Documentation for Python programs, sim_q_reach_3d.py, sum_sim_q_reach.py, sim_lakes_3d.py

J.W. Trey Grubbs

This document describes three computer programs, sim_q_reach_3d.py, sum_sim_q_reach.py, and sim_q_lakes_3d.py, that can be used to extract and aggregate simulated boundary condition fluxes from the MODFLOW River and General-Head Boundary Packages. The intended use of these programs is for the development of the North Florida Southeast Georgia (NFSEG) groundwater flow model, and are not intended for use outside of the NFSEG project. The programs are written in the Python programming language and are compatible with versions 2.6 and 2.7 of the Python interpreter on Windows and Linux systems. The Python interpreter can be freely downloaded at www.python.org, and is also preinstalled with ArcGIS so you may already have it on your system. Several utility programs for configuring some of the input files needed to run these flux-extraction programs are also described in the appendix that follows the description of the these programs. Please note the following disclaimer regarding use of these any of these flux-extraction or utility programs:

Please report any errors to Trey Grubbs at the Suwannee River Water Management District (jwg@srwmd.org).

Description of Program, sim_q_reach_3d.py

The purpose of the program, sim_q_reach_3d.py, is to extract and (where necessary) aggregate simulated fluxes from a MODFLOW-NWT output listing file for a set of user-defined river reaches[1] and springs, and write the result to a comma-delimited (plain-text) ASCII file. Please note that the version of the program referenced in this document is intended for use with a MODFLOW-NWT output listing file generated from a simulation with two stress periods, and in which exchanges between rivers, springs and aquifers are represented by the MODFLOW River and General-Head Boundary (GHB) Packages.

Program Input:

Three input files are required to execute the program, sim_q_reach_3d.py:

1. a MODFLOW-NWT output listing file, and
2. a 'gaged-reach definitions file' that associates a unique reach identifier with each river reach or spring in a set of user-defined river reaches and springs
3. a lookup table that associates a unique boundary-condition identifier with the order in which it appears in its corresponding MODFLOW boundary-condition input file

[1] River reaches evaluated with the program, sim_q_reach.py, are typically used to represent either: (1) a reach gaged by the most upstream gage among the set of gages along a river, and (2) a reach bounded by downstream and upstream gages (to evaluate changes in flows along a river reach). Simulated accumulations of flows from a collection of river reaches that contribute to a common downstream gage are evaluated using the program, sum_sim_q_reach.py (described later in this document).

The MODFLOW-NWT output listing file should contain the computed fluxes for the River and GHB Package boundary conditions. This is accomplished by setting the IRIVCB and IGHBCB variables to negative values in the input files for the River and GHB Packages, respectively.

The name of the gaged-reach definitions file is a comma-delimited, ‘plain-text’ file, and should have the name, ‘gaged_reach_definitions.csv’ (without the quotes). The program expects this file to contain one header line (which it actually skips over so the format of this header line is not important), followed by a set of ‘data records’ with the following fields (from left to right):

1. a number or string that uniquely identifies a river reach or spring being simulated[2],
2. a three-digit code indicating the type of boundary condition associated with the sequence number in the previous column: ‘riv’ and ‘ghb’ (without the quotes) are used for River and GHB Package boundary conditions, respectively.
3. An additional unique identifier that can be used to associate the River or GHB Package boundary-condition record (in this row of the gaged-reach definitions file) with a Geographic Information System (GIS) feature. This might be a line feature representing one segment of a river reach or a point feature representing an individual spring.

This gaged-reach definitions file can be created through GIS overlay analyses of the model grid with line and point features that represent the location of river and spring features associated with individual records in the River and GHB Package input files. Note that individual river reaches (defined by their unique identifier in the first field of the gaged-reach definition file) will typically have multiple data records in the gaged-reach definition file because (1) they are represented by multiple boundary condition records in the River Package input file, and (2) they may have one or more springs that occur along the reach.

Please also note that a boundary condition representing a spring is often represented in the gaged-reach definition file twice: once as a spring feature that contributes to the aggregated flow to a river reach (so that the program output for the river reach will include that spring’s contribution to the total flow of the river reach), and another time as an individual spring feature independent of any river reach (so that the program will output a record with the simulated flow for that individual spring).

The lookup table is actually a Python dictionary (called bc_reach_id_dict), contained in a Python ‘shelf’ file (called lookup_bc_reach_ids.shelf). The ‘keys’ for the dictionary, bc_reach_id_dict, are a Python tuple with two elements: a three-letter boundary-condition type identifier (see previous discussion of this identifier) and an integer stress period value. The values corresponding to these keys are Python lists containing a list of values of unique identifiers for a given MODFLOW boundary-condition type. The ordering of the elements in this list is identical to the ordering of records in the corresponding MODFLOW boundary condition input file. For example, if the 4380th MODFLOW River Package input file data record for stress period 2 corresponds to the unique identifier, 4266, for the River Package boundary condition feature in a particular model, then bc_reach_id_dict[('riv',2)][4379] would return a value of 4266 (note that Python, like many computer languages, uses ‘zero-based’ indexing, so we used a value of 4379 to refer to the 4380th element in the list). Embedding this lookup-table in a Python dictionary in a Python shelve file greatly speeds up the execution of the program sim_q_reach_3d.py. This shelve file (lookup_bc_reach_ids.shelf) can be easily created using a ‘batch file’ that calls a set of

[2] Ideally, the width this identifier should be less than twenty characters so that it can be used as part of a corresponding PEST observation name (which can be no wider than twenty characters), if desired.

utility programs that were written for this purpose. Please see the Appendix ('Creation of a Boundary-Condition Feature Lookup Table') for details.

Program Execution:

The program is executed from the command prompt using the following syntax:

```
python sim_q_reach_3d.py [listfile] [optional conversion factor]
```

where

> [listfile] is the name of the MODFLOW-NWT output listing file (for example, nfseg.lst), and [optional conversion factor] is a conversion factor for converting the MODFLOW simulated fluxes from one unit to another. For example, a conversion factor of 1.157407407E-05 (1 day /86400 seconds) should be specified if converting from units of cubic feet per day to cubic feet per second. The conversion factor argument is optional, so it can be omitted if unit conversion isn't needed.

Please note that you'll need to have the Python interpreter installed on your computer for the above command to work. The program was tested using version 2.7 of the Python interpreter, but some of the earlier '2.x' versions of the interpreter will probably work as well. You'll also need to have the PATH environment variable set appropriately so that your command-line shell will know where to find the Python interpreter. This was probably set when Python was installed on your computer. Note that on Unix-based operating systems, this can also be accomplished by inserting a string like, #!/usr/bin/python, at the top of the program.

Program Output:

The program, sim_q_reach_3d.py, creates a spaces-delimited, output file called, gaged_reach_fluxes.asc. This file has one header line, followed by a set of 'data records' containing the following fields (from right to left):

1. **gaged_reach_id**: a number or string that uniquely identifies a river reach or spring being simulated
2. **riv_sim_flux_sp1_ts1**: simulated River Package flux to the feature defined by the value of the field, **gaged_reach_id**, during the first stress period.
3. **ghb_sim_flux_sp1_ts1**: simulated GHB Package flux to the feature defined by the value of the field, **gaged_reach_id**, during the first stress period.
4. **total_sim_flux_sp1_ts1**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **gaged_reach_id**, during the first stress period.
5. **riv_sim_flux_sp2_ts1**: simulated River Package flux to the feature defined by the value of the field, **gaged_reach_id**, during the second stress period.
6. **ghb_sim_flux_sp2_ts1**: simulated GHB Package flux to the feature defined by the value of the field, **gaged_reach_id**, during the second stress period.
7. **total_sim_flux_sp2_ts1**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **gaged_reach_id**, during the second stress period.
8. **del_total_sim_flux**: the difference between **total_sim_flux_sp2_ts1** and **total_sim_flux_sp1_ts1 (i.e. total_sim_flux_sp2_ts1 - total_sim_flux_sp1_ts1).**

<u>Description of program, sum_sim_q_reach.py</u>:

The purpose of the program, sum_sim_q_reach.py, is to aggregate simulated fluxes from collections of river reaches, so that (for example) simulated fluxes from a set of river reaches that contribute flow to a common downstream location can be totaled and compared to the streamflow measured at a downstream gaging station. This program is a companion program to the previously described program, sim_q_reach_3d.py, and assumes that sim_q_reach_3d.py has already been executed and used to generate one of the input files required for successful execution.

Please note that the version of the program referenced in this document is intended for use with a MODFLOW-NWT output listing file generated from a simulation with two stress periods, and in which exchanges between rivers, springs and aquifers are represented by the MODFLOW River and General-Head Boundary (GHB) Packages.

Program Input:

Two input files are required to execute the program, sum_sim_q_reach.py:

1. A comma-delimited, 'plain-text' file called, upstream_gage_numbers.csv.
2. A comma-delimited, 'plain-text' file called, gaged_reach_fluxes.csv.

The file, upstream_gage_numbers.csv, does not have a header line. The first field of each record in this file is a unique identifier for the downstream location on a river that accumulates flows from a set of upstream river reaches. The second field in each record of the file, upstream_gage_numbers.csv, is the name of the location (typically a stream-gaging station) corresponding to the unique identifier in the first field. The fields that follow the second field in a given record correspond to the unique identifiers of each of the river reaches that contribute flow to the downstream location associated with the unique identifier in the first field. The unique identifiers of the fields to the right of the second field are the same identifiers as those used to identify river reaches in the input and output files of the program, sim_q_reach_3d.py (gaged_reach_definitions.csv and gaged_reach_fluxes.csv, respectively).

Program Execution:

The program is executed from the command prompt using the following syntax:

```
python sum_sim_q_reach.py
```

Please note that you'll need to have the Python interpreter installed on your computer for the above command to work. The program was tested using version 2.7 of the Python interpreter, but some of the earlier '2.x' versions of the interpreter will probably work as well. You'll also need to have the PATH environment variable set appropriately so that your command-line shell will know where to find the Python interpreter. This was probably set when Python was installed on your computer. Note that on Unix-based operating systems, this can also be accomplished by inserting a string like, #!/usr/bin/python, at the top of the program.

Program Output:

The program, sum_sim_q_reach.py, creates a comma-delimited, output file called gaged_fluxes_sum_details.csv. This file has one header line, followed by a set of data records with the following fields:

1. **station_id**: a number or string that uniquely identifies a collection of river reaches that contribute flow to a given location (usually identifies the gaging station at such a location)
2. **station_name:** the name of the location corresponding to **station_id**,
3. **riv_sim_flux_sp1_ts1_cfd**: simulated River Package flux to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per day.
4. **ghb_sim_flux_sp1_ts1_cfd**: simulated GHB Package flux to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per day.
5. **total_sim_flux_sp1_ts1_cfd**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per day.
6. **riv_sim_flux_sp2_ts1_cfd**: simulated River Package flux to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per day.
7. **ghb_sim_flux_sp2_ts1_cfd**: simulated GHB Package flux to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per day.
8. **total_sim_flux_sp2_ts1_cfd**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per day.
9. **del_total_sim_flux_cfd**: the difference between **total_sim_flux_sp2_ts1_cfd** and **total_sim_flux_sp1_ts1_cfd (i.e. total_sim_flux_sp2_ts1_cfd - total_sim_flux_sp1_ts1_cfd),** in cubic feet per day.
10. **riv_sim_flux_sp1_ts1_cfs**: simulated River Package flux to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per second.
11. **ghb_sim_flux_sp1_ts1_cfs**: simulated GHB Package flux to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per second.
12. **total_sim_flux_sp1_ts1_cfs**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **station_id**, during the first stress period, in cubic feet per second.
13. **riv_sim_flux_sp2_ts1_cfs**: simulated River Package flux to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per second.
14. **ghb_sim_flux_sp2_ts1_cfs**: simulated GHB Package flux to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per second.
15. **total_sim_flux_sp2_ts1_cfs**: sum of simulated River and GHB Package fluxes to the feature defined by the value of the field, **station_id**, during the second stress period, in cubic feet per second.
16. **del_total_sim_flux_cfs**: the difference between **total_sim_flux_sp2_ts1_cfs** and **total_sim_flux_sp1_ts1_cfs (i.e. total_sim_flux_sp2_ts1_cfs - total_sim_flux_sp1_ts1_cfs),** in cubic feet per second.

Description of program, sim_q_lakes_3d.py

The purpose of the program, sim_q_lakes.py, is to extract and (where necessary) aggregate the simulated fluxes from collections of River Package boundary condition features (and the 'free-surface area of those features) representing individual lakes. Simulated-flux values are obtained from a MODFLOW-NWT output listing file, and output is written to a comma-delimited (plain-text) ASCII file. Simulated flux values output for each lake are expressed in both volume per unit time and length per unit time units. Please note that the version of the program referenced in this document is intended for use with a MODFLOW-NWT output listing file generated from a simulation with two stress periods, and in which exchanges between lakes and aquifers are represented by the MODFLOW River Package.

Program Input:

Three input files are required to execute the program, sim_q_reach_3d.py:

4. a MODFLOW-NWT output listing file, and
5. a 'lake-definitions file' that associates a given River Package stress-period record (i.e. record containing layer, row, column, and stage data for a given stress period) with a lake (or more commonly, part of a lake) represented in a Geographic Information System (GIS) and the surface area of that feature
6. a lookup table that associates a unique boundary-condition identifier with the order in which it appears in its corresponding MODFLOW boundary-condition input file

The MODFLOW-NWT output listing file should contain the computed fluxes for the River Package boundary conditions. This is accomplished by setting the IRIVCB variable to a negative value in the River Package input file.

The name of the lake-definitions file is a comma-delimited, 'plain-text' file, and should have the name, 'lake_definitions.csv' (without the quotes). The program expects this file to contain one header line (which it actually skips over so the format of this header line is not important), followed by a set of 'data records' with the following fields (from left to right):

1. a lake name (this name should not contain any commas)
2. a number that uniquely identifies the lake being simulated[3],
3. a three-digit code indicating the type of boundary condition associated with the sequence number in the previous column. For the NFSEG model, this value should be 'riv' (without the quotes) because lakes are represented with the River Package boundary condition in the NFSEG model.
4. An additional unique identifier that can be used to associate the River Package boundary-condition feature with a '2-dimensional' Geographic Information System (GIS) feature. For example, this could be a GIS point feature representing the centroid of part of the free-water surface of a given lake.
5. The (free-water) surface area of the part of the lake represented with a given MODFLOW River Package boundary condition record. Note that this surface-area value should be expressed in units that are consistent with the underlying MODFLOW model (e.g. if the MODFLOW model is implemented with length units of feet, then the surface area should be expressed in square feet).

Note that individual lake features (defined by their unique identifier in the second field of the lake-definition file) will often have multiple data records in the lakes definition file because they often extend across multiple model grid cells. Also, note that the program is a modified version of the previously-described program, sim_q_reach_3d.py. As such, the capability exists for a given '2-dimensional' GIS feature to be represented with multiple River Package boundary features occurring at the same

[3] Ideally, the width this identifier should be less than twenty characters so that it can be used as part of a corresponding PEST observation name (which can be no wider than twenty characters), if desired.

horizontal location (e.g. model row and column location) but in multiple model layers (for example a lake that penetrated multiple model layers). This capability is not used for the NFSEG model. However, if this capability were to be employed, then the actual surface area of a given 2-dimensional GIS feature should only be assigned to one of the records in the lake-definition file corresponding to this feature, and values of zero should be assigned to the surface-area field for the other records corresponding to the 2-dimensional GIS feature that are in different model layers. This prevents the program from repeatedly adding the same free-water surface area (for a given 2-dimensional lake feature) when it computes the total surface area for a lake comprised of multiple River Package boundary condition records.

As described previously in the description for the program, sim_q_reach_3d.py, the lookup table is actually a Python dictionary (called bc_reach_id_dict), contained in a Python 'shelf' file (called lookup_bc_reach_ids.shelf). The 'keys' for the dictionary, bc_reach_id_dict, are a Python tuple with two elements: a three-letter boundary-condition type identifier (see previous discussion of this identifier) and an integer stress period value. The values corresponding to these keys are Python lists containing a list of values of unique identifiers for a given MODFLOW boundary-condition type. The ordering of the elements in this list is identical to the ordering of records in the corresponding MODFLOW boundary condition input file. For example, if the 4380th MODFLOW River Package input file data record for stress period 2 corresponds to the unique identifier, 4266, for the River Package boundary condition feature in a particular model, then bc_reach_id_dict[('riv',2)][4379] would return a value of 4266 (note that Python, like many computer languages, uses 'zero-based' indexing, so we used a value of 4379 to refer to the 4380th element in the list). Embedding this lookup-table in a Python dictionary in a Python shelve file greatly speeds up the execution of the program sim_q_lakes_3d.py. This shelve file (lookup_bc_reach_ids.shelf) can be easily created using a 'batch file' that calls a set of utility programs that were written for this purpose. Please see the Appendix ('Creation of a Boundary-Condition Feature Lookup Table') for details.

Program Execution:

The program is executed from the command prompt using the following syntax:

```
python sim_lakes_3d.py [listfile] [optional conversion factor1] [optional conversion factor2]
```

where

[listfile] is the name of the MODFLOW-NWT output listing file (for example, nfseg.lst),

[optional conversion factor1] is a conversion factor for converting the MODFLOW simulated 'volume-per-unit-time' fluxes from unit to another. For example, a conversion factor of 1.157407407E-05 (1 day /86400 seconds) should be specified if converting from units of cubic feet per day to cubic feet per second. The conversion-factor1 argument is optional, so it can be omitted if unit conversion isn't needed.

[optional conversion factor2] is a conversion factor for converting the MODFLOW simulated 'length-per-unit-time' fluxes from unit to another. The conversion factor is optional if the previous conversion factor is not specified. For example, a conversion factor of 4380 (1 feet per day * 12 inches per feet * 365 days per year) could be specified if converting from units of cubic feet per day to inches per year. Note that the program is structured to default to a conversion from feet per day to inches per year if the first conversion-factor is not specified.

Please note that you'll need to have the Python interpreter installed on your computer for the above command to work. The program was tested using version 2.7 of the Python interpreter, but some of the earlier '2.x' versions of the interpreter will probably work as well. You'll also need to have the PATH environment variable set appropriately so that your command-line shell will know where to find the Python interpreter. This was probably set when Python was installed on your computer. Note that on Unix-based operating systems, this can also be accomplished by inserting a string like, #!/usr/bin/python, at the top of the program.

Program Output:

The program, sim_q_reach_3d.py, creates a spaces-delimited, output file called, gaged_reach_fluxes.asc. This file has one header line, followed by a set of 'data records' containing the following fields (from right to left):

1. **lake_id**: numeric identifier for a given lake
2. **lake_name**: name of a given lake
3. **lake_area:** free-surface area of a given lake
4. **riv_sim_flux_sp1_ts1_volume_over_time:** simulated flux from the lake to the groundwater flow system by the MODFLOW River Package during stress period 1, in units of volume per unit time
5. **riv_sim_flux_sp1_ts1_length_over_time:** simulated flux from the lake to the groundwater flow system by the MODFLOW River Package during stress period 1, in units of length per unit time
6. **total_sim_flux_sp1_ts1_volume_over_time:** simulated flux from the lake to the groundwater flow system by all[4] MODFLOW boundary-condition packages during stress period 1, in units of volume per unit time**,**
7. **total_sim_flux_sp1_ts1_length_over_time:** simulated flux from the lake to the groundwater flow system by all[4] MODFLOW boundary-condition packages during stress period 1, in units of volume per unit time**,**
8. **abs_total_sim_flux_sp1_ts1_length_over_time**: absolute value of **total_sim_flux_sp1_ts1_length_over_time,**
9. **riv_sim_flux_sp2_ts1_volume_over_time:** simulated flux from the lake to the groundwater flow system by the MODFLOW River Package during stress period 2, in units of volume per unit time**,**
10. **riv_sim_flux_sp2_ts1_length_over_time:** simulated flux from the lake to the groundwater flow system by the MODFLOW River Package during stress period 2, in units of length per unit time**,**
11. **total_sim_flux_sp2_ts1_volume_over_time:** simulated flux from the lake to the groundwater flow system by all[4] MODFLOW boundary-condition packages during stress period 2, in units of volume per unit time**,,**
12. **total_sim_flux_sp2_ts1_length_over_time:** simulated flux from the lake to the groundwater flow system by all[4] MODFLOW boundary-condition packages during stress period 2, in units of volume per unit time**,,**
13. **abs_total_sim_flux_sp2_ts1_length_over_time,** absolute value of **total_sim_flux_sp2_ts1_length_over_time**

[4] I should have left these 'total' fields out (they are a holdover from the program, sim_q_reach_3d.py), because the program, sim_lakes_3d.py, is currently only setup to recognize lakes that are represented using the MODFLOW River Package

14. **del_total_sim_flux_volume_over_time**: **total_sim_flux_sp2_ts1_length_over_time** minus **total_sim_flux_sp1_ts1_length_over_time,**
15. **del_total_sim_flux_length_over_time: total_sim_flux_sp2_ts1_length_over_time** minus **total_sim_flux_sp1_ts1_length_over_time.**

Appendix A: Creation of a Boundary-Condition Feature Lookup Table

Creation of the boundary-condition feature lookup table described in the previous sections, **Description of program, sim_q_reach_3d.py** and **Description of program, sim_q_lakes.py** is facilitated through the execution of a command-line 'batch' file, create_reach_id_lookup.bat, which calls a set of Python programs that read data from files with MODFLOW River and GHB Package 'data records'[5] and ultimately stores them in a Python 'shelve' file. Input to the batch file consists of four files with the following file names:

1. ***nfseg.riv.sp1*** – a plain-text, ASCII file containing the stress-period data-records from the first stress period of the MODFLOW River Package input file
2. ***nfseg.riv.sp2*** – a plain-text, ASCII file containing the stress-period data-records from the second stress period of the MODFLOW River Package input file
3. ***nfseg.ghb.sp1*** – a plain-text, ASCII file containing the stress-period data-records from the first stress period of the MODFLOW General Head Boundary Package input file
4. ***nfseg.ghb.sp2*** – a plain-text, ASCII file containing the stress-period data-records from the second stress period of the MODFLOW General Head Boundary Package input file.
5. ***nfseg.drn.sp1*** - a plain-text, ASCII file containing the stress-period data-records from the first stress period of the MODFLOW[6]

The program is executed by opening a command-line window and executing a batch file called create_reach_id_lookup.bat (if running from a 'DOS' command line on the Windows operating system) or create_reach_id_lookup.sh (if running from a Unix-based command line). The batch file calls a set of Python programs (RiverPackageInput.py, GHBPackageInput.py, create_reach_id_lookup.py, and add_drn_to_lookup_bc_reach_ids_shelf.py), and outputs a Python shelve file called lookup_bc_reach_ids.shelf that contains a Python two python dictionaries. The first is called, bc_reach_id_dict, and its contents are described in the previous section, **Description of program, sim_q_reach_3d.py**. The second dictionary is called, reach_ids_from_2d_ids, and its contents are described in the previous section, **Description of program, sim_q_lakes.py.**

Miscellaneous Documentation Revision Notes:

When	Who	Description

[5] These are the records in the input file containing the layer, row, column, stage, conductance, and rbot (only for the River Package). These files should also contain the unique identifier (to locate the feature in a GIS), immediately to the right of either the rbot field (for River Package data) or the conductance field (for GHB Package data).

[6] Note that, if at some point in the future the Drain Package input file in stress periods 1 and 2 are no longer identical, then the processing step used to add 'lookup data' from the Drain Package input file to the shelf file will need to be updated.

2015-09-07	Trey Grubbs	Original documentation
2015-09-15	Trey Grubbs	Add text to description of sim_q_lakes.py
2016-06-13	Trey Grubbs	Corrected notes describing program, sum_sim_q_reach.py, so that the correct name of the program is used in the documentation (the program name, sum_sim_q_reach_3d.py, was incorrectly used in the previous version of the documentation). Also added some information to the introduction.
2016-02-03	Trey Grubbs	Minor corrections to Appendix A and also update Appendix A to include description of procedure for adding Drain Package features to the file, lookup_bc_reach_ids.shelf.