DataMgr and VDC library

Overview

DataMgr

Function

- Provides a common interface for read-only access to the VDC as well as all foreign data formats (e.g. WRF, ROMS, etc). This is implemented by providing an abstract base class (DataMgr) that defines virtual protected methods that must be implemented by file format-specific subclasses of the DataMgr. For example, the DataMgrWC subclass provides access to VDC type II data, while DataMgrWRF provides access to files stored in the WRF netCDF output format. In addition to protected methods, DataMgr defines public methods that are invoked directly by DataMgr clients (i.e. vaporgui). These public methods define a common API for read-only access to all supported data formats.
- Caching: The DataMgr maintains a cache for all potentially expensive access operations. Currently the cache includes: gridded variable data (for both derived and native variables); the range of values (min and max) for a data variable; whether a particular variable/timestep/refinement-level exists (is present in the file, or can be derived from native variables); and the min and max IJK voxel coordinates of a data variable (the current VDC allows variables to be partially present in the sense that only a rectangular subregion of the variable may be present in the file)
- Derived quantities: In addition to "native" data variables, those present in a data set, the DataMgr provides support for defining new, derived quantities. Currently this is supported via vaporgui's numpy interpreter. By placing support for derived quantities within the DataMgr all of the benefits of data caching can be reaped.

Limitations/issues

- Assumes all data variables in a data collection are sampled on a single, regular grid. Thus supporting many commonly used data sets requires regridding all data onto a single grid
- Doesn't fully support general structured grids. Regular grids are the only first class citizens. Extensions have been made to allow for Rectilinear (stretched) and terrain following grids.
- Assumes a Cartesian coordinate system. Though a few hacks have been made in an attempt to support data defined with spherical coordinates in practice data must first be projected to a planar space with rectangular coordinates.

Proposed DataMgr API

Deprecated DataMgr methods

The following methods would be deprecated. In most cases the provided functionality is moved elsewhere.

- GetDim() - moved to SGVariableInfo class
- GetNumTransforms() - moved to SGVariableInfo class
- GetCRatios() - moved to SGVariableInfo class
- GetCoordSystemType() - moved to SGVariableInfo class
- GetGridType() - moved to SGVariableInfo class
- GetExtents() - redundant (available in RegularGrid class)
- GetMapProjection() - moved to SGVariableInfo class
- GetPeriodicBoundary() - moved to SGVariableInfo class
- GetGridPermutation() - moved to SGVariableInfo class
- GetVarType() - moved to SGVariableInfo class
- GetMissingValue() - moved to SGVariableInfo class
- MapVoxToUser() - equivalent functionality available in RegularGrid class
- MapUserToVox() - equivalent functionality available in RegularGrid class
- GetEnclosingRegion() - equivalent functionality available in RegularGrid class
- IsCoordinateVariable() - equivalent functionality available in SGVariableInfo class
- GetValidRegion() - Storing a spatial subset of a variable will no longer be supported because it is not portable.

New and changed DataMgr methods

Proposed changes to the DataMgr access methods are described here:

- Version 1 (07/02/2013)
- Version 2 (07/16/2013)

(In addition to the changes made above, the RegularGrid class object would need to be sub-classed to provide a general structured grid object (e.g. StructuredGrid or CurvilinearGrid)

VDC

Function

- Provides API for both reading and writing data into a VAPOR Data Collection

Limitations/Issues

- The API has evolved substantially since the initial incarnation of the libvdf library. It is neither well designed, nor well documented. Examples of problem areas include:
    - Missing data value support
- As with the DataMgr the VDC API restricts a data collection to being defined on a single grid.
- The API does not resemble APIs for other commonly used scientific data formats (e.g. netCDF), nor does it provide sufficient functionality for representing common data sets.

Requirements/Desirables

- NetCDF file format (CF compliance where possible)
- Support direct simulation model outputs
    - Number or time steps and output times may not be known apriori
- Support access with NetCDF API wherever possible (e.g. for non-compressed variables, attributes, coordinate variables, dims, etc)
- Multiple time steps or variables per file?
- Append and/or modify existing VDC
- Ability to easily edit/append attributes in top-level (master) file. For example, adding new variables, new time steps.
    - Need use cases

- Ability to merge/insert new times

Proposed VDC API

Draft APIs:

- Version 1 (7/16/2013)
- Version 2 (7/25/2013)
    - Replaced DefineDimensions() with DefineDimension()
    - Added Dimension, CoordVar, and DataVar helper classes

Use Cases

vdfcreate.txt - replicates current vdfcreate functionality

momvdfcreate.txt - replicates momvdfcreate

wrfvdfcreate.txt

wrf2vdf.txt

VDC file organization (in progress)

A master NetCDF file that replaces the current XML based .vdf file. Why replace XML? 1) a NetCDF file is readable by NetCDF API. 2) XML has no support for binary

VDC Master file contents:

- Global attributes
- Table of contents
    - Data variable and coordinate variable definitions
        - Everything need to define a netCDF variable in a variable file (e.g. name, dimnames, coordvars, attributes)
    - # time steps?
    - Information needed to construct paths to data and time-varying coordinate variables
- Static (non-time varying) spatial coordinate variables
    - n.b. his could potentially make the VDC Master large
- Time coordinate variable(s)
- default compression attributes (e.g. bs, wavelet, cratios)
- # time steps per file (relax current requirement that each time step resides in a different file

VDC Master API:

The master file can be accessed directly by the NetCDF API. However, the methods provided below should be used in place of their NetCDF counterparts.

- Constructor(string path, enum mode)
    - mode is one of create, read, append
- SetMaxTSPerFile(size_t n)
    - Sets the maximum number of time steps in a single file.
- DefineDims(map(string dimname, size_t dimlen)
    - Defines all of the dimensions and dimension names
- DefineCoordVar(string name, string_vec dimnames, enum axes, string units,)
    - axes is one of X,Y,Z,T
    - units is a udunits2 parsable string
    - type is float, int, double, etc.
- DefineDataVar(string name, string_vec coordvarnames, bool compressed, bool has_missing values, float missing_value, bool separate_file)

    - compressed indicates whether the variable is to be compressed or not
    - separate file indicates whether the variable is to be stored in its own file or with other data variables.
    - Data variables may not have coordinate variables for each axes. An alternate means to specify their coordinates is needed.
- GetPath(size_t ts, string varname)
    - Return the NetCDF path for a variable at a given time step
    -
- GetVarID(size_ts, string varname)
    - Return the NetCDF varid for a variable at a given time step (is time step needed?)
    - If the file containing the variable does not yet exist it will be created. Implications here need to be thought out.

Possible VDC file structure with 100 time steps per file, with both compressed and non-compressed data variables:

```
master.nc

master_vars/

  compressed.000-099.nc0 compressed.000-099.nc1 compressed.000-099.nc2

  compressed.100-199.nc0 compressed.100-199.nc1 compressed.100-199.nc2

  notcompressed.000-099.nc

  notcompressed.100-199.nc

  U/

    U.000-099.nc0 U.000-099.nc1 U.000-099.nc2

    U.100-199.nc0 U.100-199.nc1 U.100-199.nc2

   TEMP/

    TEMP.000-099.nc

    TEMP.100-199.nc

master_coord_vars/

  coordvars.000-099.nc

  coordvars.100-199.n
```