedProperty and SOSA__usedProcedure may seem to be redundant with standard_name and long_name. That said, StatPP applications often define parameters (e.g., sine of the day of the year) that are useful in the development phase, but will not be disseminated. In these instances, careful use of OM__observedProperty and SOSA__usedProcedure can yield a number of benefits.

- It can thoroughly document the origins of a variable.
- It can enhance data exchange among StatPP development organizations.
- It can improve the quality and utility of data archives.

OM_observedProperty

All primary variables in a netCDF-CAMPS dataset should have an attribute named **OM__observedProperty** declared with a value that is a URI. This URI will refer to a clear description of the property contained in the variable. The URI need not describe any procedures performed to obtain the results contained in the variable. The procedure description is better contained in OM__procedure (SOSA__usedProcedure). Ideally, the URI will be web-accessible.

SOSA_usedProcedure

All primary variables in a netCDF-CAMPS dataset should have an attribute **SOSA__usedProcedure**, declared with a value of the form “(v1 v2 v3 … vn)” where n is a non-zero integer.

The order of tokens should be in the same order that the procedures were performed.

Example:

- For a single step: :SOSA__usedProcedure = “(firststep)”;
- For two steps: :SOSA__usedProcedure = “(firststep secondstep)”;
- For three steps: :SOSA__usedProcedure = “(firststep secondstep thirdstep)”;
- etc…

The tokens v1, etc. will name ancillary variables which are written to the netCDF-CAMPS file as separate variables whose attributes describe the procedure. The names of these attributes should be taken from conformance class schemas found in PROV. When used, the attribute PROV__Activity should always appear with a value that is a URI. This URI will refer to a clear description of the processing step. Other attributes found in PROV are permitted (e.g., PROV__Used). PROV__Used describes little more than the ingest of the data. The first token in SOSA_usedProcdure should usually have a PROV__Used token.

SOSA__usedProcedure step variables are typed short integer, have descriptive, human-readable long names, and have units of unity. Unil a more appropriate standard_name becomes available “source” should be used”.

Data producers should include sufficient detail in the SOSA__usedProcedure steps to adequately describe the data contained in the variable and distinguish it from other, similar variables and otherwise identical variables contained in other datasets. Judicious use of these attributes can add key metadata to a netCDF-CAMPS database and facilitate collaboration. SOSA_usedProcedure should only document what a user has done to the data. If highly processed data is received from outside the users system OM_observedProperty in the ancillary variable metadata that points to a code registry entry that describes the external processing.

SOSA__usedProcedure should not be used to convey a list of routines that were called. If needed, those details can be captured in the various entries in the codes registry.

Geospatial Coordinates

The guidelines set forth in the NetCDF-CF conventions for geospatial coordinates in the horizontal and vertical axes seem to thoroughly cover all use cases envisioned for CAMPS. No extensions are planned.

Time Coordinates

Time nomenclature seems to be a perennial stumbling block in StatPP. This often becomes evident as data consumers attempt to process data that has been shared by one or more data producers..The following table provides examples of time-related terms used in netCDF-CAMPS and their definitions. Historically, these terms have not been defined with sufficient precision or used consistently in meteorology. Our aim is to address this issue early in the lifespan of CAMPS.

The term	Means this…	Not this
forecast_reference_time	The “data time”, the time of the analysis from which the forecast was made (from the CF Standard Name Table). Must be an instant in time.	The time for which the forecast is valid.
OM__phenomenonTime	Colloquially, “when the weather happens”. Can be either an instant in time or a period of time.	
OM__resultTime	When the result (analysis, forecast) became available to data consumers. Must be an instant in time.	
OM__validTime	Time of intended use. Must be a period of time.	The time for which the forecast is valid.
leadTime	Length of time (a duration) from forecast_reference_time to OM_phenomenonTime. Must be a duration.	An instant in time.

The netCDF-CF Conventions provide considerable guidance on the encoding of time coordinates. This section expands on the terms outlined in the above table and uses netCDF-CF Conventions as supplemental guidance while defining best practices for netCDF-CAMPS.

TM__Instant

“An instant is a zero-dimensional geometric primitive that represents position in time. It is equivalent to a point in space. In practice, an instant is an interval whose duration is less than the resolution of the time scale” [TS].

In NetCDF-CAMPS, a number of weather elements are usually represented as occurring at a point in time. These include temperature, wind direction, wind speed, sky cover, and precipitation type. For some of these elements, defining them as an instant is a matter of convenience. In reality, the systems that assess conditions are conducting time averages. Moreover, analyses for multiple weather elements are generally associated with a single instant. In reality, of course, the data that modern analyses process come from a variety of time scales. In the same way, various weather elements in a METAR-encoded observation are associated with a single instant.

TM__Period and TM__Duration

“The period is a one-dimensional geometric primitive that represents extent in time. The period is equivalent to a curve in space. Like a curve, it is an open interval bounded by beginning and end points (instants), and has a length (duration). Its location in time is described by the temporal positions of the instants at which it begins and ends; its duration equals the temporal distance between those two temporal positions” (ISO 19108).

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.

Here are some examples of periods of time: from 00:00 UTC 24 Feb 2017 to 12:00 UTC 24 Feb 2017 the 12 hours ending at 12:00 UTC 24 Feb 2017

…and some examples of durations of time:

- 30 seconds
- 1 hour
- 2 days

Periods and durations that are long enough (generally > one day) often encounter encoding issues because of the vagaries associated with various calendar systems (e.g., leap years, non-Gregorian calendars) The existing netCDF-CF Conventions seem to address these issues thoroughly. No extensions are planned.

Encoding TM__Period

Variables that are inherently of type TM__Period should be encoded that way, using the methods outlined here. We discourage the practice of encoding such fields as TM__Instant and providing human-readable labels that specify duration.

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.

Declared variables defining objects of type TM_Period should include a dimension (the one that varies fastest) of 2. That dimension can be interpreted as any two of the following three concepts: TM_Duration, TM_Beginning, and/or TM_Ending. This variable will also contain sufficient attributes to remove any ambiguity about the interpretation of that dimension of 2. Ideally, this disambiguation will use URIs and be machine-readable.

Example:

```
|  OM__phenomenonTimePeriod:PROV__specializationOf =
|  "StatPP__concepts/TimeBoundsSyntax/BeginEnd", "OM__phenomenonTime";
```

Note

The PROV-O concept specializationOf can take on multiple values. In the example above, the variable OM__phenomenonTimePeriod is a specialization of the O&M concept phenomenonTime and it is also a specialization of the concept of the time bounds syntax named “BeginEnd”. This is interpreted as dimension 0 contains the TM_Beginning and dimension 1 contains the TM_Ending.

forecast_reference_time

The forecast_reference_time is a standard name in the netCDF-CF Conventions. In that document it is defined as “The ‘data time’, the time of the analysis from which the forecast was made” [CF]. This description seems to draw its origin from NWP. It has broad applications within StatPP as well. Clearly, 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 NWP output.

> **Best Practice:** *In datasets where this concept is meaningful, a variable should contain this time (these times) and the attribute standard_name assigned the value “forecast_reference_time.” That said, netCDF-CAMPS does not discourage other practices that convey this concept.*

leadTime

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. We are aware of no formal definition of this concept by any Standards Development Organization (SDO; Various WMO standards provide instructions for encoding this concept). There are a number of expressions that are commonly used to describe this concept (e.g., forecast period, forecast lead, time projection). The CF Standard Names includes an entry for forecast_period that corresponds well to this concept. For netCDF-CAMPS, we recommend that data producers adopt the term “lead time” and use it uniformly.

> **Best Practice:** *In datasets where this concept is meaningful, a variable of appropriate dimensionality should be defined and contain leadTime values. This variable should be declared as an auxiliary coordinate variable, as needed. This variable should have the attribute PROV__specializationOf declared with a value that expresses the concept of leadTime. Ideally this value will use a URI and be machine-readable. This variable should also have a standard_name of “forecast_period.”*

periodicTime and cadence

The “duration of one cycle” (ISO 19108) is defined with the words “periodic time.” While the term is defined in ISO 19108, it does not appear formally in the data models. (Hence, we call it periodicTime, not TM__PeriodicTime.) The concept of periodic time has two primary applications in StatPP.

First, periodic time can describe the duration between successive runs of an NWP system. We suggest the term **cadence** to describe this characteristic.

Second, periodic time can be used to characterize certain datasets whose data elements have a regular spacing in time. We suggest the term **periodicTime** to describe the duration between two successive leadTimes in datasets of this sort.

> **Best Practice:** *In datasets where this concept is meaningful, the attribute structure should be added to the applicable time coordinates. The value of the attribute should be a character string of the form “firstLeadTime=value periodicTime=value lastLeadTime=value.” The values in this string are not intended for automated interpretation. Rather, they are intended to convey information from data producers to data consumers. ISO 8601 and 19108 define an encoding scheme for TM__Duration that is readily readable. It is frequently used in web applications, and suits this purpose quite well.*
>
> *E.g., “firstLeadTime=P6H periodicTime=P3H lastLeadTime=P24H” describes leadTimes of 6, 9, 12, 15, 18, 21, and 24 hours.*
>
> *E.g., “firstLeadTime=P12H periodicTime=P24H lastLeadTime=P84H” describes leadTimes of 12, 36, 60, and 84 hours.*

OM__phenomenonTime

“The attribute phenomenonTime shall describe the time that the result (6.2.2.9) applies to the property of the feature-of-interest (6.2.2.7). This is often the time of interaction by a sampling procedure (8.1.3) or observation procedure (6.2.2.10) with a real-world feature. NOTE 1: The phenomenonTime is the temporal parameter normally used in geospatial analysis of the result.” [O&M}

While not obvious from this definition, OM__phenomenonTime can be of type TM__Instant or TM__Period. This makes OM__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).

> **Best Practice:** *OM__phenomenonTime should always be specified for elements that are observed, analyzed, or forecast. Variables of appropriate dimensionality should be defined and contain OM__phenomenonTime values.*
>
> **Best Practice:** *This variable should be declared as an auxiliary coordinate variable, as needed. We recognize that in many applications a combination of model_reference_time and leadTime can convey the same content as OM__phenomenonTime. The Best Practice, however, is to create an auxiliary coordinate variable explicitly for this purpose and assign it the appropriate attribute. The purpose, of course, is to limit implicit metadata wherever possible.*
>
> **Best Practice:** *This variable should have the attribute PROV__specializationOf declared with a value of the form “(v1 v2 v3 … vn)” where n is a non-zero positive integer. (The multi-valued syntax allows these variables to be specializations of more than one concept.) One of the tokens v1, etc. will express the concept of phenomenonTime. Ideally this value will use a URI and be machine-readable.

Note

Please recall the Best Practice described for all variables that contain the concept TM_Period.

OM__resultTime

“The attribute resultTime:TM__Instant shall describe the time when the result became available, typically when the procedure (6.2.2.10) associated with the observation was completed For some observations this is identical to the phenomenonTime. However, there are important cases where they differ. …

EXAMPLE 3 Where sensor observation results are post-processed, the resultTime is the post-processing time, while the phenomenonTime is the time of initial interaction with the world.

EXAMPLE 4 Simulations may be used to estimate the values for phenomena in the future or past. The phenomenonTime is the time that the result applies to, while the resultTime is the time that the simulation was executed.” [O&M]

OM__resultTime has clear applications to StatPP in general and operational meteorology as well. This concept provides a clear, standards-based way to label a product with it’s time of production.

> **Best Practice:** *This concept is always meaningful for elements that are observed, analyzed, or forecast. [O&M] requires it for all observations. For these elements a variable of appropriate dimensionality should be defined and contain OM__resultTime values.*
>
> **Best Practice:** *This variable should be declared as an ancillary variable, as needed.*
>
> **Best Practice:** *This variable should have the attribute PROV__specializationOf declared with a value of the form “(v1 v2 v3 … vn)” where n is a non-zero positive integer. (The multi-valued syntax allows these variables to be specializations of more than one concept.) One of the tokens v1, etc. will express the concept of resultTime. Ideally this value will use a URI and be machine-readable.*

In general, for observations, OM__resultTime = OM__phenomenonTime; for analyses, OM__resultTime ≥ OM__phenomenonTime; and for forecasts, OM__resultTime ≤ OM__phenomenonTime.

OM__validTime

The term “valid time is used widely in NWP and StatPP however, the O&M definition of the term differs significantly from it’s common usage definition.

O&M definition: “If present, the attribute validTime:TM__Period shall describe the time period during which the result is intended to be used.

NOTE This attribute is commonly required in forecasting applications.” [O&M]

While this attribute is commonly required in forecasting applications [O&M], OM__validTime is not meaningful in all contexts. E.g., a temperature observation taken at a TM__Instant can be fruitfully used indefinitely. In operational meteorology, however, a temperature forecast will probably be replaced by a new (and presumably better) forecast in a matter of hours. Thus, OM__validTime can provide a standards-based way for data producers to inform data consumers of their intentions for the use of their products.

> **Best Practice:** *In datasets where this concept is meaningful, a variable of appropriate dimensionality should be defined and contain OM__validTime values.*
>
> **Best Practice:** *This variable should be declared as an ancillary variable, as needed. In applications where confusion is possible, data producers should take care to declare this ancillary variable with a name that will be meaningful to data consumers (e.g. useful_time, time_of_intended_use, validity_time).*
>
> **Best Practice:** *This variable should have the attribute PROV__specializationOf declared with a value of the form “(v1 v2 v3 … vn)” where n is a non-zero positive integer. (The multi-valued syntax allows these variables to be specializations of more than one concept.) One of the tokens v1, etc. will express the concept of validTime. Ideally this value will use a URI and be machine-readable.*

Note

Please recall the Best Practice described above for all variables that contain the concept TM_Period.

Equation output files

StatPP techniques that include a distinct Training Phase often yield a set of equations (StatPP Parameters) that are used to carry information into the Production Phase. (See section Statistical Post-Processing Background of the Overview for a more complete explanation of these terms.) Of course, these equations must be encoded in one or more equation output files.

The Observations and Measurements (O&M) model is again used as a guideline when encoding data files. However, we focus more on the W3C Provenance Ontology (PROV-O) as a model for encoding StatPP equation output. The various values stored in the equation output file are *entities* (equation coefficients and other ancillary data) that were generated by some *activity* (e.g. the Development Phase), and that activity was performed by some *agent* (the responsible organization).

In most cases a StatPP Equation File will contain one or more weather elements with their associated metadata, that were used as inputs into the regression, that create a set of outputs. These outputs include; an array of equation coefficient (including the Equation Constant), statistical values (Reduction_of_Variance, Average_Predictand, etc…), and any associated ancillary and auxiliary coordinate variables.

> **Best Practice:** *Files that potentially contain equation output should contain global attributes that describe the origin of the file itself. Some useful attributes include PROV__Entity, PROV__wasGeneratedBy, PROV__wasAtributedTo, and PROV__generatedAtTime.*
>
> **Best Practice:** *Equation output variables that are used in StatPP and the names of their dimensions should have names that are drawn from the nomenclature of the technique.*
>
> **Best Practice:** *All equation output variables should include the attribute PROV__Entity or something similar that indicates the nature of the variable and its use. Ideally, both the attribute name and value will be web-referencible URIs.*

Global attributes

We begin with a set of global attributes that describe the file itself.

```
|   :StatPPTime__FcstCyc = "06" ;
|   :StatPPTime__SeasDayFmt = "StatPP__Data/Time/SeasDayFmt/MMDD" ;
|   :PROV__wasAttributedTo = "StatPP__Data/Source/MDL" ;
```

```
|   :institution = "NOAA/National Weather Service" ;
|   :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~wisps/" ;
|   :season = "warm" ;
|   :StatPPTime__SeasEndDay = "0706" ;
|   :PROV__wasGeneratedBy = "StatPP__Methods/Stat/MOSDev" ;
|   :StatPPSystem__Status = "StatPP__Methods/System/Status/Dev" ;
|   :StatPPTime__SeasBeginDay = "0702" ;
|   :PROV__Entity = "StatPP__Data/Source/MOSEqnFile" ;
|   :version = "WISPS-1.0" ;
|   :file_id = "7426a556-9b99-468e-b73e-3825b763ce80" ;
|   :Conventions = "CF-1.7 WISPS-1.0" ;
|   :primary_variables = "MOS_Equations Standard_Error_Estimate Reduction_of_Variance Multiple_Correlation_Coefficient Predictand_Average" ;
|   :StatPPTime__FcstCycFmt = "StatPP__Data/Time/FcstCycFmt/HH" ;
```

The W3C standard for provenance (PROV-O) provides key entries to describe the file and its history. Entries declare what sort of file this is, how it was generated, the file version, the primary source of its contents, who created it, and when. Many of these entries are entries in a codes registry where they can be more fully described by the data producer.

Included in the list of global attributes are three attributes support a common StatPP practice called “seasonal stratification.” Since a number of NWP biases change seasonally as weather patterns (and their simulation) shift, many StatPP techniques repeat their development phase multiple times with seasonal datasets. Thus, StatPPTime__SeasBeginDay and StatPPTime__SeasEndDay convey the first and last dates (in any year) these equation outputs should be used. StatPPTime__SeasDayFmt conveys the encoding format for these dates. Similarly, many StatPP techniques stratify their development phase by forecast cycle. The next two attributes, StatPPTime__ForecastCycle and StatPPTime__FcstCycFmt, convey this concept. Additionally, the attribute “season” should be included int he list of global attributes, identifying the appropriate season for which the equation output is valid.

Finally, the attributes StatPPSystem__Status and StatPPSystem__Role reflect the practices of the author’s institution. They are presented here as a pattern that other data producers may choose to emulate. StatPPSystem__Status tracks the maturity of the equations as they go through their development life cycle. It can convey values such as *developmental*, *prototype*, *operational*, and *experimental*. StatPPSystem__Role is also patterned on a specific institution and technique. StatPPSystem__Role can take on the values *primary* and *backup*, designating primary and backup MOS equations. (Primary equations are developed in ways that permit the most recent observation to serve as a predictor; backup equations are developed with that predictor withheld.)

Equation output variables

The StatPP coefficients and Equation_Constant, for one or more MOS equations, should be saved as a single variable in the equation output file, dimensioned by the number of stations, the number of coefficients (including the Equation_Constant) and the number of Predictands. The StatPP statistics, for one or more MOS equations, should each be stored as individual variables. The first example, below, contains the coefficients (including the Equation_Constant) of a set of MOS equations. Note that the name of this variable and its dimensionality are not part of the CAMPS standard. Rather, attributes are used to convey the information needed to successfully interpret the variable’s contents.

```
|    float MOS_Equations(number_of_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 ;
```

The first, and perhaps most important, attribute we will discuss is PROV__Entity. In effect, PROV__Entity helps the data producer convey to the data consumer “what it is.” In this case, it is the coefficients of one or more MOS Equations. Ideally, the technical information found at StatPP__Data/Source/MOSEq will be sufficient to allow a data consumer to read these data and use them effectively.

Additionally, the ancillary_variables attribute for a 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 for a MOS equations output variable references an Equations_List and Predictand_List variable. These variables provide ordered lists that apply to the dimensions of the MOS equation 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 variables, provide all the metadata needed to access and use the predictors.

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 equation 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. The following CDL shows an example of how such variables are encoded.

```
|    float Standard_Error_Estimate(number_of_stations, number_of_predictands) ;
|      Standard_Error_Estimate:PROV__Entity = "StatPP__Methods/Stat/MOS/OutptParams/MOSStdErrEst" ;
|      Standard_Error_Estimate:standard_name = "source" ;
|      Standard_Error_Estimate:long_name = "MOS Standard Error Estimate" ;
|      Standard_Error_Estimate:coordinates = "station Predictand_List" ;
|      Standard_Error_Estimate:SOSA__usedProcedure = "( PolyLinReg )" ;
|      Standard_Error_Estimate:units = 1LL ;
|    float Reduction_of_Variance(number_of_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 )" ;
```

References

[CF] NetCDF Climate and Forecast (CF) Metadata Conventions. Eaton, B., et al. Version 1.6 accessed on 27 February 2017.

[DMWMO] Guidelines for Data Modelling for WMO Codes. WMO Commission for Basic Systems. 2014. 127 pages.

[GCSN] Guidelines for Construction of CF Standard Names. 2008. Version 1.

[MD1] ISO 19115-1:2014, Geographic information – Metadata – Part 1:Fundamentals. ISO/TC 211 Secrtariat. 167 pages.

[MD2] ISO 19115-2:2009, Geographic information – Metadata – Part2: Extensions for imagery and gridded data. ISO/TC 211 Secretariat. 143 pages.

[NEDCDFLD] netCDF-Classic-LD. Accessed on 14 June 2018.

[NCU] NetCDF Uncertainty Conventions. OGC Discussion Paper.. Available at NetCDF-U Web Home wiki. Open Geospatial Consortium. 20 pages.

[O&M] ISO 19156:2011, Geographic information – Observations and measurements. ISO/TC 211 Secretariat. 60 pages.

[PO] ProbOnto 2.5 Ontology and Knowledge Base of Probability Distributions. Available at probonto.org web page. Swat, et al. 302 pages.

[SOSA] Semantic Sensor Network Ontology, Accessed on 15 June 2018.

[TS] ISO 19108:2002 and ISO 19108:2002/Cor 1:2006, Geographic information – Temporal schema. ISO/TC 211 Secretariat. 55 pages.

The following sections detail the use of various necessary components required for a Netcdf-CAMPS data file. Requirements, best practices, and justifications are given for a netCDF-CAMPS data file. Extra care is given in explaining some of the more complicated aspects, such as; time coordinates, event probabilities, and percentiles.

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.)

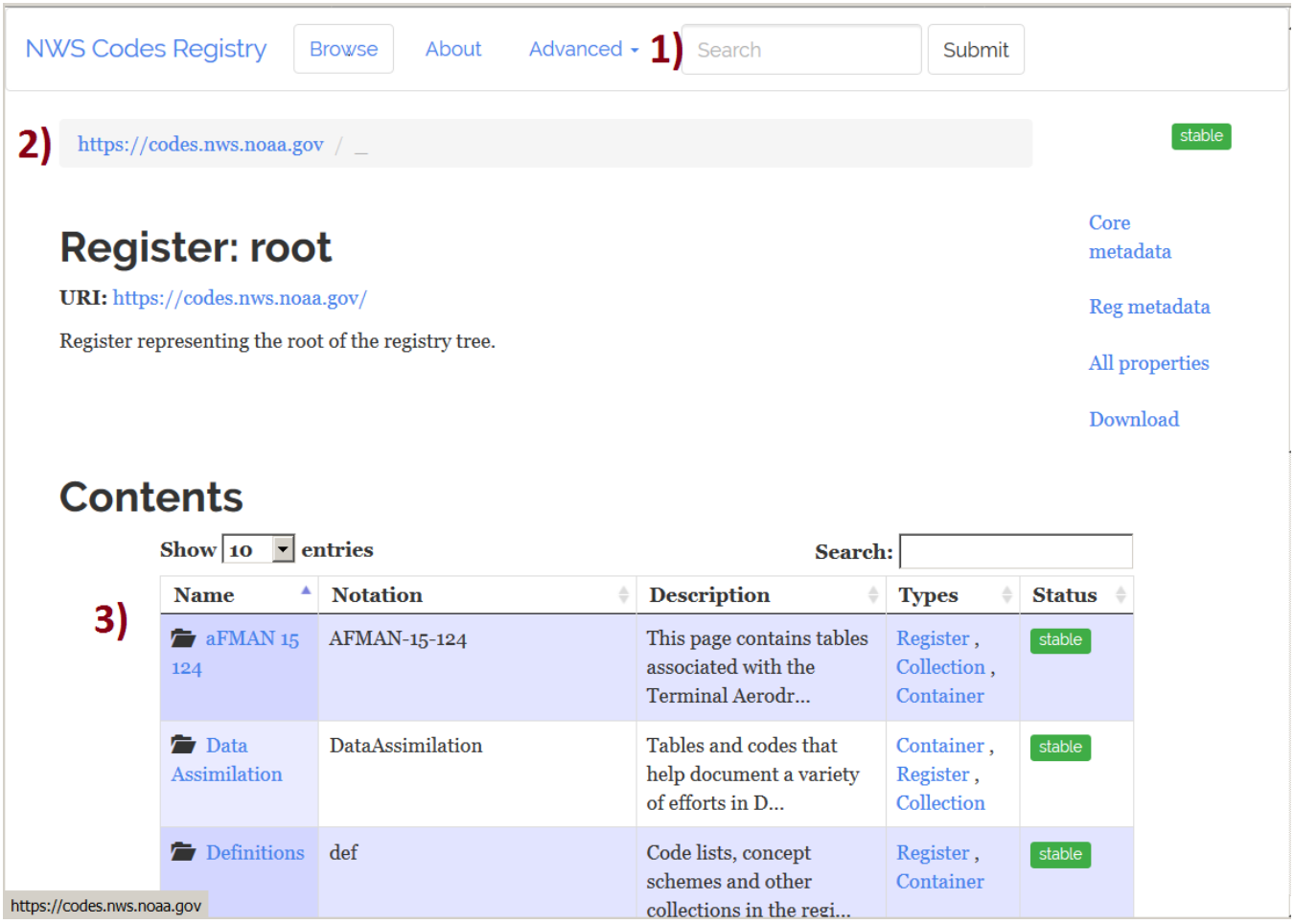


Figure 1: Main or "Home" page of Registry at root level

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

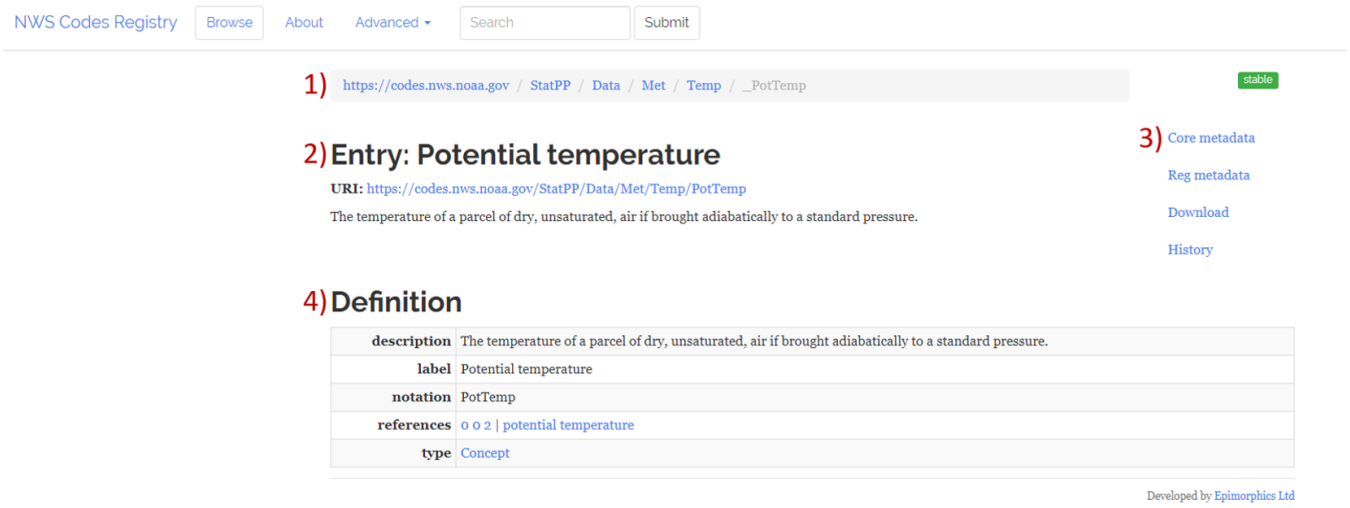


Figure 2: Registry GUI display for “Potential Temperature” under /StatPP/Data/Met/Temp

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.

CAMPS Techinical description

Overview

The purpose of this section is to provide some detail on the technical and unseen side of CAMPS. It will also cover the reasoning for certain design decisions.

Goals and Scope

For context, it's important to understand what problems the CAMPS project does and does not try to address.

It does:

- Provide a way of encoding CAMPS formatted metadata.
- Aid in the processing of predictors from model output.

- Have modules for converting observations and model output to netCDF.
- Allow for alternative statistical postprocessing techniques to be added.

It does NOT:

- Have ways of remotely pulling data (You'll need to have the data available locally).
- Promise to be able to handle all file types.
- Support very diverse situations. It was intended to be used with NWP output, for multiple linear regression post processing. However, with a bit of hacking, other types of StatPP applications could be implemented.

Python version and Libraries

Anaconda2-5.3.1 (or newer) is recommended for running CAMPS. The following libraries are required even when using Anaconda.

Language	Version
python	2.7.13

Library (with Anaconda)	Recommended Version
netCDF4	1.5.1
pyproj	1.9.5.1
pygrib	2.0.3
metpy	0.9.2
basemap	1.2.0
basemap-data-hires	1.2.0

Note

If not using Anaconda, the libraries in the following table must be added

Library (without Anaconda)
matplotlib
scipy
numpy
PyYAML
matplotlib
pandas
seaborn

Directory Structure

camps/camps/core

Contains code that defines the Camps_data object. Also stored here are composite classes such as Time and Loaction, 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 here.

camps/camps/core/data_conversion/marine_to_nc

Marine buoy observations to netCDF code here.

camps/camps/core/data_conversion/metar_to_nc

Metar observations to netCDF code here. Also includes QC.

camps/camps/registry

Contains all the example configuration and control files for the various drivers. Also, contains a key file, netcdf.yaml, that is used widely in metadata templating.

camps/camps/registry/db

Contains the sqlite database and associated code to access it. The database stores all variables and their metadata that have been written from the used package

camps/camps/scripts

This contains all the driver scripts. All scripts require a control file to be used as an argument when running the script.

camps/camps/StatPP/regression

Code that contains the modules used for the multiple linear regression.

camps/camps/gui

Various GUIs and modules used to display data are here.

camps/camps/mospred

Contains modules that support the capabilities of the u201 equivalent code, which does a does a number of things. It creates new predictors and predictands – typically from model output. It applies procedures to the variables, such as smoothing and interpolating. It organizes the variables into appropriate dimensions.

camps/camps/libraries/mathlib

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

Running CAMPS Software

Every driver script in CAMPS will have an accompanying configration file that controls certain aspects of the run. It is imperative that you read the appropriate example config file to determine how to format your control file for the given driver script you are running. Without a control file passed as an argument, CAMPS software will error out.

METAR processing:

Driver: camps/camps/scripts/metar_driver.py

Config example: camps/camps/registry/metar_control.yaml

Marine bouy processing:

Driver: camps/camps/scripts/marine_driver.py

Config example: camps/camps/registry/marine_control.yaml

grib2 processing:

Driver: camps/camps/scripts/grib2_to_nc_driver.py

Config example: camps/camps/registry/grib2_to_nc_control.yaml

mospred processing:

Driver: camps/camps/scripts/mospred_driver.py

Config example: camps/camps/registry/mospred_control.yaml

equation processing:

Driver: camps/camps/scripts/equations_driver.py

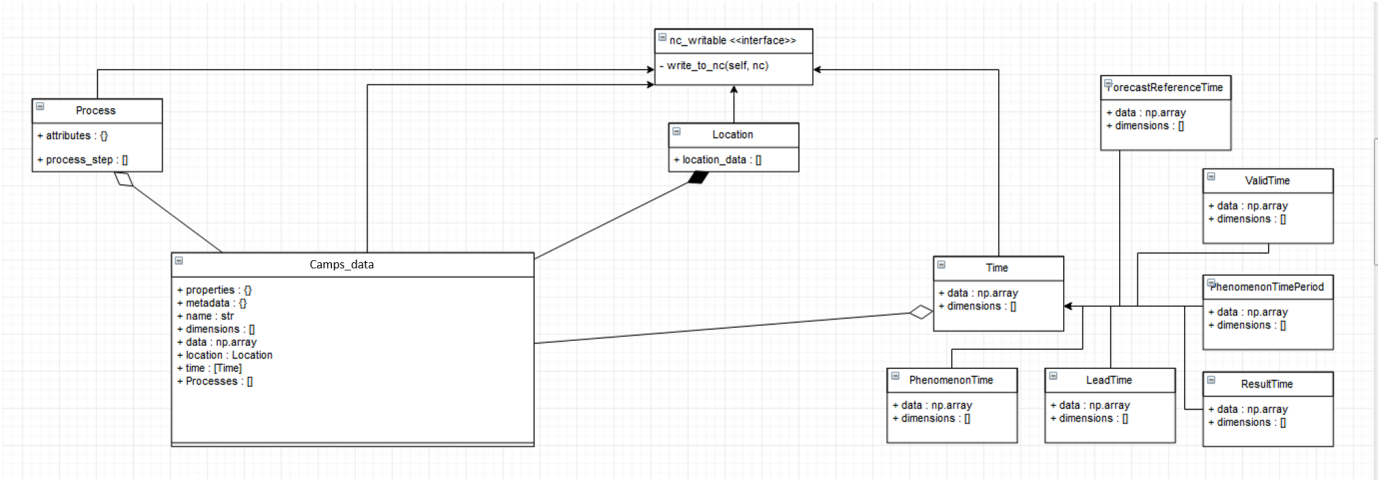
Config example: camps/camps/registry/equations_control.yaml

forecast processing:

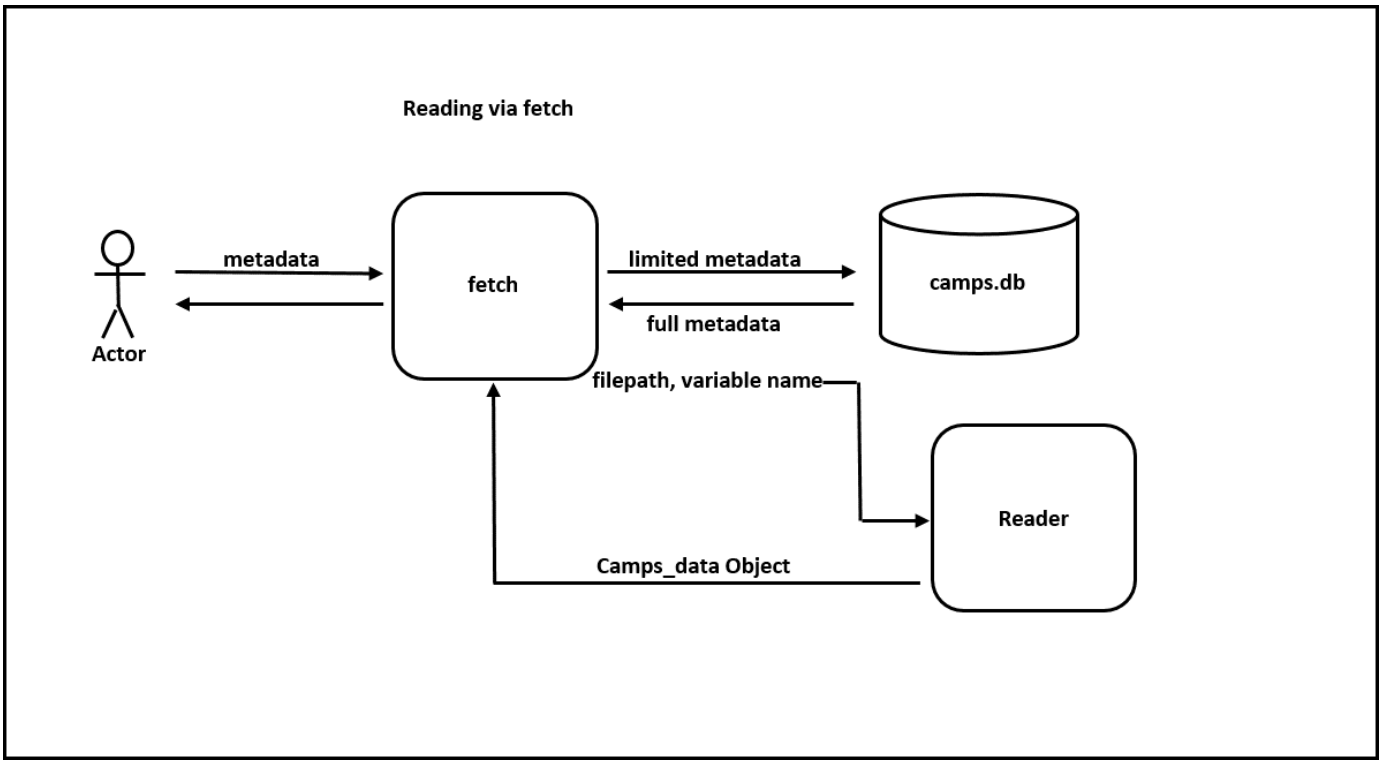
Driver: camps/camps/scripts/forecast_driver.py

Config example: camps/camps/registry/forecast_control.yaml

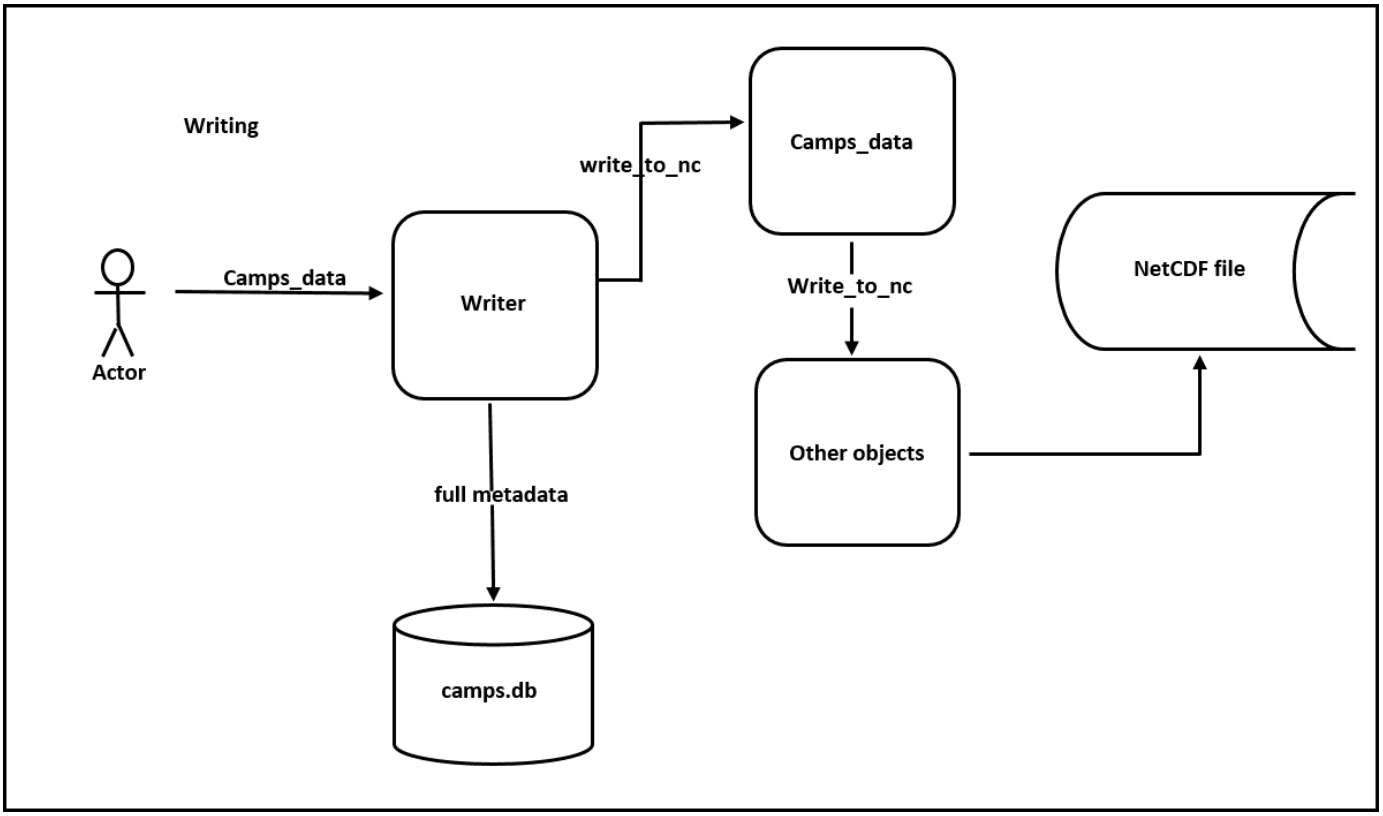
Architecture and Design



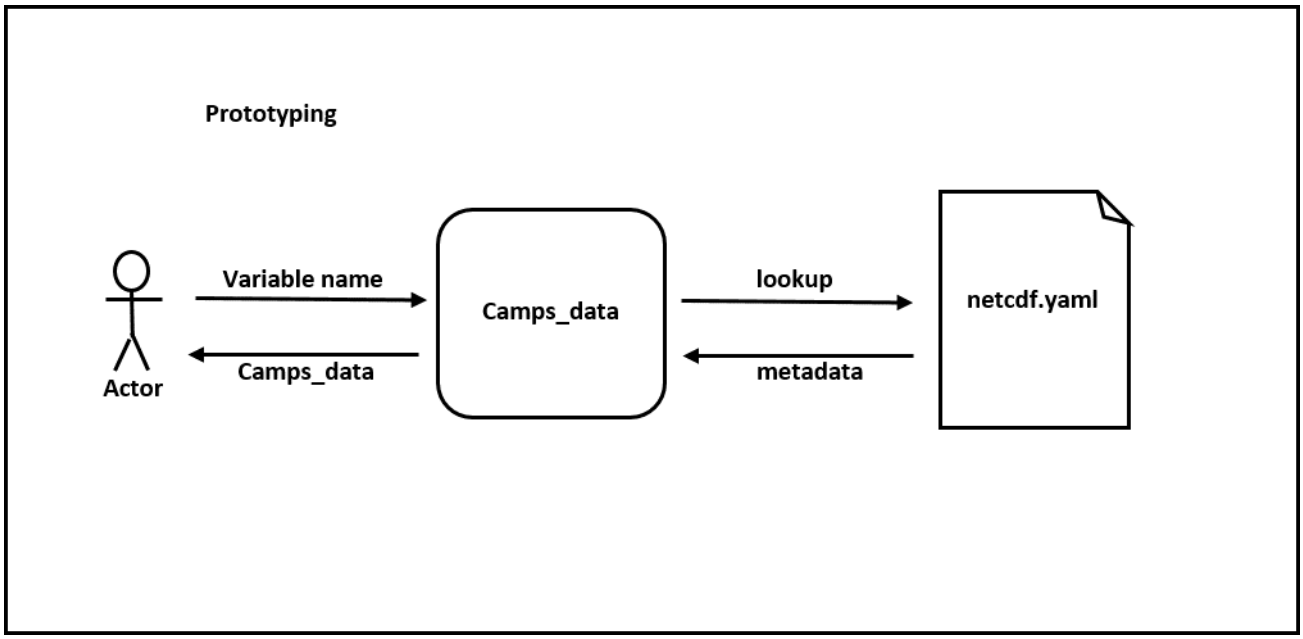
CAMPS' core data structure is the Camps_data class. It is a composition of different components that often represent variables that can be written to a netcdf file independently of the parent object. Classes like Time, Location, and Process all share a common interface, **nc_writable** which enforces the definition of the method **write_to_nc**.



Fetching utilizes the database, which keeps track of variables and their metadata. If fetch is passed metadata, it will return one of the entries that contains that metadata. If a phenomenon time or lead time is also given, it will slice the variable based on those parameters.



Writing a variable simply invokes the Camps_data object's **write_to_nc** function, which in turn calls composite object's **write_to_nc**. Upon succesful write, the variable and its metadata are stored in the database.



Prototyping is CAMPS' way reducing time spent adding metadata to an object. Metadata is defined in **netcdf.yaml** and is added to a Camps_data object when initialized with the appropriate key.

Camps Data Object

Introduction

The Camps Data class should be a representation of ANY kind of 'CAMPS' formatted data. That is, the data can be gridded or vector, it may or may not have a lead time associated with it, and can be n-dimensional.

Dimensions

Allowing for the flexiblility of the object, the number of dimensions of the data are unlimited. However, these dimensions should be able to be identified.

For example, Assume there is a variable that is gridded and is a snapshot of a single time. This variable would have three dimensions: x, y, and time. Another variable is a vector of stations and includes lead times at multiple forecast reference times. This would also have three dimensions: number of stations, lead time, and time.

Standard dimension names are located in netcdf.yaml in the 'Dimensions' section.

These separate variables should be distinguished based on other metadata.

Creating Camps_data Objects

Regardless of the type of data, there are similar procedures for creating a camps data object.

First, You'll need to initialize the object. If the argument passed into the constructor is a key defined in **netcdf.yaml**, such as,

```
>>> wspd = Camps_data('wind_speed_instant')
```

then it will be initialized to the metadata corresponding to that key.

Next, you'd want to give your object a time period. There are different *flavors* of Time in CAMPS, so which ones you need depend on the application. However, there's a very good chance you'd need a phenomenon time defined. You might create it like this,

```
>>> ptime = Time.PhenomenonTime(start_time='19910518', end_time='20180810')
```

And add it to your object like this,

```
>>> wspd.time.append(ptime)
```

CAMPS will try to handle the formatting, writing, and reading of these variables.

Sometimes, you'll want to apply a procedure to the variable. If that's the case, it would be a good idea to add a Process object to your object.

A Process is created similar to a Wisps_data object, you can initialize it using a key in **procedures.yaml**.

```
>>> p = Process('BiLinInterp')
```

Lastly, since we're wring to netcdf, you need to provide dimensions to your object. You can name them whatever you'd like, but there are a few dimensions that have special properties, that act slightly differently. These are found in **netcdf.yml** in the dimensions section.

```
>>> wspd.add_dimensions('lat','lon')
```

Fetching and Database

The fetch module is the CAMPS way of pulling data by metadata. Traditionally, this would be done by searching all files in a given directory until the correct variable was found. For CAMPS, to improve performance, we opted to use a sqlite3 database to keep track of where our variables are located and what metadata is associated with them. When a user initially imports CAMPS from a python interpreter a hidden directory named ".camps" is created in their "home" directory. This is where the database will be stored automatically when running a CAMPS application. Each user should have their own database, although if two users are on the same machine with similar file permissions, a database could be easily shared.

In most cases, interacting with the database directly is unneeded, as the normal interaction with CAMPS will handle updating the database. In cases where direct intervention is necessary, the db.py module has some helper methods to assist in this interaction. Of course, those comfortable with SQL can modify it directly with some very limited knowledge of python's sqlite3 package or by running sqlite3 directly.

Time

Time in CAMPSS is divided into 5 concepts, and the software makes use of 5 distinct classes to handle their differences. They include, ForecastReferenceTime, LeadTime, PhenomenonTime, PhenomenonTimePeriod, ResultTime, and ValidTime.

Registry and configuration

Examples

Below are a few very basic examples of data manipulation in CAMPS. Since CAMPS was built with specific metadata requirements, the use of the built in data objects and I/O utilities is necessary.

Create

```
>>> from camps.core.Camps_data import Camps_data
>>> import numpy
>>> wspd = Camps_data('wind_speed_instant') # Initializes object
>>> wspd.data = numpy.random.rand(10,10)*50 # Assign data
>>> wspd.add_dimensions('x','y') # Need to specify dimension names
>>> wspd.metadata['my_important_attribute'] = 42
>>> wspd
***** wind_speed ******
*
* dtype               : float64
* processes           : ( )
* dimensions          : ['y', 'x']
  Metadata:
* comment             : Wind speed is set to -9 if winds are variable.
* OM__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:
(10, 10)
Data:
[[ 44.32559503  29.6
29957  48.87075532]]
```

Write

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

```
$ ncdump output.nc
```

```
netcdf output {
dimensions:
        y = 3 ;
        x = 3 ;
        elev = 1 ;
        level = 1 ;
variables:
        int64 elev0(elev) ;
                elev0:long_name = "height above surface" ;
                elev0:units = "m" ;
                elev0:standard_name = "height" ;
                elev0:positive = "up" ;
                elev0:axis = "Z" ;
        double WindSpd_instant__(x, y, level) ;
                WindSpd_instant__:_FillValue = 9999. ;
                WindSpd_instant__:comment = "Wind speed is set to -9 if winds are variable." ;
                WindSpd_instant__:OM__observedProperty = "StatPP__Data/Met/Moment/WindSpd" ;
                WindSpd_instant__:ancillary_variables = "" ;
                WindSpd_instant__:valid_min = 0. ;
                WindSpd_instant__:coordinates = "elev0 x y" ;
                WindSpd_instant__:long_name = "horizontal wind speed" ;
                WindSpd_instant__:standard_name = "wind_speed" ;
```

```
                WindSpd_instant__:my_important_attribute = 42LL ;
                WindSpd_instant__:valid_max = 75. ;
                WindSpd_instant__:SOSA__usedProcedure = "( )" ;

// global attributes:
                :primary_variables = "WindSpd_instant__" ;
                :version = "CAMPS-1.0" ;
                :references = "" ;
                :file_id = "b0592d4f-88b1-4160-adb9-9cb031c025dd" ;
                :url = "http://www.nws.noaa.gov/mdl/, https://sats.nws.noaa.gov/~wisps/" ;
                :organization = "NOAA/MDL/SMB" ;
                :institution = "NOAA/National Weather Service" ;
                :Conventions = "CF-1.7 CAMPS-1.0" ;
data:

 elev0 = 2 ;

 WindSpd_instant__ =
  6.59734754874646,
  41.240254448533,
  40.076160141887,
  40.8188432115729,
  45.445944531748,
  33.2932430382621,
  16.531263567237,
  13.2378413416106,
  41.7054178526444 ;

group: prefix_list {

  // group attributes:
                :StatPP__ = "http://codes.nws.noaa.gov/StatPP/" ;
                :OM2__ = "http://codes.nws.noaa.gov/StatPP/" ;
                :SOSA__ = "http://www.w3.org/ns/sosa/" ;
                :OM__ = "http://www.w3.org/ns/sosa/" ;
                :PROV__ = "http://www.w3.org/ns/prov/#" ;
                :StatppUncertainty__ = "http://codes.nws.noaa.gov/StatPP/Uncertainty" ;
  } // group prefix_list
}
```

CAMPS start to finish

Once CAMPS is properly installed into your Anaconda2 distribution, make sure you do the following first step:

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

This first import to the camps module will create a hidden directory in your home directory which contains both the CAMPS database and a control directory which contains all your control file templates.

CAMPS makes use of the python feature “console scripts” which allows a user to run driver scripts directly from the command line

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

Obs

To start, you’ll need to process some observations.

Go to `camps/camps/registry/metar_conrol.yaml` to view an example control file. Then create your own control file, following the example syntax very carefully.

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

Let's get some marine observations too. Again, you'll want to view the example and create your own control file, `camps/camps/registry/marine_conrol.yaml`, and then execute the driver.

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

Model

Great! We have our observation data. Let's try to process some model data. For now, grib2_to_nc only accepts grib2 data on a Polar Stereographic grid. More functionality will be added as CAMPS continues to grow.

First, create your control file, as done previously `camps/camps/registry/grib2_to_nc_control.yaml`. This might start to get repetative. Then,

```
$ CAMPS_grib2_to_nc /path/to/controlfile/grib2_to_nc_control.yaml
```

So far so good?

Generating Predictors

Now, model data doesn't have *everything* we want for a regression. Sometimes, we'll want to create our own predictors that are derived from the model variables. We can also apply different procedures to the data we produce that changes its characteristics. Also, if we're doing a station based MOS run, we'll need to interpolate to stations.

Mospred_driver can be run for both creating predictors and/or predictands. Be sure to specify which (or both) you want to run for.

In addition to the standard control file, mospred_driver also requires a couple other input files, the locations of which should be added as arguments inside mospred_control.yaml.

The config file to follow: `camps/camps/registry/mospred_conrol.yaml`

And the driver,

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

Regression

Now we're ready for the regression. The config file `camps/camps/registry/equation_conrol.yaml` will give an example of tuning, and is where you can specify which predictors and predictands you want run the regression over.

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

Forecast

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

The config file to follow: `camps/camps/registry/forecast_control.yaml`

And the driver,

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

Output

That's it! That is all you need for a MOS-2000 like MOS development using CAMPS software! All output files are saved in NetCDF format and can be found in the output paths you specified inside each driver control file.