User and Developer manual to the WBMsed.v2 model

Sagy Cohen, CSDMS, University of Colorado at Boulder
sagy.cohen@colorado.edu

1. Introduction

The WBMsed model (Cohen et al., 2011) is an extension of the WBMplus water balance and transfer model (Wisser et al., 2010). In its initial version WBMsed introduced a riverine suspended sediment flux module based on the BQART (Syivitski and Milliman, 2007) and Psi (Morehead et al., 2003) models.

This document will provide instructions on how to run the model and further develop its code and datasets. The instructions here are based on my experience and understanding of the WBMplus model (with some correspondence with the latest developer of WBMplus: Balazs Fekete, The City College of New York) and may not always be the most optimal solution. The instructions here are based on using the CU-CSDMS HPCC (High Performance Computer Cluster, aka 'beach') but are transferable to a desktop computer (preferably a Mac).

2. The model infrastructure

The WBM modeling platform was designed to be highly modular. Each process is written as an individual or a sequence of module (e.g. `MFDischarge.c`). These modules, as well as the main module (`WBMMain.c`), are stored in `/Model/WBMsed/src`. The modules are typically a simple piece of code that utilizes the numerous WBM functions which are responsible for the computational heavy lifting. Most of these functions are stored in the `CMlib` and `MFlib` libraries (in the `/Model` directory).

The model I/O is based on the RGIS (River GIS) formats. The model uses the RGIS data manipulation functions (stored at the `'ghaas'` directory) to read, write and convert the datasets.

The model run is controlled by Unix shell scripts located at the `/Scripts` directory. These scripts determine the input datasets and simulation parameters. More about these at the next section.

3. Running the model

3.1. Compiling – the model needs to be recompiled before lunching it on a new computer platform or if its C code was modified. When compiling the model on a new platform all three model libraries (`WBMplus, MFlib, CMlib`) and the RGIS library (`ghaas`) needs to be compiled. I recommend compiling each one individually by running the '`make`' command in each directory e.g.:

```
> cd MFlib
> make
```

On a desktop computer you will likely need to install the 'Fink' libraries.

Typically you will only modify the WBMplus code and will only need to recompile it:

```
> cd Model/WBMsed
> make
```

A successful compilation will create a new `wbmsed<version>.bin` file in `/Model/WBMsed/bin`

3.2. Creating the simulation directory – Create a new directory and copy the following folders into it: `ASCII, Model, Scripts, BQARTmeanInputs.` I would also recommend copying the '`ghaas`' to your home directory (not to the simulation directory). You can use a SFTP software (like Cyberduck or Fugu) to manage your data.

3.3. Setting the run script – the run shell script (e.g. `WBMsed3.2.2_Qn1_dTS1960-2010.sh`) control the simulation. It has a rigid structure that must be kept.

The WBMsed model (unlike WBMplus) requires a separate initial simulation. This initial simulation is needed only once for each simulation domain (e.g. Global, North America). This simulation is controlled by the `BQARTpreprocess.sh` script. It generates long-term temperature and discharge outputs (/`BQARTmeanInputs` directory) used in the main simulation controlled by the main script (e.g. `WBMsed3.2.2_Qn1_dTS1960-2010.sh`).

Below I describe the important variable in the model simulation script file.

`PROJECTDIR` –Define the simulation directory –where the model is and where to save intermediate and final results. Assumes that the script file and the model are in the same directory (simulation directory).

GHAAS_DIR – the location of the ghaas directory. I would recommend copying this directory to your home directory and set this variable accordingly. You don't need to change this variable between simulations as long as the ghass directory is accessible.

MODEL – the location of the model bin file. If you copy the Model folder to your simulation directory (as directed in step 2) you don't need to change this variable.

RGISARCHIVE – the location of the input datasets directory. On beach use /scratch/ccny/RGISarchive. More about adding input datasets later.

RGISPILOT – on beach set to /scratch/ccny/RGISpilot

RGISRESULTS – here you determine where you want the model to save the results. The default is "${PROJECTDIR}/RGISresults"

RGISCSDMS- the location of WBMsed specific (as appose to WBMplus) input files on beach set to /scratch/saco2635/RGISarchiveCSDMS

MEANINPUTS- the location of the WBMsed phase 1 final output files.

EXPERIMENT – the name of the simulation. Will be imbedded in the output files name.

STARTYEAR – the simulation start year. Make sure the input datasets begins at that or an earlier year.

ENDYEAR – the simulation end year. Make sure the input datasets reach this year.

AIRTEMP_STATIC up to PRECIP_FRAC_DYNAMIC – setting the air temperature and precipitation input datasets.

NETVERSION – the flow network dataset. Use a different dataset for 6 and 30 minute spatial resolution. I found the "PotSTNv602" good for global 30min simulations, "STN+HydroSHEDS" for global 6min simulation and "PotSTNv120" for North-America (6min).

FwArguments – here you define some of the simulation parameters. The important ones are:

-s (on/off) – spin-up – a set of simulation cycles to initializing the model runs;

-f (on/off) – final run – the actual simulation after the spin-up cycles;

-n (#) - number of spin-up cycles;

-u (on/off) – purge files – delete intermediate outputs after each year of simulation (save disc space);

`-D (on/off)` – daily output – when set to 'on' the model will create yearly, monthly and daily output layers, when 'off' it will only create yearly and monthly (can save a lot of disk space).

To see the full list of option go to the `FwArguments()` function in `/Model/MFlib/Scripts/fwFunctions20.sh`

`FwInit` – a function call (to `fwFunctions20.sh`) which setup the flow network file.

`DATASOURCES` - the input datasets array. Here you control the dataset which corresponds to each input parameter in your simulation. You can add inputs here but must follow the syntax. More about how to add an input later.

`OPTIONS` – the simulation options array. The "`Model`" option defines which module the model will start with. So if you want to simulate discharge alone you write "`Model discharge`"; and the model will first go to the discharge module and initiate all the relevant modules from there. If you want to simulate sediment you write "`Model sedimentflux`". In this case discharge will also be simulated as it is called by the `SedimentFlux` module.

`OUTPUTS` – the simulation output array. Here you define which of the model parameters will be exported as outputs. These names must correspond to parameters name in the modules and in the `/Model/WBMplus/include/MF.h` file. More about adding new parameters in the developers section (#4) below.

`FwDataSrc` - a function call (to `fwFunctions20.sh`) which set up the input data source listed.

`FwOptions` - a function call (to `fwFunctions20.sh`) which set up the option listed.

`FwOutputs` - a function call (to `fwFunctions20.sh`) which set up the outputs listed.

`FwRun` - a function call (to `fwFunctions20.sh`) which controls the model run.

`./rgis2netcdf.sh ../RGISresults/ ../RGISresults/` - (only in WBMsed) after the simulation, run a script to convert the simulation outputs from RGIS format to NetCDF.

`./rgisDelete.sh ${RGISRESULTS} ${RGISRESULTS}` - (only in WBMsed) run a script to delete the output RGIS files.

As you can see the `fwFunctions20.sh` (in `/Model/MFlib/Scripts`) is an important file containing many of the shell functions needed to run the model. I have modified this file to improve its parallel I/O for WBMsed. The modified file is called `fwFunctions20_multi_postproc.sh`.

You can use any text-editing tool (e.g. vi) to edit the script file. If you create a new shell script you will probably need to define its permissions. You do so with the `chmod` command:

```
> chmod 755 filename.sh
```

3.4. Launching the model – when you log into beach you start in its headnode. DO NOT run long or heavy calculations on the headnode as it slows it down for everyone. For running the model on beach you MUST use the Torque system. Torque has many useful options and tools for controlling your simulation (see: http://csdms.colorado.edu/wiki/HPCC_usage_rules). You used the interactive mode that opens one of beach's sub-nodes in which you can run your simulation:

```
> qsub -I
```

This interactive mode is simple and limited. Note that you cannot lunch graphic (X11) applications in a sub-node only from the headnode.

After you are directed to a sub-node. Go to the script directory:

```
> cd <name of simulation directory>/Scripts
```

Launch the run script like this:

```
> ./ WBMsed3.2.2_Qn1_dTS1960-2010.sh Global 30min dist
```

Or

```
> ./ WBMsed3.2.2_Qn1_dTS1960-2010.sh NAmerica 06min dist
```

The first argument is the shell script name.

The second argument is the simulated domain (e.g. Global, NAmerica). You need to be aware of the input datasets available for each domain. The model will automatically use a Global dataset to fill in for missing datasets in smaller domain. The most important dataset to run a model in a smaller domain is the Network.

The third argument is the spatial resolution (30min or 06min). If the model cannot find a high-resolution dataset (06min) it will use a lower resolution.

The fourth argument can take the following options:

`dist` - distributed simulation (standard);

`prist` - pristine simulations (turning off the irrigation and reservoir operation modules);

`dist+gbc` or `prist+gbc` - are the same but turning on the water chemistry modules.

A better way of lunching the model is using a Torque script file. This allows you to define a number of simulation parameters and will direct you to the most appropriate queue on beach. Here is an example of a Torque script file to run WBMsed (`Torque_long_Qbf.sh`):

```
#!/bin/sh

#PBS -N WBMsed_6min_runoff
#PBS -o job_out.txt
#PBS -e job_err.txt
#PBS -m ae
#PBS -M sagy.cohen@colorado.edu
#PBS -l walltime=100:00:00
#PBS -l nodes=1:ppn=1
#PBS -l mem=10GB

cd ${PBS_O_WORKDIR}
pwd
./WBMsed3.2.2_Qn1_dTS1960-2010.sh Global 06min dist
```

The #PBS notation define the different options for the simulation:

`-N` – simulation name which will appear on the Torque queue (`>qstat` command);

`-o` – a file to which Torque write the runtime output text (in the Scripts directory);

`-e` - a file to which Torque write the runtime errors t (in the Scripts directory);

`-M` – Torque will send you an email to this address when the simulation is done;

`-l walltime=` - how much time do you need for this simulation, will determine the queue your simulation will be put on (default, long etc.)

`-l nodes=` - how many nodes and processors in each node (ppn) do you need. For the WBMsed version you need only one node and up to 3 ppn depending on the '`fwFunctions`' you defined (section3.3; `fwFunctions20.sh` = 1ppn, `fwFunctions20_3multi_postproc.sh`=3ppn).

`-l mem=` - how much memory do you need for this simulation. For 1ppn you need about 10GB for 3ppn you will need 30GB which will direct you to the 'himem' queue.

Write (or copy) the Torque script in the '`Scripts`' folder in your simulation directory. To lunch it:

```
> cd Scripts
> qsub Torque_long_Qbf.sh
```

This will put your simulation in the Torque queue. You can check the queue with the `qstat` command. You can limit the display to just your runs but with more details:

```
> qstat -u saco2635
```

3.5. During the simulation – first the model will create a folder named `GDS` where the intermediate input and output datasets and log files will be created and updated during the simulation. The spin-up cycles will use the first year input datasets in all the spin-up cycles. After the last spin-up cycle the model will create a new folder named `RGISresults` where the output files will be stored. For each simulation year the model will create a new set of input and output files in GDS and at the end of the year it will export that year's final output files to '`RGISresults`'.

As described, WBMsed has a separate initial simulation to generate input files for the main simulation. This initial simulation results are stored in a folder named '`RGISresults-phase1`' and the inputs to the main simulation are stored in a folder named '`BQARTmeanInputs`'.

3.6. Viewing and analyzing the results – there are tools for viewing the RGIS format on beach but they are not so good in my experience. I use the `VisIt` package either on beach (at `/usr/local/visit-2.0.2-parallel`) or on my local computer (free download at: https://wci.llnl.gov/codes/visit/executables.html). `VisIt` reads NetCDF files (hence the conversion at the end of the run script) and has a number of useful tools for analysis.
You can also import NetCDF files to ArcGIS (Tools -> Multidimension Tools) on your local computer.

4. Developer guide

This section will show how to develop the WBM code and how to compile new input and output datasets and incorporate them in the model simulation. The explanations here are based on my experience from developing the WBMsed model.

4.1 Building and editing a module

As in all C programs the first lines in a WBM module are the `#include` definition. In addition to the standard and general C libraries (e.g. `stdio.h` and `math.h`) a module must include the WBM libraries: `cm.h, MF.h` and `MD.h`. These header files are located in the `'/include'` directories in the: `CMlib, MFlib` and `WBMplus` directories respectively. They contain the functions and parameters used in WBM.

After the `#include` list we define all the input and output parameters ID in the module like this:

```
static int _MDInDischargeID = MFUnset;
```

These IDs are used in the WBM functions to query (e.g. `MFVarGetFloat`) and manipulate (e.g. `MFVarSetFloat`) the model parameters. `MFUnset` is an initial value before an actual ID is set in the definition function.

WBM is built as an array of modules each typically computes a specific process (e.g. `MFDischarge.c`). Each module contain two functions: (1) a what we will call **main function** and (2) a **definition function.** In `MFSedimentFlux.c` module the main function is called `'_MDSedimentFlux'` and the definition function is called **'**`MDSedimentFluxDef`**'**.

The definition function set the ID of all the input and output parameters used in the main function. If a parameter (e.g. Discharge) is calculated in a different module within the WBM model this module is initialized like this:

```
((_MDInDischargeID = MDDischargeDef ()) == CMfailed) ||
```

where `_MDInDischargeID` is the variable that holds the discharge parameter ID in the `MFSedimentFlux.c` module, `MDDischargeDef ()` is the name of the definition function in the Discharge module (`MFDischarge.c`) and `CMfailed` is an error control variable (note the '`if`' at the start of the parameter definition list).

This is how WBM works, it starts with one module (`MFSedimentFlux.c` in the WBMsed case) and each module calls the other modules it needs. This chain of module interactions is recorded

at the 'Run<year>_Info.log' file during the simulation (at the /GDS/…..../logs directory).

An input and output dataset parameter (e.g. air temperature) ID is defined like this:

```
((_MDInAirTempID = MFVarGetID (MDVarAirTemperature, "degC", MFInput,
MFState, MFBoundary)) == CMfailed) ||
```

where _MDInAirTempID is the parameter ID variable, MFVarGedID is a WBM function that define IDs to input and output parameters. It requires the following arguments:

The first argument is the parameter name. This variable is what links the module to the simulation shell script (e.g. BQARTdaily.sh) input and output list (DATASOURCES and OUTPUTS respectively; see section 3 above).

The second argument is the parameter units (e.g. "degC", "km2", "m3/s"). This variable does not seem to have an effect on the model run and is for cataloging purposes.

The third argument can take the following options: MFInput- a regular input parameter; MFOutput a regular output parameter; MFRoute- the model will accumulate the parameter content in the downstream grid cell, so by the time the execution gets to the downstream grid cell it already contains the accumulated values from upstream.

The fourth argument affects how temporal disaggregation is handled. It can take the following options: MFState- the input value is passed to the modules as is; MFFlux- the input value is divided by the number of days in the input time step.

The fifth argument affects how a variable is read. It can take the following options: MFBoundary- the parameter is read constantly from input, by matching up to the current time step; MFInitial- the model forwards to the last time step in the data, reads in the latest values and closes the input stream assuming that the parameter will be updated by the model.

After the list of input and output parameters ID definitions the module initiates the main function:

```
(MDModelAddFunction (_MDSedimentFlux) == CMfailed)) return (CMfailed);
```

where MDModelAddFunction is a WBM function, _MDSedimentFlux is this function argument – the name of the module main function. CMfailed is an error controller.

Next is the MFDefLeaving function which takes a string argument (e.g. "SedimentFlux") of the module name.

The final line in a module definition function returns the module main output ID (e.g. `_MDOutSedimentFluxID`) which was defined in the parameters ID definition list above it.

The main function (e.g. `_MDSedimentFlux`) is where the processes are simulated. It gets at least one argument called '`itemID`' in its definition line:

```
static void _MDSedimentFlux (int itemID) {
```

`itemID` is a pixel number. WBM assigns a number to each pixel based on its location in the flow network. The model goes through the whole simulation cycle a pixel at a time at a daily time step. So for one model iteration (a day) it calls each simulated module a number of times equal to the number of simulated pixels. For each such call the `itemID` change in accordance to downstream pixel location (e.g. pixel #345 is downstream of pixel #344). Note that the pixel numbering is continues so a hinterland pixel number of a basin can follow an outlet pixel of a different basin on a different continent.
As in all C functions the first lines in a module main function are the local variable declarations. WBM use a multitude of functions to query and manipulate its parameters. My personal preference is to assign a variable to both input and output parameters.

First I **get the input parameter value**, for example:

```
R = MFVarGetFloat (_MDInReliefID, itemID, 0.0);
```

where `R` is the relief variable, `MFVarGetFloat` is a WBM function that reads a parameter value. It get the following arguments:
The first argument is the parameter ID (e.g `_MDInReliefID`). This ID was assigned at the module definition function (see above);
The second argument is the pixel ID (i.e. `itemID`);
The third argument is an initial value (I'm not sure what it's for!).

I can then easily manipulate this variable:

```
R = R/1000; // convert to km
```

I then use the variables to calculate the module processes, for example:

```
Qsbar = w * B * pow(Qbar_km3y,n1) * pow(A,n2) * R * Tbar;
```

Finally I **set the resulting output parameters**:

```
MFVarSetFloat (_MDOutQs_barID, itemID, Qsbar);
```

where `MFVarSetFloat` is a WBM function that set or update a parameter value. It gets the following arguments:

The first argument is the parameter ID (e.g `_MDOutQs_barID`).

The second argument is the pixel ID (i.e. `itemID`);

The third argument is the variable that holds the value you wish to set (`Qsbar`).

These get and set operations are also used for bookkeeping purposes. For example in WBMsed we need to calculate long-term average temperature for each pixel and then basin-average it (we consider each pixel to be a local outlet of its upstream contributing area).

I start by getting daily temperature for each pixel from an input dataset:

```
Tday = MFVarGetFloat (_MDInAirTempID, itemID, 0.0);
```

I then temporally accumulate a pixel temperature into a bookkeeping parameter (`_MDInNewAirTempAccID`):

```
T_time=(MFVarGetFloat(_MDInNewAirTempAcc_timeID, itemID, 0.0)+ Tday);
MFVarSetFloat (_MDInNewAirTempAcc_timeID, itemID, T_time);
```

Note that I'm using the `MFVarGetFloat` function within the calculation in this case. This is a more efficient way of coding but can be harder to debug.

In order to spatially average temperature I first spatially accumulate the temporal summation (`T_time`):

```
Tacc = (MFVarGetFloat (_MDInAirTempAcc_spaceID, itemID, 0.0) + T_time
* PixelSize_km2);
```

where `PixelSize_km2` is the size (area) of the pixel (calculated with the `MFModelGetArea` function).

I then average the temperature by dividing it by the number of iteration passes since the start of the simulation (`TimeStep`) and the size of the pixel upstream contributing area (`A`):

```
Tbar = Tacc/TimeStep/A;
```

Finally I pass the `Tbar` value to an output parameter (`_MDOutBQART_TID`):

```
MFVarSetFloat (_MDOutBQART_TID, itemID, Tbar);
```

An important note: to allow downstream spatial accumulation a parameter needs to be defined (in the definition function) as `MFRoute`. If it is set as `MFOutput` it will accumulate things in time. The definition arguments were described earlier.

4.2 Adding a new output parameter to a module

WBM is an interaction between a core C program and shell script files that control it. Therefore in order to add a new output parameter we need to accurately define this new parameter in several locations. Below is a step-by-step description of how to add an output parameter using the sediment load (`Qsbar`) parameter in the `MFSedimentFlux.c` module.

Step 1- at the start of the module C file (`MFSedimentFlux.c`) add the ID variable declaration:

```
static int _MDOutQs_barID  = MFUnset;
```

Step 2- in the module definition function (`MDSedimentFluxDef`) add an ID definition line:

```
((_MDOutQs_barID = MFVarGetID (MDVarQs_bar,"kg/s",MFOutput, MFState,
MFBoundary)) == CMfailed) ||
```

This code was explained earlier.

Step 3- define the parameter "nickname" in the WBMsed header file (`MD.h` located at `/Model/WBMplus/include`):

```
#define MDVarQs_bar "Qs_bar"
```

Note that the parameter name (`MDVarQs_bar`) must be the same as in step 2.

Step 4- you can now use the parameter in the module main function (`_MDSedimentFlux`) for example to set its value:

```
MFVarSetFloat (_MDOutQs_barID, itemID, Qsbar);
```

This code was explained earlier.

Step 5- to set WBMsed to create an output dataset of this parameter you need to add it to simulation script (e.g. `BQARTdaily.sh`) output list:

```
OUTPUTS[${OutNum}]="Qs_bar"; ((++OutNum))
```

Note that the parameter "nickname" must be similar to step 3.

4.3 Adding an input parameter of an existing dataset

The steps for adding an input parameter linked to a dataset already in existence at the WBM datasets library (the `/data/ccny/RGISarchive` directory on beach) are similar to adding an output parameter with the following differences:

In **Step 2** the arguments of the `MFVarGetID` function need to reflect an input parameter (see description in section 4.1).

In **step 5** you should add the parameter to the datasources (rather than the `OUTPUTS`) list like this:

```
DATASOURCES[${DataNum}]="FieldCapacity static Common file $(RGISfile
${RGISARCHIVE} ${DOMAIN}+ field_capacity WBM ${RESOLUTION}+ static)";
((++DataNum))
```

This exact syntax must be kept for the model to get the dataset name and location. In the example above '`FieldCapacity`' is the parameter "nickname" defined in `MD.h` (**Step 3**) and '`field_capacity`' is an RGIS "nickname" defined at the RGISfunctions.sh file located at the `/ghaas/scripts` directory. In the above example if:

`RGISARCHIVE = "/data/ccny/RGISarchive"` (defined in the simulation script e.g. `BQARTdaily.sh`);

`DOMAIN = "Global"` (entered at the model run command; see section 3.4);

`RESOLUTION = "30min"` (entered at the model run command; see section 3.4);

The model translates this to the following file location and name:

```
/data/ccny/RGISarchive/Global/Soil-FieldCapacity/WBM/30min/Static/
Global_Soil-FieldCapacity_WBM_30min_Static.gdbc
```

4.4 Preparing and adding a new input dataset

WBM use the RGIS (River GIS) format for both input and output datasets. Below I describe the procedure I use to compile new input datasets that can be read by WBMsed. This description is focused on manipulating ArcGIS and NetCDF rasters and using the RGIS tools available on beach (at the `/ghaas/bin` directory). Other tools may be needed for different GIS packages and formats. The compilation of the maximum relief input dataset is used as an example.

The maximum relief layer is the difference between a pixel local topographic elevation (DEM) and the maximum elevation of its upstream contributing area. The calculation of the relief layer was done in ArcGIS. For local elevation I used a DEM available at the `RGISarchive` directory (e.g. `Global_Terrain_MeanElevation_LTxxxx-30min.gdbc`). There are RGIS tools for converting a RGIS dataset to an ArcGIS format but I have found them to be somewhat outdated.

Importing an RGIS dataset to ArcGIS is done by:

Step 1- Convert the RGIS layer to NetCDF format with the `rgis2netcdf` tool (at `/ghaas/bin`):

```
> rgis2netcdf Global_Terrain_MeanElevation_LTxxxx-30min.gdbc
Global_Terrain_MeanElevation_LTxxxx-30min.nc
```

where the first argument is the RGIS file name and the second is the name of the converted NetCDF file (.nc).
Step 2- Import the NetCDF file to ArcGIS using the ArcGIS 'Make NetCDF Raster Layer' tool (at Tools -> Multidimension Tools).

After calculating the relief layer in ArcGIS (not described here) **export the product to ASCII** format with the ArcGIS 'Raster to ASCII' tool (Conversion Tools -> From Raster).

On beach **convert the ASCII file to RGIS** with the RGIS '`grdImport`' tool (`/ghaas/bin`):

```
> grdImport Max-Relief_30min_Global.asc
```
(and follow the prompt)

Set the resulting RGIS file header either with the RGIS `setHeader` tool:

```
> setHeader –a Max-Relief_30min_Global.gdbc
```
(and follow the prompt)

Or use the RGIS GUI (`/ghaas/bin`):

```
> ./rgis21
```

and use the '`File -> Header Info`' tool.

It is crucial to **set the layer date** in the RGIS GUI with the '`Edit -> Date Layer`' tool. For a temporally static dataset (e.g. Relief) set the year to `xxxx`.
Save these changes in the RGIS GUI ('`File -> Save`').

Test the layer conversions and edits by **converting it to NetCDF**:

```
> rgis2netcdf Max-Relief_30min_Global.gdbc Max-Relief_30min_Global.nc
```

and open it in VisIt on beach (`/usr/local/visit-2.0.2-parallel/bin`):

```
> ./visit
```

Finally you need to **rename the RGIS layer and put it in a directory** in accordance to the model's datasources format (in the simulation script e.g. `BQARTdaily.sh`, described in section 4.3 above):

```
/data/ccny/RGISarchive/Global/Relief-Max/ETOPO1/30min/Static/
Global_Relief-Max_ETOPO1_30min_Static.gdbc
```

This dataset name and location should be **referenced in the simulation script** datasources list:

```
DATASOURCES[${DataNum}]="ReliefMax static Common file $(RGISfile
${RGISARCHIVE} ${DOMAIN}+ relief_max PotSTNv120 ${RESOLUTION}+
static)"; (( ++DataNum ))
```

For this reference to work we need to **set the two "nicknames":**

The first (e.g. "`ReliefMax`") in `/Model/WBMplus/include/MD.h` file:

```
#define MDVarRelief "ReliefMax"
```

The second (e.g. "`relief_max`") in the `/ghaas/Scripts/ RGISfunctions.sh` file:

```
(relief_max)
     echo "Relief-Max"
;;
```

The new dataset can now be used in a module (e.g. `MFSedimentFlux.c`) as explained in section 4.3 above.

4.5 Incorporating a new module to WBM

The WBM module structure was explained in section 4.1. Here I will describe the additional steps needed to incorporate a new module to WBM. As before I will use the `MFSedimentFlux.c` module as an example.

I recommend starting with a simple module with just a couple of parameters and gradually add to it once it is successfully incorporated in WBM (as will be described next). A simple way to do so is to copy an existing module file (e.g. `MFDischarge.c`) change its file and functions (main and definition functions) names and delete most of its code, leaving just a couple of parameters and variables:

```
#include <stdio.h>
#include <string.h>
#include <cm.h>
#include <MF.h>
#include <MD.h>

// Input parameter
static int _MDInDischargeID = MFUnset;
// Output parameter
static int _MDOutSedimentFluxID = MFUnset;

//The main function:
static void _MDSedimentFlux (int itemID) {
      float Qs,Q;

      Q = MFVarGetFloat (_MDInDischargeID,itemID, 0.0);
      Qs = pow(Q,2);
      MFVarSetFloat (_MDOutSedimentFluxID,itemID, Qs);
}

//The definition function:
int MDSedimentFluxDef() {
      MFDefEntering ("SedimentFlux");

      if (((_MDInDischargeID = MDDischargeDef()) == CMfailed) ||
      ((_MDOutSedimentFluxID = MFVarGetID (MDVarSedimentFlux,"kg/s",
MFRoute, MFState, MFBoundary))== CMfailed) ||

      (MFModelAddFunction (_MDSedimentFlux) == CMfailed)) return
(CMfailed);

      MFDefLeaving ("SedimentFlux");
      return (_MDOutSedimentFluxID);
}
```

The next step is to add the new definition function to the model header file (`/Model/WBMplus/include/MD.h`):

```
int MDSedimentFluxDef();
```

Add the input and output parameters to the simulation script file (e.g. `/Scripts/BQARTdaily.sh`) and model header file (`MD.h`) as described in sections 4.2-4.4. In the example above we only need to add the `MDVarSedimentFlux` parameter to the header (`MD.h`) file:

```
#define MDVarSedimentFlux "SedimentFlux"
```

and set it as an output in the simulation script file:

```
OUTPUTS[${OutNum}]="SedimentFlux"; ((++OutNum))
```

The input parameter in this case is an existing module (`MDDischargeDef`) so is it is already defined.

The next two steps depend on whether the new module will be a leading module or not. A leading module is the first module at the module simulation chain. For example in WBMsed `MFSedimentFlux.c` is the leading module. The model starts with this module which then calls the modules it needs which intern call the modules they require and so on.
In the case of **a new leading module** you need to add it to the WBM main file (`/Model/WBMplus/WBMmain.c`) in the following places:

1. In the `enum` declaration: `MDsedimentflux};`
2. In the `*options[]` definition: `"sedimentflux",(char *) NULL };`
3. In the `switch(optID){` list:

```
case MDsedimentflux: return (MFModelRun
(argc,argv,argNum,MDSedimentFluxDef));
```

At the simulation script file (e.g. `/Scripts/BQARTdaily.sh`) you need to change the `OPTIONS "Model"` argument to:

```
OPTIONS[${OptNum}]="Model sedimentflux"; (( ++OptNum )):
```

And add:

```
OPTIONS[${OptNum}]="SedimentFlux calculate"; (( ++OptNum ))
```

In the case where the module is **not a new leading module** you only need to add an initiation call in a relevant calling module. In the example above the `MFSedimentFlux.c` module is initiating the `MFDischarge.c` module by calling its definition function:

```
((_MDInDischargeID = MDDischargeDef()) == CMfailed) ||
```

The final step is to **add the new module to the WBM '`Makefile`' file** (at `/Model/WBMplus/src/` directory):

```
$(OBJ)/MDSedimentFlux.o\
```

and **compile the model** (see section 3.1).