# INPUT DATA Directory Structure for FUDGE darkchocolate and beyond

Note:
Moving forward, all input climate variable-based data files are to be stored in this directory structure, regardless of the original data source or scientific project being pursued. The structure is very similar, but not identical, to what has been used in previous ESD work, which in turn was based in large part upon DRS and CMIP standards. This revised structure aims to define a single directory structure that can meet the needs of the wider range of ESD Team science applications planned for 2015.

Somewhat paradoxically, by defining a more standardized and rigid overall input data directory structure, the revised approach affords us greater scientific flexibility and efficiency when setting up and executing new experiments. Additionally, the standardized structure should contribute to streamlining and reducing ongoing maintenance of those parts of the workflow that parse XML to determine input directory paths (e.g., for expergen, this can diminish or remove the need for look-up tables and logic to handle 'special cases' and eliminate defaults for which there would otherwise be many exceptions in 2015 projects). After specifying the parent or $inRoot directory, all input data files will reside in directory paths having the exact same number and type of subdirectories. This aspect of standardization eliminates the need to have the expergen and postProc parts of the FUDGE workflow, analysis software, and the humans that run those jobs, adapt to cases in which the number or ordering of input directories may have differed.

Compatibility of file locations across FUDGE cinnamon and darkchocolate (and later) versions may be handled via a combination of symbolic links and/or file copies, and thus be transparent to the workflow. Similarly, differences between this updated input directory structure and DRS conventions can be addressed, if needed at a later date, via copying files from the standardized FUDGE paths and names to a DRS-compliant structure. That this structure diverges from GCM-oriented DRS conventions is acceptable and deemed to be a requirement if we are to facilitate the organization, generation, and analysis of FUDGE's input files and downscaled outputs.

This revision of the FUDGE input directory structure will affect the XML and software that parses the XML. In effect, the benefits of a more flexible and nimble FUDGE experiment set-up are being pursued by shifting some responsibilities 'upstream'. For one, this approach will determine aspects of how the ESD Team's input files are stored in /archive. Also, the XMLgen tool will be writing more information to the XML files, while ensuring internal consistency across the different XML elements.

Together with updates being made concurrently to generalize file naming conventions and output file specifications, we envision that the expergen and postProc tools will be able to treat the vast majority, if not all, of XML elements associated with building input file names, directory paths, and metadata associated with the input files as arbitrary character strings that can be used without needing to apply specialized logic to handle different cases. Other non-input directory relevant XML elements that control R-code downscaling method options may still require more individual treatment and logic in expergen.

The color-coded documentation that follows divides the standardized input data directory structure into 13 individual pieces. In so doing, we present the information at the finest level of granularity, providing somewhat descriptive names that can be used by code and people alike when considering the meaning of the different directory levels. Currently, it is anticipated that each of the color-coded building blocks ultimately be represented in an individual XML element, subject to change only if compelling reasons are found to support alternative ways to enhance overall robustness of the workflow.

## INPUT DATA Directory Structure

Beginning with FUDGE version darkchocolate, all of the ESD team's input data will be stored in locations conforming to this structure.
If needed to ensure compatibility between the FUDGE cinnamon and darkchocolate branches, we will copy (duplicate) existing cinnamon input files to new locations -or- use symbolic links.

==========

**INPUT DIRECTORY FORMAT =**
**$inRoot/$dataCategory/$dataType/$dataSource/$epoch/$freq/$realm/$misc/$rip/**
**$dataVersion/$variable/$gridDomain/$dim**

==========

**$inRoot** = root (or parent) directory path, below which input data files (and other files) are stored.
Will often be /archive/esd/PROJECTS/DOWNSCALING

**$dataCategory** = Allows coarse level grouping of data into general categories or group types.
Current and anticipated future values include, but are not limited to "GCM_DATA", "OBS_DATA", "SYN_DATA", "REANLYS_DATA".

**$dataType** = A second level grouping of data for sets of data under each $dataCategory. Generally, a particular $dataType will be specific to one $dataCategory.
For example, one could have "CMIP5", "CMIP3", "NARCCAP" under GCM_DATA. Under OBS_DATA one could have "GRIDDED_OBS" and "STATION_DATA". And there will be total flexibility in how to organize and name groups of SYN_DATA types (e.g., "FUDGE_TEST", "NORMAL", "GAMMA", "EXPONENTL", "LONGTAIL")

**$dataSource** = At this 3rd level of data groupings, we have unique identifiers for the source of the data, re-using names from original sources when applicable.
For example, re-use CMIP5 model names such as MPI-ESM-LR, GFDL-HIRAM-C360, MIROC5 per conventions at http://cmip-pcmdi.llnl.gov/cmip5/availability.html. However, at times we may want to modify names to reflect GFDL in-house needs. And when reasonable established names do not exist, we will develop descriptive names, such as we did with "livneh", "prism" and "daymet" under OBS_DATA/GRIDDED_OBS.

The trio of $dataCategory/$dataType/$dataSource often will be thought of as linked, when designing and discussing experiments. As shorthand, only the $dataSource term may be used, with the $dataCategory/$dataType/ levels implied. To date, this shorthand is applicable because each $dataSource appears just once in our archive.

**$epoch** = This subdirectory level provides a general description of the time period or epoch covered by the data sets below. Usually, different dataSources grouped under a single $dataType will have the same set of $epoch subdirectories.

Current examples include "amip", "historical", "future", "rcp26", "rcp45", "rcp85", & "sst2090". The subdirectory level can be re-purposed SYN_DATA cases for which the concept of a temporal epoch does not apply.

**$freq** = Patterned after GFDL Fre and CMIP styles, this identifies the time interval between successive data points in the time series files (aka sampling frequency).

Through cinnamon, we have only used "day". During 2015 we may have need for others, such as "mon", "1yr", and "ssn". Synthetic data designed to mimic certain temporal sampling frequencies could adopt the same directory names, while more generic synthetic data could re-purpose this level and adopt another name.

**$realm** = Patterned after GFDL Fre and CMIP conventions, this is generally intended to identify the component of the physical climate system represented by the data sets below.

Through cinnamon, we have only used "atmos". The most common 4 examples are: "atmos", "land", "ocn", "ice". It's possible that some other designation might become necessary, for example, in a case where the data is derived from two or more variables that are from different realms.

**$misc** = This sub-directory level will serve somewhat different purposes, affording us another degree of freedom to categorize our data.

In some cases, using a MIP table name could be reasonable. But often the data will not be directly associated with a MIP table. In those cases, $misc will be re-purposed. For example, data that has been processed into anomaly fields via use of a 15 day boxcar smoother could be given the value of "anom15SBX". Some indices computed from input data sets (e.g., GCMs or observations) perhaps could be grouped under "climdex" , etc.

**$rip** = A string that identifies ensemble members for different ways of generating multi-member ensembles. Generally in form "rKiMpN" where K,M,N are integers. The general convention is r=realization (varies for climate model ensemble members started from different initial conditions); i = initialization_method (we may encounter variations in this if we downscale seasonal forecasts); p = physics_version (varies in perturbed physics ensembles).

For CMIP data sets, the $rip identifier will be identical to that found in the CMIP archive. For non-CMIP data sets, we will adapt this to provide info meaningful to our applications. (e.g. We have use "r0i0p0" and "r0i0p1" as in-house conventions for *observation* data sets to simply differentiate between Julian (r0i0p0) vs NOLEAP calendar (r0i0p1) time series.

**$dataVersion** = version of the data.

**$variable** = long name of input variable of interest (tasmax, tasmin, psl, ta850,etc.).

This subdirectory level is applicable to climate input variables that will be specified in the XML both as "target" and "predictor".

**$gridDomain** = Name given to the rectangular spatial domain covered by an input data set.

In cinnamon, this is defined by the <grid region=" "> element in XML. (e.g. "SCCSC0p1" for Red River and the pending 3 to the 5th project, and US48 for the NCPP-era Perfect Model data sets.)

**$dim** = identifies the subdirectory in which the input files are stored at the lowest level. **$dim** refers to the spatial dimension.

Conventions are "OneD" for the oft-used "minifiles" (1 longitude per file, many latitudes) and "ZeroD" (each file contains a single geographic point) Otherwise, names are given to two-dimensional data file arrangements (e.g., "US48", "9pts").

---

**INPUT FILE NAME IS DEFINED AS:**

**$variable_$freq_$dataSource_$epoch_$rip_$gridDomain_$trange.**
**I$Islice_${file_j_range}.nc**

---

where

**$variable**, **$freq**, **$dataSource**, **$epoch**, **$rip**, **$gridDomain** are as defined above.

**$trange** = time range character string. To date, we have used the form YYYYMMDD-YYYYMMDD for daily data (when $freq = day). Currently the MMDD is hardwired in expergen, R-code and possibly postProc. The starting and ending YYYY character strings come in through XML. To increase generality, support for $freq values other than "day" (e.g., "mon", "1yr") need to be made. Common conventions are for monthly data to use the YYYYMM form and annual data the YYYY form. Additionally, for synthetic data, one could envision $trange character string having somewhat different forms, such as "10000N" for a time series having 10,000 time levels.

**$Islice** = Thus far, for OneD minifiles and ZeroD files, a single longitude index has been encoded in the file name (e.g., I924). Currently, the FUDGE workflow generates a separate runscript for each $Islice value, with the full set of longitude index values defined in XML.

**${file_j_range}**. = defined in XML, this character string documents the latitude index range covered by the input file.

Examples:

Red River data : /archive/esd/PROJECTS/DOWNSCALING/GCM_DATA/CMIP5/MPI-ESM-LR/historical/day/atmos/dayr2i1p1/v20111006/tasmax/SCCSC0p1/OneD/tasmax_day_MPI-ESM-LR_historical_r2i1p1_SCCSC0p1_19610101-20051231.I300_J31-170.nc

Perfect Model: /archive/esd/PROJECTS/DOWNSCALING/GCM_DATA/CMIP5 GFDL-HIRAM-C360/amip/day/atmos/dayr11p1/v20110601/tasmax/US48/OneD/tasmax_day_GFDL-HIRAM-C360_amip_r1i1p1_US48_19790101-20081231.I924_J454-567.nc