GNE (**G**lobal **N**EWS Modeling **E**nvironment) Manual

Including discussion of the NEWS 2 model implementation in GNE

Emilio Mayorga, mayorga@apl.washington.edu
http://www.marine.rutgers.edu/globalnews/GNE

May 24, 2009

Last substantial edits: November 21, 2008

Table of Contents

Introduction and Overview

GNE (**G**lobal **N**EWS Modeling **E**nvironment, pronounced as "genie") is a code package that brings together all Global NEWS models into a common framework. It also provides basic functionality for standard pre-processing of gridded GIS data to create basin-aggregated values for use in a model run. With GNE, anyone can run all the models, with choice on input data to use, output variables to export, models to run, etc.; individual nutrient forms can be run all at once or separately.

GNE is written in Python, a free, open source programming language available on all major operating systems. Model equations are largely coded using the NumPy add-on, which provides a simple array syntax similar to Matlab. While a key strength of Python is that it's fairly easy to learn and understand, no

programming whatsoever is required to simply run the models; changes to the code will generally only be needed when modifying the underlying model equations and core coefficients.

GNE is also a generic Python framework that can be used to implement any lumped basin model that uses basin-aggregated input variables and outputs basin-aggregated variables. Model inputs and outputs are stored in simple (ASCII) text tables (CSV, or comma-separated variables format) that can be easily viewed in a spreadsheet program like Excel.

This manual describes the Global NEWS model implementation in GNE and focuses on Model Users rather than Model Developers or GNE Developers (see GNE user types, below). The manual assumes that the user's computer has Microsoft Windows as its operating system; examples will work fine on Windows 2000, XP and Vista.

GNE user types

Three types of GNE users can be envisioned, listed below by increasing level of programming skills:

1. *Model Users* only need to learn how to setup or change a model run using a set of text configuration files, and to organize input and output files in a regular, structured fashion. Python skills are not required.
2. *Model Developers* can focus on translating model equations into Python GNE code using simple code tools for variable management provided by the GNE framework. They are largely isolated from the "behind-the-scenes", low-level GNE functionality that handles file read-write and basic error checking. Model developers should be fairly comfortable with Python and NumPy.
3. *GNE Developers* may modify any internal aspect of the low-level GNE functionality that does not involve the model equations directly. They should be *very* comfortable programming with Python and NumPy.

Overview of capabilities and functionality

The primary functionality supported by GNE is the ability to run steady-state, basin-scale nutrient models where the inputs are already provided as one or more tables of basin-aggregated variables and the outputs are also at the basin scale (see Figure 1). GNE is particularly useful when the model run encompasses many basins (tens, hundreds, or thousands) and multiple nutrient forms. The current Global NEWS nutrient export model applied at the global scale fits this description.

After this overview, the **Major Components** section contains extensive details about files and file organization in a GNE model run, and particularly about the set up of each configuration file. More information about running GNE and managing model runs can be found in the **Usage, Conventions, Modeling Steps, and Demo** section.

Model run. If an existing model code version can be applied without modification for a particular application and the basin-scale inputs are already prepared and well organized, a "Model User" can set up and execute a model run with just a few steps involving edits to simple text configuration files:

1. In *gensetup.cfg*, specify the base folder path for the model run's inputs and outputs, if necessary
2. In *constants.cfg*, adjust global calibration constants for dissolved sub-models, if necessary
3. In *vars.cfg*, specify model input and output tables and associate GNE variable names with corresponding table column names

Naturally, the model code itself can be modified as well, but this GNE usage level ("Model Developer") is more advanced and is described elsewhere. Together with the model code itself, the configuration files become archives documenting the exact input data used in a model run.

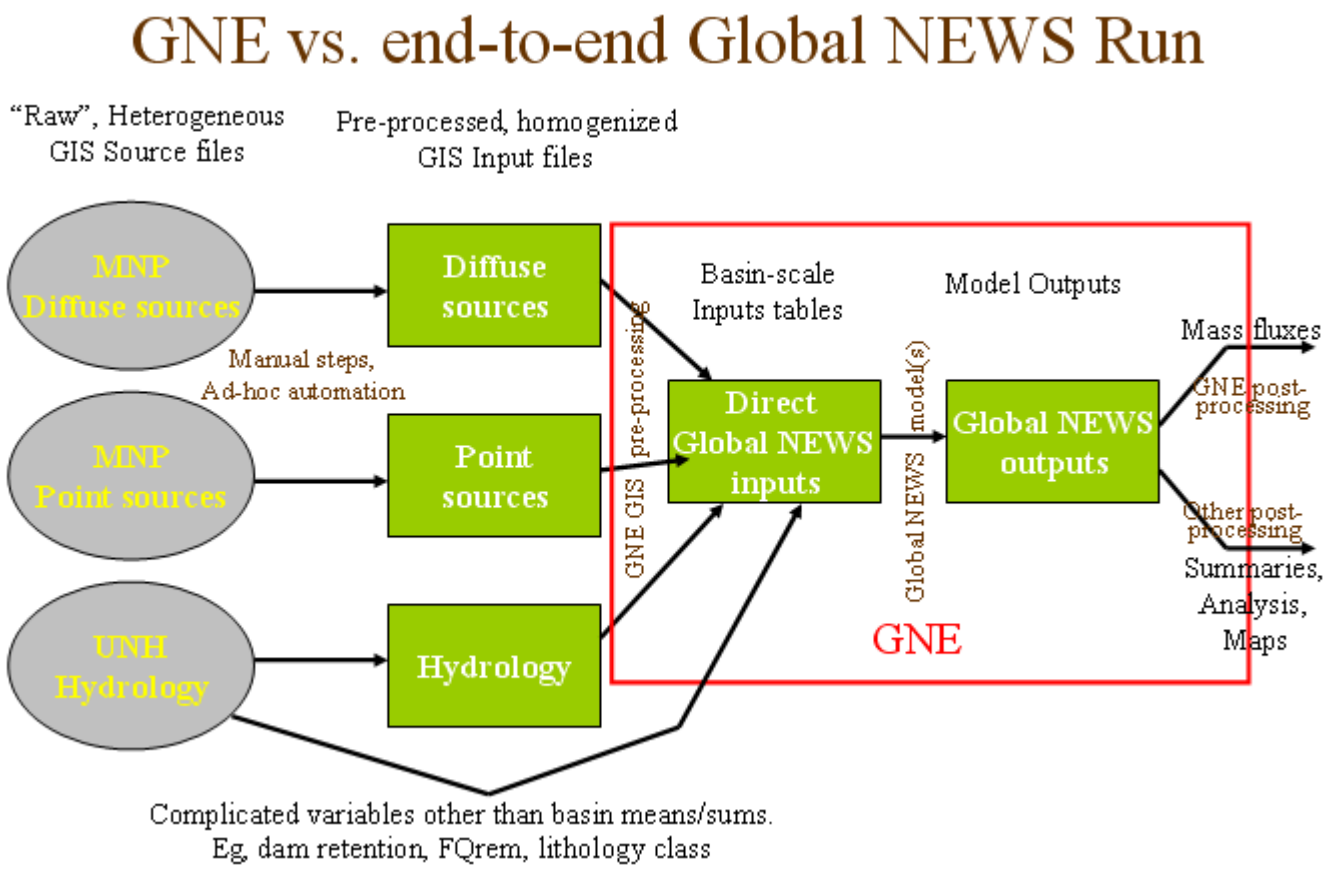

Figure 1. Input data processing and model run involved in Global NEWS. Scope encompassed by GNE.

Model execution is carried out at the Command Prompt window with a simple command. First open the Command Prompt application (typically found under Start > All Programs > Accessories) and change directory to the model-run base folder. The model is executed using a command as simple as this:

```
python globalnews.py
```

Using the "verbose" optional argument ("–v") enables a large set of run-time messages that are printed to the screen during run-time, which can be useful for debugging:

```
python globalnews.py –v
```

Re-directing (">") the run-time messages to a log file (e.g., *modelrun_logfile.txt*) instead of the screen makes the model run more self-documenting, for future reference:

```
python globalnews.py –v > modelrun_logfile.txt
```

Note that re-directing run-time messages to a file that already exists will overwrite that file.

Each of these commands can be placed in a Windows shortcut, but that option will not be discussed here.

GIS pre-processing. GNE also provides a set of easy-to-use GIS pre-processing tools that facilitate the transformation of raster input datasets into basin-aggregated tables of input variables. These tools cover typical, relatively straightforward basin aggregation operations involving the summation or averaging of

spatially distributed, gridded inputs, with some options for cell-area scaling. They rely on the ESRI ArcGIS GeoProcessing engine, so ArcGIS 9.0 or higher *must* be installed. Like mode runs, this functionality is specified via a text configuration file (*gis2tbls.cfg*). GNE GIS pre-processing is executed through the use of the optional argument, "–g", which makes GNE read *gis2tbls.cfg* rather than *vars.cfg*:

```
python globalnews.py –g
```

Currently, GNE pre-processing is limited to the use of rasters in ESRI binary grid format, and a "geographic" (latitude-longitude) projection is assumed. The cell resolution and grid extent must be identical across all input grids and match those of the basin definition grid (for global modeling, this is currently STN-30p vers. 6.01, at 0.5-degree resolution). Basin means and sums are calculated using cell-area scaling that accounts for the decrease in actual surface area (km^2) from the equator to the poles covered by a 1-degree x 1-degree square, due to the (approximately) spherical shape of the Earth. This area scaling uses a pre-existing grid matching the extent and cell resolution of the basin definition grid, where each cell stores the actual cell area (km^2) value.

The GNE GIS pre-processing can perform an additional cell scaling based on the land fraction (or %) of each cell, or other cell fractions (e.g., the fraction of cells that is in a particular land use class); see Fig. 2. Computed basin-scale variables can also be combined through simple algebraic statements to create new basin variables by writing corresponding Python code in the optional file *gis2tbls.py*; this is the only element of the GNE GIS pre-processing tools that may involve user-customized Python code. All other, standard elements of GNE GIS pre-processing are controlled exclusively through the configuration file.

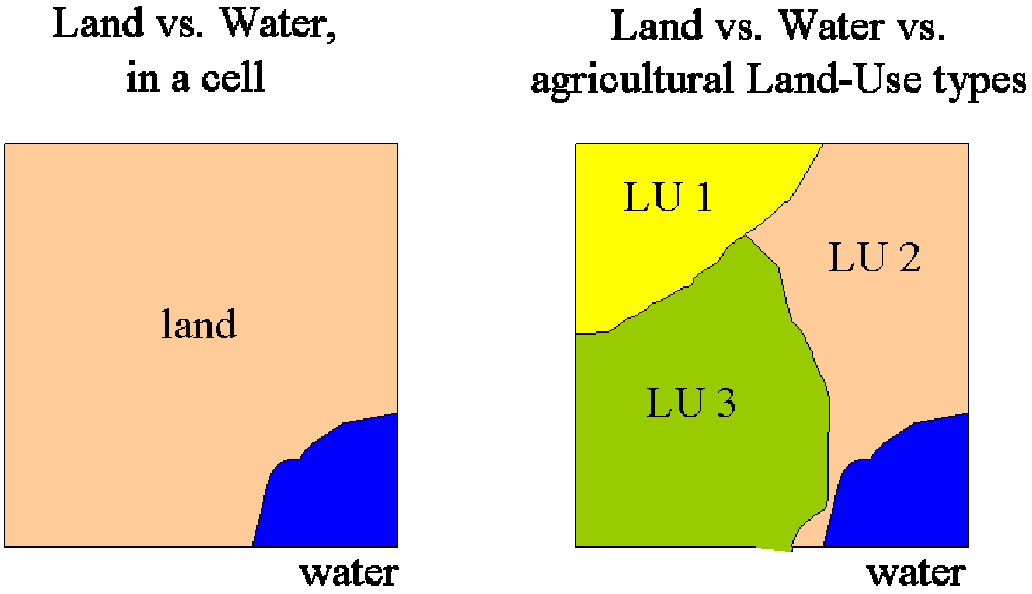

Figure 2. Representation of cell areas, cell sub-areas and land-use ("LU") classes. In MNP/IMAGE convention, land portion of cell is always either 100% agricultural or 100% natural.

Finally, it should be noted that some pre-processing steps involve more complex or heterogeneous basin aggregations schemes; an example of such a step is the reservoir retention factor calculation. Another example may include the calculation of basin averages from state- or province-level values, an operation best performed in spreadsheet software like Excel, rather than as GIS raster operations. Pre-processing of such variables to tables of basin-aggregated values must be performed by the user outside of GNE, using any approach and software that are deemed appropriate.

GNE cannot convert units on demand. A model code version will have a specific set of expected units for each input variable. All basin inputs should be in the units expected by the Global NEWS model (see corresponding GNE configuration files or related documentation for input variables); the alternative would be to modify the model code to perform on-the-fly unit conversion from the units used with input variables, but this can be more complicated and can lead to confusion through excessive customization of the model code.

Post-processing. Finally, GNE includes a "post-processor" option that can be run after a model run is completed. Currently this functionality is used to calculate a set of basin input and output variables as loads (mass/time) by basin, creating a new table. This functionality is also configured through *vars.cfg*, but the code performing the post-processing step is found in *postprocess.py*. It is requested through the optional argument "–p":

```
python globalnews.py –p
```

Other post-processing steps carried out for the MA Scenarios project (specifically, regional summaries and global map images) were done outside of GNE, but also using Python and NumPy as well as additional Python add-ons (matplotlib) and external software (ImageMagick); these are not documented in this manual. Additional information about GNE post-processing can be found in this manual under **Major Components > GNE post-processing**.

Basin-scale input and output tables. GNE uses *csv* (ASCII files using commas as the delimiter between columns) as the standard file format for tables of basin-scale inputs and outputs. See the *csv* **Glossary** entry for more details on this widely used, simple format, as implemented in GNE.

Major Components

The components of a GNE model run are all the files and their organization into sub-folders that make up a complete model run. They include all basin-aggregated *csv* input tables, homogenized rasters used with GNE GIS pre-processing to create basin-aggregated input tables, model code, GNE low-level code, GNE model code, configuration files, *csv* model output tables, model run log files, etc. These are grouped into three major components: Folder organization, Python code, and Configuration files. Each of these major components is described in detail below.

Folder organization

A model run should be organized within a single, master folder that holds all GNE code, homogenized GIS input rasters, pre-processed, basin-aggregated input tables used by the model, and model output tables. The base folder typically holds the GNE code that includes the model code directly; configuration files; and documentation files as needed. The following sub-folder structure is strongly recommended as a convention, but is not strictly required (except as noted):

1. *gnecode*: GNE core files (generally should never be modified, except by GNE Developers). This sub-folder name and its content are mandatory.
2. *inputs*: Pre-processed, homogenized GIS raster data and basin-aggregated model input *csv* tables; the latter are typically

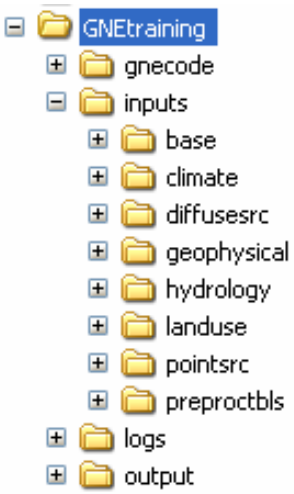

Figure 3. Base folder and sub-folders in a GNE model run.

stored in subfolder *preproctbls*. All basin-aggregated input tables used in the model run should be found within this folder.
3. *output*: Model output *csv* tables.
4. *logs*: Files containing model run-time logs. These log files will only be created if GNE is run with a file redirection (" > logfile") at the command line. For example:
   python globalnews.py –v > logs\c00gnlog.txt

This folder organization is illustrated in Fig. 3. For MA scenarios, an additional sub-folder was created, *preprocessing*. This sub-folder holds most of the code used to process raw GIS source data to create pre-processed, homogenized GIS raster data found under *inputs*; this support code is not part of GNE per-se.

Python code

All GNE Python code files and some optional, supporting code files that are not directly a part of GNE are listed and briefly described below.

1. GNE core ("generic framework") files, in subfolder *gnecode*. Four code files make up the GNE core: *dbf.py*, *gncfg.py*, *gncore.py* and *gngis2tbls.py*. These files should never be modified, except by GNE Developers.
2. Master controller, *globalnews.py*. This is the file that's run at the command line to execute any of the GNE steps (GIS pre-processing, model run, or post-processing). It should never be modified, except by GNE Developers. It has optional arguments for "verbose" message-printing (–v), GIS pre-processing (–g), and GNE post-processing (–p). When *globalnews.py* is called with the help (–h) argument, GNE will print out a brief message and a description of all run-time arguments, perform basic Python configuration checking, then exit without running a model or reading any input data.
3. Model code (Global NEWS model): *dissolved.py* and *particulate.py*. These code files hold all the model or "science" code. They are called by *globalnews.py* at run-time, and can be modified by GNE Model Developers. The GNE core file *gncfg.py* is loaded into the model code files through a standard Python *import* statement; this file includes a set of global-scope GNE variables used within the model code.
4. Optional GNE GIS pre-processing add-on *gis2tbls.py*. Embeds optional, basin-scale algebraic calculations for new variables derived directly from pre-processed variables and output to tables as basin-scale variables.
5. GNE post-processing code, *postprocess.py*. Executed optionally as a separate step, and only after a model run has been completed. Creates basin tables of input and output variables as loads, in the same units (Mg/yr), for convenient mass-balance comparisons. Customized through *vars.cfg;* variable references found there must be matched by the Python code in *postprocess.py*. Like the model code, *postprocess.py* is called by *globalnews.py* at run-time, and can be modified by GNE Model Developers.
6. *.pyc* files are compiled files created automatically by the Python interpreter when GNE is first called; these files are updated when GNE is run after a Python file has been updated. *pyc* files are managed by Python and should be ignored.
7. Optional supporting code, like *batch_run.py* and *batch_regionsmry_maps.py*. These are custom scripts used to help execute multiple GNE runs, including pre- and post-processing steps. They are not a part of GNE *per se* and will not be described in this manual.

Configuration files

**.cfg* configuration files are simple text files that are edited with a *text* editor such as Notepad (but not MS Word!). They generally follow the MS Windows INI file specification. Sections are separated by *headers* enclosed in square brackets, []; a section may contain one or more *options*, each declared in a separate line using the format "*name = value*". Option names can only use alphanumeric characters or underscores, and can't start with a number. Lines starting with "#" or ";" are comments; comments can also be placed after an option name and value statement. Example:

```
[SECTION_HEADER]
# Section options are listed below
optionname1 = N:\data\table.csv
optionname2 = 4.5     # from here to the end of the line, this is a comment
```

There are four types of configuration files in GNE, with the following required file names: *gensetup.cfg, constants.cfg, vars.cfg* and *gis2tbls.cfg*. The purpose and rules for each of these files are explained below. Samples generally corresponding to the c00 (year 2000) MA scenarios model run are included in the Appendix. File paths, table names and table column names used in input and output datasets are restricted to alphanumeric characters and underscores. Character case is important (e.g, "a" and "A" are not the same). Option values in *vars.cfg* and *gis2tbls.cfg* often include the divider character "|", which separates distinct elements within an option declaration line.

gensetup.cfg defines folder paths referenced in the main configuration files, *vars.cfg* and *gis2tbls.cfg.* All options are defined under a single section, [STRSUBS] ("string substitution"). None of these options are required, and no specific option name is required. The only requirement is that if a name is referenced (enclosed in parenthesis) in *vars.cfg* or *gis2tbls.cfg*, it must be present and defined in *gensetup.cfg*. In the standard MA scenarios runs, the options used are *fpath_in* (folder path for inputs) and *fpath_out* (folder path for outputs). A simple example of an option definition:

```
fpath_in = N:\newsmodel\MAscenarios\NEWSruns\inputs
```

and the use of this string-substitution option in *vars.cfg*:

```
difflanduse = (fpath_in)\preproctbls\c00_diffsrc_landuse_ng.csv|basin
```

constants.cfg assigns values for the calibration constants for each of the dissolved sub-models (by nutrient form). Options (constants) for each form are defined under a section header corresponding to the form name with the prefix "CAL." ("calibration"; [CAL.DIN], [CAL.DIP], etc.). Some of the assigned values represent dummy, default values listed simply to facilitate the use of a common model structure for all dissolved nutrient forms. Specifically, the following constants are default values: *a* for DIN, *c* for DIP, and *c* & *e* for DOC. Constants *a* & *e* are required for all nutrient forms, while *c* is required for all forms except DIN. Some constants are used in only a sub-set of the nutrient forms. Note that the names of these constants cannot be changed arbitrarily, as they're expected by the model code. An example, listing constant values for DIN:

```
[CAL.DIN]
a = 1
e = 0.94
enat = 0.1
```

The particulate sub-models currently do not use this configuration file. Regression coefficients and other constants used in these sub-models are embedded in the Python model code.

vars.cfg defines basin-aggregated inputs and outputs for a Global NEWS GNE run. This file is also used to define GNE model "post-processing", where the overall file structure is largely the same as in model run, but the code that is used with the configuration file is *postprocess.py* rather than *dissolved.py* and *particulate.py*. More details about post-processing will be provided in another section; the discussion here focuses on the use of *vars.cfg* for model runs. *vars.cfg* is divided into three "blocks": Model Run definition, Inputs, and Outputs. These blocks can be listed in any order, but the convention is to define them in the order just listed. The input and output blocks are each divided into a table definition section and a variable definitions section. Within each block, the tables section should be defined before the variables section. The order of tables or variables within a section doesn't impact the operation of a model run.

- Model Run block. Made up of a single section, [MODELRUN], with only two options, *p* (nutrient form, referred to in GNE as "parameter") and *param_ord* (parameter order sequence). *p* is a list of nutrient forms (upper case) separated by commas. The following "parameter groups" (or aliases) are also available for selecting multiple nutrient forms: *all, nitrogen, phosphorus, carbon, inorganic, organic, dissolved* and *particulate* (while the DIC model is still under development, DIC is included in several of these groups); if a group is used, no other group or nutrient form can be listed. *param_ord* is a comma-separated list of indices representing the left-right order in which each nutrient form will be written out in the output *csv* table (or tables); the index values correspond to the relative order of each nutrient form in the following sequence: DIN,DIP,DIC,DON,DOP,DOC,PN,PP,POC.
- Inputs block. Made up of two sections: tables [IN.TBLS] and variable definitions [IN.VARS].
  - *[IN.TBLS] section*. Defines complete file path (folders and file name) and shorthand alias (table reference) for each input data table used. Any number of individual table files can be used, using any reference names, as long as each table reference in the variables section has a match in the tables section. The format used for declaring a table is as follows (where "basin" is a dummy element that's not currently being used, but is still required!):
    <table reference> = <table file path>|basin
    Example: *difflanduse = C:\GNErun\inputs\preproctbls\c00_diffsrc_landuse_ng.csv|basin*
    String substitution references from *gensetup.cfg* can be used in the definition of table file paths: *difflanduse = (fpath_in)\preproctbls\c00_diffsrc_landuse_ng.csv|basin*
    - Two core basin attributes are required for a model run: Basin ID and basin area. It is typically convenient and clearer to include these two variables in a single table (for the MA scenarios, this table is called *STN30v6ngNEWS.csv*); however, a user may choose to split the variables into two tables. The conventional table reference for the basin ID's table is *basins*; this table reference name is not required, but it is strongly encouraged and this manual always refers to this table as the *basins* table. This is a critical table that determines the actual basins included in the model run (see *Running a sub-set of basins*, below).
    - All other input tables (including the basin area table, if basin area is not in *basins*) must have a basin ID column matching the ID's used in the *basins* table. The basin ID column on all input tables except *basins* must be named either "basinid" or "BASINID" (all lower or upper case). GNE matches input variables by basin ID as found in each source table.
    - Basins (rows) in each input table *must* be sorted in ascending order by basin ID.
    - Input variables can be grouped in one or many input tables; the grouping makes no difference in GNE, and will typically be a practical decision based on the origin and pre-processing steps required to produce each variable.

- *[IN.VARS] section.* Defines input variables to be used by the model. Information about variables includes table source, column name, and data type (currently integer or floating point [*int* or *double*]). The format used for declaring a variable is as follows:
  <variable reference> = <table reference>|<column name>|<data type>
  Example: *agric = difflanduse|c0agrpct|double*
  This line declares the variable reference *agric*, to be read from table reference *difflanduse*, (declared in section *[IN.TBLS]*), where it's found with column name *c0agrpct*; it will be read and stored in memory as a floating point variable (data type *double*).
  - Variable references are used in the model code and cannot be changed unless the model code is changed, too. GNE Model Users should never modify variable references. GNE Model Developers can modify all variable references except *BasinID*, as long as variable references are maintained in the model code.
  - GNE mandates the *BasinID* variable reference; this variable reads the basin ID column from the *basins* table. Typically the column name will be "BASINID" (all uppercase), but this name is not required.
  - For basin area, the variable reference *A* is recommended and is currently used. However, a GNE Model Developer may change this reference.
- *Rules for names.* Only alphanumeric characters and underscores can be used in table and variable references, file paths and file names, and column names. All names and table or variable declaration elements are case sensitive (e.g, "a" and "A" are not the same).
- *Running a sub-set of basins.* GNE can accept a subset of basins in the *basins* table to select the basins in a model run, even when all other input tables include a larger set of basins. With this sub-setting mechanism, input tables can have different sets of basins, as long as they're in ascending order and the requested basin subset is included in all input tables (e.g., if 1 basin is requested, one input table may have 6000 basins while another only has that 1 basin).

- Outputs block. Like the Inputs block, this block is made up of two sections: tables [OUT.TBLS] and variable definitions [OUT.VARS].
  - *[OUT.TBLS] section.* Defines complete file path (folders and file name) and shorthand alias (table reference) for each output data table used. Any number of individual table files can be used, using any reference names, as long as each table reference in the variables section has a match in the tables section. The format used for declaring a table is identical to the format for input tables, except the dummy "basin" element is not required:
    <table reference> = <table file path>
    As with input tables, string substitution references from *gensetup.cfg* can be used in the definition of table file paths: *outtbl = (fpath_out)\c00_NEWSOutput.csv*
    - For the MA scenarios, the convention was to write all output variables into a single table. However, output variables may be written out into more than one table, where each table holds a subset of variables. Also, a variable may be written out to more than one table.
    - Basins will be written out as rows in ascending order by basin ID. As discussed in the Inputs block, the set of basins that is modeled and exported is based on the list of ID's specified in the *basins* table.
  - *[OUT.VARS] section.* Defines variables that will be output to files. They typically represent model output, but they can represent input as well; the choice of variables *available* for output is made by the GNE Model Developer and is embedded in the model code. Information about variables includes table destination, column name, data type (currently integer [*int*] or floating point [*double*], and a flag stating whether to write out the variable or not (yes [*Y*] or no [*N*]).

The format used for declaring a variable is as follows:
<variable reference> =
<table reference 1>[,<table reference 2>,…]|<column name>|<data type>|<write flag>
Basic example: *TSSyld = outtbl|c0TSSyld|double|Y*
This line declares the variable reference *TSSyld*, to be written out (*Y*) as a floating point variable (*double*) to table reference *outtbl*, (declared in section *[OUT.TBLS]*) with column name *c0TSSyld*.

- As in the Inputs block, variable references are used in the model code and cannot be changed unless the model code is changed, too. GNE Model Users should never modify variable references. GNE Model Developers can modify all variable references except *BasinID*, as long as variable references are maintained in the model code.
- Though not required by GNE, an output basin ID variable should always be included. It must be called *BasinID*, as in [IN.VARS]. By convention, this should be the first variable listed. The recommended output column name (used in the MA scenarios) is "basinid" (all lowercase), but this name is not required. Example:
  BasinID = outtbl|basinid|int|Y
- If the write-flag element is set to N (no), the variable will not be written out to a file. If this element and its preceding "|" divider are left out, a default value of Y (yes) is used.
- A variable can be written out to more than one table by listing multiple table references. The same, single column name is used in all tables. Example, outputting to two tables:
  OutVar = tblref1,tblref2|OutVarColName|double|Y
- A nutrient form ("parameter", in GNE) group facility is provided for output variables to eliminate redundancy in the configuration file and in model code. This is useful because many model variables like nutrient form yield are calculated for all forms or more than one form; in the code, the yield variable name can remain generic (*Pyld*) and be re-used for all forms. The syntax is illustrated with an example: ***<p|all>****Pyld = outtbl|c0<**p**>yld|double*
  where the form grouping components are in bold (the write-flag has been left out, for simplicity). Here, the variable reference "form selector" ***all*** is the "parameter group" (nutrient form group) representing all nutrient forms specified in the *Model Run* block (see the discussion on parameter groups found there and in the glossary); form selectors are specified only within the variable reference. The *Pyld* variable reference component will be expanded by GNE into form-specific variable references, and the column name will be expanded into corresponding, form-specific names. To continue the example above, if DIN and DON are the only nutrient forms specified in the Model Run block, the variable declaration will be expanded into two, specific variable declarations:
  DINPyld = outtbl|c0DINyld|double
  DONPyld = outtbl|c0DONyld|double
  The angle brackets <> are part of the form-group facility syntax. In the variable reference, the form-group component must be placed at the beginning; in the column name element, this component may be found anywhere (start, middle, or end). The "form selector" can be a single form, a list of forms, or a form group, as illustrated in the following examples:
  <p|DIN>*FEwsnat = outtbl|c0<**p**>FEwsnat|double*
  <p|DIP,DOP>*srcAntSewDet = outtbl|c0<**p**>srcAntSewDet|double*
  <p|dissolved>*FEriv = outtbl|c0<**p**>FEriv|double*

        - The left-to-right order of columns used in the output csv file(s) reflects the top-to-bottom listing order for output variables, together with the *parameter* (nutrient form) order list specified in the Model Run block (see the nutrient form group facility description, above).
    - *Rules for names.* Only alphanumeric characters and underscores can be used in table and variable references, file paths and file names, and column names. All names and table or variable declaration elements are case sensitive (e.g, "a" and "A" are not the same).

gis2tbls.cfg defines pre-processed GIS raster and table files, cell area scaling, and resulting basin-aggregated tables for use as inputs in a Global NEWS GNE run. It shares much of the syntax, rules and structure used in *vars.cfg* to define a model run. However, unlike a model run that involves Python model code tuned to a specific *vars.cfg* configuration, typical GNE GIS pre-processing steps involve no user-customized Python coding. The conversion of raster data into basin-aggregated values is performed through a set of standardized procedures that are fully specified through the *gis2tbls.cfg* file and are built into the GNE core (except for the optional use of additional, custom algebraic manipulations in *gis2tbls.py*, discussed later). Like *vars.cfg*, this configuration file is divided into three blocks: Create Basin Areas, Inputs, and Outputs. These can be listed in any order, but the convention is to define them in the order listed above. *Inputs* and *Outputs* define corresponding files and variables. The key, distinctive feature of this configuration file is that the variable reference name of a basin-aggregated variable declared in the *Outputs* block must match the variable reference name of an *Inputs* variable defining a raster source and processing. The Inputs and Outputs blocks are each divided into a file (or tables) definition section and a variable definitions section. Within each block, the files (or tables) section should be defined before the variables section. The order of files or variables within a section doesn't impact the operation of the GIS pre-processing facility.

- Create Basin Areas block. This block enables or disables a special facility for calculating a table of basin areas aggregated from a cell-area grid. It contains one section, [CREATEBASAREAS], with just one, mandatory variable: *BasinAreas*. Typically, reliable basin areas already exist or are created only once at the start of a set of model runs. In such cases, when only standard creation of basin-aggregated values from gridded data (the core functionality provided by GNE GIS pre-processing) is requested, set *BasinAreas = NO*. When the basin-areas facility is requested, the control variable is set as: *BasinAreas* = YES (but note that any text string other than NO will be interpreted as YES). For the [IN.FILES] section, the *tblareas* table definition is not needed or required when using the *BasinAreas* functionality. Also, *grdarea* is expected but never actually used. *area* (full cell area) and *arealand* (area of land portion of cell) an be processed together.
- Inputs block. Made up of two sections, table and grid (raster) files, [IN.FILES], and GIS variables, [IN.GISVARS].
    - *[IN.FILES] section*. Defines complete file path (folders and file name) and shorthand alias (file or grid reference) for each input table or grid used. Any number of individual grid files can be used, using any reference names, as long as each file reference in the variables section has a match in the set of grids declared in the files section. The format used for file declarations is similar to the table declaration format used in *vars.cfg* (see that discussion); however, the element text at the end should be "grid" or "table" (but as in *vars.cfg*, this is a dummy element not currently being used, but still required!).
        - Four core basin grid and table definitions must be present, preferably at the start of the section; the file reference names must be exactly *tblbasins*, *tblareas, grdbasins* and *grdarea*. *tblbasins* and *tblareas* refer to two or a single csv table of basin-scale values (basin ID's and cell-based basin area, respectively). The basin ID column in

tblbasins and *tblareas* must be named "BASINID" (all uppercase). The grids referred to by *grdbasins* and *grdarea* contain cell values for basin membership (ID) and cell area (or cell *land* area), and should typically be stored in the *inputs\base* folder.

- *tblbasins* determines the actual basins that will be processed and exported to basin-aggregated tables. As with *basins* in *vars.cfg*, a subset of basins can be selected through *tblbasins* as long as that subset is found within the *grdbasins* grid.
- *Cell and basin areas. tblareas* lists basin areas (in column "area", all lower case; in km^2) used in the area-weighted averaging (or summation) of gridded input variables. For the MA scenarios, this basin area always represents the whole-cell-based basin area, (possibly calculated earlier through the Create Basin Areas facility); it's strongly recommended that this approach be used. *grdarea* may contain cell area values (km^2) representing full-cell areas or the area of the land portion of a cell (i.e., excluding open water portions). Note that I wrote non-GNE Python code for creating global cell-area grids for any lat-lon cell size; these grids are then used (possibly after additional processing) as *grdarea*.
- *Cell fraction grids.* These are a special set of grids that are used to perform additional cell-area scalings on user-selected grids. Their grid reference names must begin with the string "clfr", and the declaration lines should typically be placed right after the declaration of the four core file references. Cell fraction grid references are *never* used as the grid reference in a GIS input variable declaration. Cell values in these grids represent the fraction of the cell (or of the land-portion of the cell) covered by a land-use type, or some other landscape category such as natural areas, agricultural areas, land, open water, etc. If cell values are fractions (0-1), the units of affected grids will not be modified; but if cell values are percentages (%), the units of affected grids will be multiplied by 100 – a combination that was often used in the MA scenarios with MNP grids (e.g., fertilizers in kg/ha/yr).
- All other file references declare grids that will be processed into basin-aggregated values. For clarity, the grid reference name should end with the string "grd".

- *[IN.GISVARS] section.* Defines GIS (grid) input variables and variable-specific cell-area scaling options to be used for calculating basin-aggregated values. Information about the variable includes grid reference source, grid sub-dataset, and cell fraction grid:
  <variable reference> = <grid reference>|<grid sub-dataset>|<cell fraction grid>
  where grid sub-dataset is just the term "VALUE" (all uppercase) in most cases, but can also be a grid VAT field name (more details below); and cell fraction grid is the term "all" when no additional cell-fraction scaling is requested, or the cell fraction grid reference (see above) to apply an additional scaling. Example: *nfele = nfelegrd|VALUE|clfrle*
  where *nfelegrd* is the grid reference for the legume N fertilizer application grid (defined as a rate [kg N/ha/yr] over the *legume* land-use portion of the *land* within the cell), and *clfrle* is the cell fraction grid (% of the land portion of the cell cultivated with legumes).
  - Unlike *vars.cfg*, there's no *BasinID* input variable declaration in *gis2tbls.cfg*.
  - A general cell-area weighing defined by *grdarea* and *tblareas* is applied to all input grids that will be processed into basin-scale values (see *Cell and basin areas* above).
  - The grid sub-dataset element may be used with categorical ESRI binary grids having a "value attribute table" (VAT) with additional variables. Such grids

typically represent administrative units such as country, province, county, etc. A list of variables such as GDP and % connected to sewage systems can be directly attached to each through the VAT.

- Outputs block. Like the Inputs block, this block is made up of two sections: tables [OUT.TBLS] and variable definitions [OUT.VARS]. Similar to a model run (*vars.cfg*), outputs correspond to *csv* tables where each column is a variable, including a basin ID column. This is in contrast to the Inputs block, where the files and tables generally represent grids and grid processing steps.
    - *[OUT.TBLS] section*. Defines complete file path (folders and file name) and shorthand alias (table reference) for each output data table used. See the corresponding description in *vars.cfg*, as the function and rules of this section are identical. As in *vars.cfg*, the format used for table declarations is: <*table reference*> = <*table file path*>
      As in *vars.cfg*, a dummy label may be found at the end of the declaration, but this text is currently not used or required.
    - *[OUT.VARS] section*. Defines variables that will be output to files. See the corresponding description in *vars.cfg*, as the function and rules of this section are nearly identical, except as noted below.
        - In OUT.VARS, The match between an output (OUT.VARS) variable reference name and an input (IN.GISVARS) variable reference is used to declare that the basin-aggregated output variable is derived from the corresponding input GIS grid.
        - Export basin ID variable must be called *BasinID*.
        - If an input grid doesn't have valid values over a *grdbasins* basin, the basin will be assigned a default "no-data" value of 0.0 for that input grid variable.
        - The nutrient form group or selector described in *vars.cfg* (one of the main features of OUT.VARS in *vars.cfg*) is not available; generally there's no need for it in this context, as input grids are typically somewhat generic.
        - The <data type> element is ignored, but should be included (GNE checks for its presence). *BasinID* is always exported to integer values, while all other variables are always exported to floating-point values.
        - The <*write flag*> is available, but there is never a reason to set it to N (otherwise why bother processing the grid, since it's time consuming?!). It's best to omit the element so that it defaults to Y.
        - The optional Python GNE file *gis2tbls.py* can be used to embed basin-scale algebraic calculations for new variables derived directly from pre-processed variables and output to tables as basin-scale variables. In *gis2tbls.cfg*, output variable references that don't have matching input variable references are assumed to be handled in *gis2tbls.py*, through Python code that can be customized by a Model Developer.

GNE post-processing

See some descriptions of the GNE post-processor in sub-sections "Overview of capabilities and functionality" (in **Introduction and Overview**) and "Python code" (in **Major Components**). Note that the listing of variables in the inputs variable section is highly verbose when referring to model output with a nutrient form name, unlike the listing in output variables which can use the form-selector facility (see *vars.cfg*). This is because a form-selector facility has not been implemented for input variables.

Like a model run, the GNE post-processor also relies on a configuration file named *vars.cfg*. This file has a syntax, structure and rules that are identical to *vars.cfg* used in a model run. However, it is customized to fully match the code found in *postprocess.py*. As with model runs, input and output variable reference names must not be modified without corresponding modification to *postprocess.py*. When post-processing is performed after a model run, users should be careful to retain an intact copy of each configuration file; the dependence on a single file name (*vars.cfg*) can lead to inadvertent overwriting of configuration files.

Installation

Python

GNE requires a basic Python installation (version. 2.4 or higher) and the "NumPy" (Numerical Python) module. For Microsoft Windows I use and recommend the ActivePython distribution. Copies of these installation files are found in the Global NEWS password-protected web site, at the URL: http://www.marine.rutgers.edu/globalnews/data/GNE/python/

The GNE component that does "GIS pre-processing" (takes GIS rasters and creates basin properties) relies on ArcGIS geoprocessing. It won't work if ArcGIS is not installed.

If ArcGIS is not installed, use Python 2.5 and its NumPy module, numpy-1.0.2.win32-py2.5.exe. Just double click on the Python installation first, accept all defaults, then double click on the NumPy file and accept all defaults. Newer versions of Python and NumPy should be ok, too, but are untested; however, be very careful with Python 3.0 (once it comes out, probably in Fall 2008), as it will introduce some substantially new behavior that will need to be tested first.

Python configuration and ArcGIS

If ArcGIS is installed, check the version. If it's 9.2, Python 2.4 should be already installed. Just download and install numpy-1.0.2.win32-py2.4.exe. If a version of ArcGIS older than 9.2 is installed, it's much more complicated; contact Emilio for advice. Also, there have been cases where the Python installation folder created by ArcGIS (e.g., C:\Python24) was not added to the PATH Windows environment variable. This is probably a rare circumstance, but in such cases, the Python interpreter (python.exe) will not be accessible by default; contact Emilio for help (I will add instructions for dealing with this situation in a later version of this manual).

Finally, it's important to keep in mind that if ArcGIS is upgraded on your system in the future, and this upgrade installs a newer version of Python or sets up a Python installation as the new default, GNE may no longer work; this will be especially true if NumPy is not installed.

GNE

Once Python and NumPy are installed, GNE can be installed by simply copying all files corresponding to a model run (python files, configuration files, etc.) to a new folder. *Avoid the use of blank spaces or characters other than underscores and alphanumeric characters anywhere in the folder path, as these may cause complications!* A complete, self-contained GNE training model-run folder has been set up, based on the year 2000 inputs for the Millennium Ecosystem Scenarios. A zip file containing these files (*GNEtraining_<datetime stamp>.zip*) can be downloaded from the Global NEWS password-protected web site, http://www.marine.rutgers.edu/globalnews/data/GNE/

This zip file will be decompressed into a single folder called *GNEtraining*. GNE and the overall Python environment can then be tested by opening the Command Prompt window, changing directory to the *GNEtraining* folder, and running GNE with the "help" argument (–h):

```
python globalnews.py –h
```

A message listing the run-time arguments available for GNE will be displayed on screen, and GNE will exit without running a model.

GNE has been tested on Windows 2000, XP, and Vista. As Python runs on multiple operating systems, GNE should be able to run with only minor modifications under Linux and Mac OS X, but has not been tested in those operating systems. However, the GNE GIS pre-processing facility will only run on Windows, as it depend on ArcGIS, which is only available on Windows.

Additional installation details and potential caveats will be added in a future version of this manual.

Usage, Conventions, Modeling Steps, and Demo

Review of modeling steps and command line

As described in **Overview of capabilities and functionality**, a GNE model run involves a number of steps (Fig. 1). The first step is typically the selection of a base folder where the run will be carried out and all pre-processed inputs and model outputs stored (though inputs from an existing run can be re-used). Inputs may need to be modified "manually" (outside of GNE) before they're ready to be pre-processed with GNE into basin-aggregated values. For example, file names may be modified into more uniform names across variables, or unit conversions may be needed. Homogenized GIS raster files should ideally be stored within the model run folder, as done in the MA scenarios and the training sample run. In addition, certain raster inputs may need to be pre-processed manually into basin-aggregated values; this is the case when highly customized processing is required, as is the case for reservoir retention variables.

Please refer to **Introduction and Overview > Overview of capabilities and functionality > Model run** for additional instructions, especially about GNE execution at the command line. **Major Components > Python code** provides information about obtaining help on optional run-time arguments to *globalnews.py*. **Major Components > Configuration files** provides a detailed description of the configuration files.

GNE execution is carried out at the Command Prompt window. A shortcut to the Command Prompt application is typically found under Start > All Programs > Accessories. This application is the descendant of the old DOS, and the file system commands and syntax are generally the same. Two common tasks involving file system navigation are:

- Change to a different drive. Type the drive letter followed by colon; eg, `N:`
- Change to a different directory, typically the model-run base folder. Use the `cd` command

General guideline and tips

This user manual has described some of the flexibilities and constraints provided by GNE. Like any modeling environment, the use of documented conventions for file names and organization strengthens the robustness and traceability of a model run. Some of the conventions used in the Global NEWS Millennium Ecosystem (MA) Scenarios have already been described in this document.

It is important to keep in mind that every time a new GNE model run or pre/post-processing is performed, the output files specified in the configuration files will be overwritten if they already exist. If the intent is to create a new, distinct output file (e.g., as a result of a model experiment), a new output file name should be specified in *vars.cfg*.

For any model run, it is very important to preserve intact all the files involved in the run: input files, configuration files, code, and output files. When file names, content, and locations remain unchanged after a model run is completed, GNE will provide a high degree of traceability from GIS inputs to basin-scale model output and post-processed files.

As GNE relies on the fixed configuration file name *vars.cfg* for model runs and post-processing, it is strongly suggested that a master copy of each unique configuration file is saved (archived) under a name other than *vars.cfg*. In the MA scenarios, the naming convention used was *vars_model.cfg* and *vars_postprocess.cfg*. This issue is also relevant for GNE GIS pre-processing because a fixed configuration file name (*gis2tbls.cfg*) is also used, while it is often convenient or necessary to break up this pre-processing into separate subsets; for the MA scenarios, two GNE GIS pre-processing subsets were used (per year-scenario), and the names for the master copies of the configuration files were *gis2tbls_diffsrc.cfg* and *gis2tbls_hydro_pnt_otherdiffsrc.cfg*.

Demo

A GNE demonstration will be provided in the training sessions, and will be added to this manual in a later version.

Model experiments

The information provided in this manual may be used to set up simple model "experiments" based on the existing MA scenarios. For example, one experiment may involve the use of future nutrient inputs with contemporary climate. Such an experiment can be set up by simply modifying one or more configuration file – typically just *gensetup.cfg* and *vars.cfg*. Note that in such experiments, input variables that previously were found in one table may now originate in two or more tables; new table references may need to be declared.

The use of a two-character code like "x1" (experiment 1) has been found useful for identifying the configuration files and output (in output file and *csv* column names) corresponding to these experiments. As discussed above, a copy of the model configuration file should be preserved (e.g., *x1_vars_model.cfg*). Documentation briefly describing the model experiment and corresponding configuration and output files (e.g., *x1_NEWSOutput.csv*) is also very useful for future reference.

Appendix

Syntax used in this manual

Describe the text syntax and conventions used in this manual to refer to code, optional arguments, lines in configuration files, etc. **This will be completed in a future version of the manual.**

Glossary

Briefly explain common abbreviations, file formats, etc.

- **csv**. File with comma-separated values. Standard text (ASCII) file format used in GNE for GIS pre-processing outputs, and inputs and outputs for a model run and the post-processor. Variables (columns or fields) are delimited by commas. GNE uses the default, "excel" *csv* dialect: blank spaces immediately following a delimiter will be ignored; numeric values are typically not surrounded by quotes (but quotes will be stripped out automatically); and the line termination characters used are based on the convention followed by ASCII files in Windows ("carriage-return and line-feed", \r\n). *Csv* files can be opened in Excel (in Windows) typically by just double-clicking the files, or opened through the menus or toolbar buttons. They will behave in Excel just like a native Excel file, but in order to preserve changes such as formulas, the file must be saved as an Excel file.
- **form selector.** Used in *vars.cfg* to specify output variables using a short-hand for multiple nutrient forms.
- **INI file format.** The syntax or format generally followed by GNE configuration files. INI files are also called configuration files, and sometimes use the extension .cfg rather than .INI. They are text files with a structure originally defined by the MS Windows INI file specification, widely used for configuring software on MS Windows; however, note that there are variants to this format. See http://en.wikipedia.org/wiki/INI_file for generic details.
- **parameter**. The GNE name for nutrient forms (DIN, DIP, etc.), often referenced with the shorthand name "p".
- **parameter group**. An alias for multiple nutrient forms (GNE "parameters"), corresponding to elements, inorganic vs. organic, and size fraction. The following groups are currently defined: *all, nitrogen, phosphorus, carbon, inorganic, organic, dissolved, particulate*. Note that DIC is included among these groups (in *all, carbon, inorganic*, and *dissolved*), even though the DIC sub-model is not yet implemented operationally.
- **Table or file reference**. (in *vars.cfg* and *gis2tbls.cfg*)
- **Variable reference**. (in *vars.cfg* and *gis2tbls.cfg*)

Error messages

Common errors and corresponding error messages. Currently, these will be limited to errors in the *vars.cfg* file. It is assumed that "verbose" error reporting is being used.

- **Input table does not exist (file path is incorrect)**. Python will report a *Traceback* error specifying the code file and line where the error occurred, and ending with: IOError: [Errno 2] No such file or directory, followed by the full file path of the input csv table that could not be found. GNE reports a similar error (written to the log file or the screen): Table does not exist, followed by the full file path of the input csv table.
- **Incorrect column name on input table.** Python will report a *Traceback* error specifying the code file and line where the error was first encountered; the actual error message will typically read:

ValueError: shape mismatch: objects cannot be broadcast to a single shape, because GNE did not fail explicitly when the column name could not be found and the data read (variable was created and populated with one value, 0), but only when the variable is first used in a model equation in combination with other, full-length variable. GNE run-time messages include a section that prints out each variable reference name followed by the number of basins (length of variable, in parenthesis), then the first few values of the variable (in square brackets); this section begins right after the line *** Done reading variable configurations (PopulateCfgVars()) ***. When the variable is not found, the number of basins is 0 and no variable values are listed. When the *BasinID* column name is incorrect, Python will report a *Traceback* error that ends in KeyError:, followed by the specified column name.

Brief documentation for GNE Model Developers

This list is currently only a set of *very* concise notes. It will be used as the starting point to develop more consistent and complete documentation for Model Developers in the future.

- List all forbidden/reserved variable names (i.e., those used in GNE that are global in scope or imported into model code)
- IN and OUT are the core global dictionary variables used for reading and writing variables specified in the configuration files; each dictionary value is a NumPy array of all basins for a particular variable, which corresponds to the dictionary key
- Typically, the function ExportVar() is used to "save" an internal variable for writing to a file (as defined in the configuration file); a model developer will rarely use the OUT variable directly. For variables that are not requested to be written to a file, ExportVar() can also be used as a mechanism for run-time storage of global-scope variables (through the OUT dictionary), for re-use in other functions (see variables used this way in SourceContrib()).
- Input/output variables must have identical names in the configuration files (where they're referred to as "option" names) and in the key to the IN and OUT dictionaries (for OUT, typically this will be through the variable name passed to ExportVar()).
- For output, OUT['BasinID'] is created automatically by GNE; ExportVar() doesn't have to be called from the model code
- The GNE Core file *gncfg.py* defines a set of variables related to nutrient forms ("parameters") that are useful when writing model code. These are PARAMETERS, a list of nutrient form names such as 'DIN' and 'POC'; and PGRP, a dictionary containing several lists of nutrient forms, such as 'nitrogen' ('DIN','DON','PN'). Since *gncfg.py* is imported into each model code files, these variables are readily accessible.
- The run_parameters argument passed to the model() function in dissolved.py and particulate.py is the list of nutrient form ("parameter"; form names as strings) that were requested to be run through the configuration file *vars.cfg*.
- NumPy should be imported into each model code script file. See the import statements in *dissolved.py* for an example.
- For output variables in a model run (as defined in vars.cfg): "If variable is actually a literal (single value) and not a full-length NumPy array, expand it to full-length float32 NumPy array using IN['BasinID'] to extract the length" (extracted from description of ExportVar() function).
- For BasinAreas GIS pre-processing functionality, the table definition *tblareas* is not used. Note that this table definition is hard-wired into function gngis2tbls.py/LoadBasinIDArea()

Python online resources

General Python sites (and source of installation files)

- *Python*: http://python.org/
- *ActivePython*, a Windows Python Distribution: http://activestate.com/Products/activepython/
- *NumPy*. Numerical, array-oriented Python module used extensively in GNE. See the NumPy documentation and tutorials found here. http://www.scipy.org/NumPy

Good sites explaining the strengths of Python in general and for scientists in particular, or providing tutorials and training materials

- *"Official" Python Tutorial*. http://docs.python.org/tut/
- *Instant Python.* http://hetland.org/writing/instant-python.html
- *Python 101 -- Introduction to Python.* http://www.rexx.com/~dkuhlman/python_101/python_101.html
- *Dive into Python*. http://diveintopython.org/
- *NumPy arrays documentation.* Not from the NumPy web site per, but very nice. http://www.hjcb.nl/python/Arrays.html
- *Why I think Python is good (as in granola) for the scientist who programs.* Good blog posting about Python in general, with brief mention of NumPy as a key strength of Python for scientists. http://boscoh.com/science/why-i-think-python-is-good-as-in-granola-for-the-scientist-who-programs
- *Python for Climate Science.* http://geosci.uchicago.edu/~rtp1/itr/PythonLectures/PyLecHome.html
- *Bryan Lawrence's "Why Python?".* http://home.badc.rl.ac.uk/lawrence/blog/2004/12/16/WhyPython
- *Python, all a scientist needs.* http://openwetware.org/wiki/Julius_B._Lucks/Projects/Python_All_A_Scientist_Needs
- *Python: Batteries Included.* Special issue of Computing in Science and Engineering, May/June 2007 (Vol. 9, No. 3). http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/cs/&toc=comp/mags/cs/2007/03/c3toc.xml
- *Software Carpentry*. Basic software development practices for scientists and engineers, using Python as the main language to illustrate general concepts. http://www.swc.scipy.org/

Sample configuration files

gensetup.cfg

```
[STRSUBS]
fpath_in    = N:\newsmodel\MAscenarios\NEWSruns\run4\inputs
fpath_out   = N:\newsmodel\MAscenarios\NEWSruns\run4\output
```

constants.cfg

```
# CONFIGURATION FILE FOR CONSTANTS, INCLUDING CALIBRATION PARAMETERS
# The following parameters are not calibrated; they're simple constants (usually, with a value of 1)
# added to facilitate the integration of model structures: CAL.DIP c; CAL.DIN a; CAL.DOC c, e
[CAL.DIN]
a = 1
e = 0.94
enat = 0.1

[CAL.DIP]
Dnat = 26
a = 0.85
b = 2
c = 1
e = 0.29

[CAL.DON]
Dnat = 280
a = 0.95
c = 0.14
e = 0.01

[CAL.DOP]
Dnat = 15
a = 0.95
c = 0.010
e = 0.010

[CAL.DOC]
Dnatdry = 3883
Dnatwet = 12475
a = 0.95
c = 1
e = 1
```

vars.cfg (as used for a model run)

```
# SPECIFY MODEL OR MODEL GROUPS TO RUN. eg: DIN OR DON,DIN OR dissolved OR all OR nitrogen
# More than one group, or mixing of groups with indiv. params., don't work
[MODELRUN]
p = DIN,DIP,DON,DOP,DOC,PN,PP,POC
param_ord = 1,2,3,4,5,6,7,8,9
# for param_ord: (DIN,DIP,DIC,DON,DOP,DOC,PN,PP,POC)
```

[IN.TBLS]
 # This table is invariant with time or scenario
basins = (fpath_in)\preproctbls\STN30v6ngNEWS.csv|basin
 # This one is also fixed, but is not a hydrographic characteristic
geophysical = (fpath_in)\preproctbls\LithologySlope_ng.csv|basin
 # These data sources vary with time and scenario (except for Wpct in hydropntother, which is time-invariant)
 # difflanduse and hydropntother are generated by GNE GIS pre-processing,
 # while reservdisch is compiled semi-manually from non-GNE pre-processing steps
difflanduse = (fpath_in)\preproctbls\c00_diffsrc_landuse_ng.csv|basin
hydropntother = (fpath_in)\preproctbls\c00_hydro_pntsrc_other_ng.csv|basin
reservdisch = (fpath_in)\preproctbls\c00_disch_fqrem_reservret_ng.csv|basin

[IN.VARS]
 # varname = tablesource|fieldname|fieldtype
 # ---- GENERAL BASIN ATTRIBUTES ----
BasinID = basins|BASINID|int
A = basins|area|double

 # ---- OTHER TIME-INVARIANT BASIN PROPERTIES ----
 # Geophysical/geological variables (for TSS model)
frnrslope = geophysical|FournierSlope|double
LiClass = geophysical|LithClass|int
 # Climate zones and wetland coverage
KoppenGrpAperc = basins|KoppenGrpA_prc|double
W = hydropntother|Wpct|double

 # ---- LAND USE AS PERCENTS ----
agric = difflanduse|c0agrpct|double
wetlndrice = difflanduse|c0wrpct|double
marggrass = difflanduse|c0mrgrpct|double

 # ---- HYDROLOGY ----
R = hydropntother|c0R|double
frnrprecp = hydropntother|c0frnrprecp|double
FQrem = reservdisch|c0FQremSclGEMS|double
Ddin = reservdisch|c0Ddin0to1|double
Ddip = reservdisch|c0Ddip0to1|double
Dsed = reservdisch|c0Dsed0to1|double

 # ---- DIFFUSE LAND N & P BALANCE ----
 # Nitrogen
TNfe = difflanduse|c0TNfe|double
TNma = difflanduse|c0TNma|double
TNexp = difflanduse|c0TNex|double
TNfixnat = difflanduse|c0TNfxna|double
TNfixagr = difflanduse|c0TNfxag|double
TNdepnat = hydropntother|c0TNdpna|double
TNdepagr = difflanduse|c0TNdpag|double
 # Phosphorus
TPfe = difflanduse|c0TPfe|double
TPma = difflanduse|c0TPma|double

```
TPexp      = difflanduse|c0TPex|double

# ---- POINT SOURCES (N & P) ----
TNsewhum = hydropntother|c0TNsewhum|double
TNfrem     = hydropntother|c0TNfrem|double
TPsewhum = hydropntother|c0TPsewhum|double
TPsewdet   = hydropntother|c0TPsewdet|double

[OUT.TBLS]
outtbl = (fpath_out)\c00_NEWSOutput.csv|basin

[OUT.VARS]
# Out variables for all models
BasinID = outtbl|basinid|int|Y

# OUTPUT YIELDS AND LOADS FOR ALL N, P & C FORMS (p = "PARAMETER", nutrient form)
<p|all>Pyld  = outtbl|c0<p>yld|double|Y
<p|all>Pload = outtbl|c0<p>load|double|Y

# PARTICULATES
<p|particulate>Ppct = outtbl|c0<p>pct|double|Y
TSSyld  = outtbl|c0TSSyld|double|Y
TSSload = outtbl|c0TSSload|double|Y
TSSc    = outtbl|c0TSSconc|double|Y

# DISSOLVED
<p|dissolved>Tsew  = outtbl|c0<p>Tsew|double|Y
<p|dissolved>Tnat  = outtbl|c0<p>Tnat|double|Y
<p|dissolved>Tant  = outtbl|c0<p>Tant|double|Y
<p|dissolved>FEpnt = outtbl|c0<p>FEpnt|double|Y
<p|dissolved>FEws  = outtbl|c0<p>FEws|double|Y
<p|DIN>FEwsnat     = outtbl|c0<p>FEwsnat|double|Y
<p|dissolved>FEriv = outtbl|c0<p>FEriv|double|Y

# coarse-scale (Level 1) source attribution (all dissolved forms)
<p|DIN,DIP,DON,DOP>srcMax1   = outtbl|c0<p>srcMax1|double|Y
<p|DIN,DIP,DON,DOP>srcAntSew = outtbl|c0<p>srcAntSew|double|Y
<p|DIN,DIP,DON,DOP>srcAntDif = outtbl|c0<p>srcAntDif|double|Y
<p|DIN,DIP,DON,DOP>srcNatDif = outtbl|c0<p>srcNatDif|double|Y

# fine-scale (Level 2) source attribution (often form-specific)
<p|dissolved>srcMax2           = outtbl|c0<p>srcMax2|double|Y
<p|DIN,DIP,DON,DOP>srcAntSewHum = outtbl|c0<p>srcAntSewHum|double|Y
<p|DIP,DOP>srcAntSewDet        = outtbl|c0<p>srcAntSewDet|double|Y
<p|DIN,DIP,DON,DOP>srcAntMan    = outtbl|c0<p>srcAntMan|double|Y
<p|DIN,DIP,DON,DOP>srcAntFer    = outtbl|c0<p>srcAntFer|double|Y
#   source apportionment specific to DIN
<p|DIN>srcAntFix = outtbl|c0<p>srcAntFix|double|Y
<p|DIN>srcAntDep = outtbl|c0<p>srcAntDep|double|Y
<p|DIN>srcNatFix = outtbl|c0<p>srcNatFix|double|Y
<p|DIN>srcNatDep = outtbl|c0<p>srcNatDep|double|Y
```

```
#   source apportionment specific to DIP
<p|DIP>srcAntWth = outtbl|c0<p>srcAntWth|double|Y
<p|DIP>srcNatWth = outtbl|c0<p>srcNatWth|double|Y
#   source apportionment specific to DON, DOP
<p|DON,DOP>srcAntLch = outtbl|c0<p>srcAntLch|double|Y
<p|DON,DOP>srcNatLch = outtbl|c0<p>srcNatLch|double|Y
#   source apportionment specific to DOC
<p|DOC>srcWetlnd = outtbl|c0<p>srcWetlnd|double|Y
<p|DOC>srcNonwet = outtbl|c0<p>srcNonwet|double|Y
```

gis2tbls.cfg (as used for processing most diffuse source grids)

```
[CREATEBASAREAS]
BasinAreas = NO

[IN.FILES]
# base table, basin area table, base grid, cell area grid, cell fraction grids, & input grids
# tblareas can't have any zero values.
# CORE BASIN GRIDS AND TABLES
tblbasins  = (fpath_in)\preproctbls\STN30v6ngNEWS.csv|table
tblareas   = (fpath_in)\ preproctbls\STN30v6ngNEWS.csv|table
grdbasins = (fpath_in)\base\g_basin_noice|grid
grdarea    = (fpath_in)\base\arealand|grid

# CELL FRACTION GRIDS (AS PERCENTAGES)
clfrag    = (fpath_in)\landuse\c00agrc|grid
clfrna    = (fpath_in)\landuse\c00nat|grid

# LAND USE GRIDS (AS PERCENTAGES)
agricfr   = (fpath_in)\landuse\c00agrc|grid
wricefr   = (fpath_in)\landuse\c00wr|grid
margrfr   = (fpath_in)\landuse\c00mg|grid

# DIFFUSE SOURCE GRIDS - FERTILIZER, MANURE, CROP EXPORTS
# (kg ha-1 yr-1), N & P
#     nitrogen
nfegrd    = (fpath_in)\diffusesrc\c00nfe|grid
nmagrd    = (fpath_in)\diffusesrc\c00nma|grid
nexgrd    = (fpath_in)\diffusesrc\c00nex|grid
#     phosphorus
pfegrd    = (fpath_in)\diffusesrc\c00pfe|grid
pmagrd    = (fpath_in)\diffusesrc\c00pma|grid
pexgrd    = (fpath_in)\diffusesrc\c00pex|grid

# DIFFUSE SOURCES - N FIXATION & AGRIC ATM. DEPOSITION (kg ha-1 yr-1)
nfxaggrd = (fpath_in)\diffusesrc\c00nfxag|grid
nfxnagrd = (fpath_in)\diffusesrc\c00nfxna|grid
ndpaggrd = (fpath_in)\diffusesrc\c00ndpag|grid

[IN.GISVARS]
# 2nd item is the grid attribute name (in the VAT).  3rd item is optional cell fraction grid
# DIFFUSE SOURCES - AGRICULTURAL, WETLAND RICE, & MARGINAL GRASS CELL PERCENTAGES
```

```
agric = agricfr|VALUE|all
wrice = wricefr|VALUE|all
margr = margrfr|VALUE|all

# DIFFUSE SOURCES - FERTILIZER, MANURE, CROP EXPORTS, N & P
#     nitrogen
TNfe     = nfegrd|VALUE|clfrag
TNma    = nmagrd|VALUE|clfrag
TNex     = nexgrd|VALUE|clfrag
#     phosphorus
TPfe     = pfegrd|VALUE|clfrag
TPma    = pmagrd|VALUE|clfrag
TPex     = pexgrd|VALUE|clfrag

# DIFFUSE SOURCES - N FIXATION & AGRIC ATM. DEPOSITION
TNfxag  = nfxaggrd|VALUE|clfrag
TNfxna  = nfxnagrd|VALUE|clfrna
TNdpag = ndpaggrd|VALUE|clfrag

[OUT.TBLS]
outputtbl = (fpath_in)\preproctbls\c00_diffsrc_landuse_ng.csv|table

[OUT.VARS]
BasinID = outputtbl|basinid|int

# DIFFUSE SOURCES - AGRICULTURAL, WETLAND RICE, & MARGINAL GRASS CELL PERCENTAGES
agric   = outputtbl|c0agrpct|double
wrice  = outputtbl|c0wrpct|double
margr  = outputtbl|c0mrgrpct|double

# DIFFUSE SOURCES - FERTILIZER, MANURE, CROP EXPORTS, N & P
#     nitrogen
TNfe     = outputtbl|c0TNfe|double
TNma    = outputtbl|c0TNma|double
TNex     = outputtbl|c0TNex|double
#     phosphorus
TPfe     = outputtbl|c0TPfe|double
TPma    = outputtbl|c0TPma|double
TPex     = outputtbl|c0TPex|double

# DIFFUSE SOURCES - N FIXATION & AGRIC ATM. DEPOSITION
TNfxag = outputtbl|c0TNfxag|double
TNfxna = outputtbl|c0TNfxna|double
TNdpag = outputtbl|c0TNdpag|double
```