The HL Research Distributed Hydrologic Model (HL-RDHM)
Developer's Manual v. 2.0

August 14, 2008

Contents

Chapter 1

Introduction

This Developer's Manual for the HL Research Distributed Hydrologic Model (HL-RDHM) describes the software structure and the common programming framework the HL-RDHM provides to distributed hydrologic model developers. For information on various modeling capabilities of RDHM, please refer to the RDHM User's Manual[7].

In addition distributed modeling capabilities, another goal of RDHM is to develop a common programming framework for hydrologic model developers and scientists to test scientific ideas and new models in a fast and easy way. However, it is not meant for operational usage, although it may share common structures with an operational system.

We define three level of users:

1. **General User**, who uses the system mainly for rainfall-runoff, stream flow simulations.
2. **Model Developer**, who develop various distributed hydrologic models using the programming framework. They may also use it as a research tool to do hydrology simulations.
3. **Advanced Developer**, who develop and maintain the entire framework.

This document is for **Model Developers** and **Advanced Developers** only. If you are a **General User**, you should look at the RDHM User's Manual. [7]

For the **Model Developer** level, knowledge of C/Fortran programming language but not C++, is required. To work in the **Advanced Developer** level, C++ generic object-oriented programming skill is necessary.

Note that RDHM is not a Integrated Development Environment (IDE). It relies on other software packages to build the executable and debug it. Developers may work on the building system level (Makefiles) to let the compiler know how to build the executable from source files.

Chapter 2

System Requirements

Operating System
- Red Hat Enterprise Linux 4.0

Compiler
- GNU GCC/G++ 3.2.3 or later
- PGF90 4.1-2 or later

Library
- The C++ BOOST library[2] 1.34.x
- The GNU Scientific Library (GSL) 1.6 or later, www.gnu.org/software/gsl

Others Autoconf[1] 2.13, Automake[3] 1.4-p5, GNU Make[4] 3.79.1

Chapter 3

Where to Get Source Files

HL-RDHM Sources can be checkout-ed from the CVS repository on OHD Linux machines (LX1, ..., LX10), or obtained from a tarball (gorms.tar.gz). To checkout RDHM source files from OHD CVS repository, you must have access to one of OHD Linux machines.

3.1 Checkout from CVS

First, set the `CVSROOT` environmental variable to the CVS repository.

```
$ export CVSROOT=/fs/hsmb5/hydro/CVS_root
```

or you can put this line in your `.profile` file.
Then change to your working directory, for example,

```
$ cd /fs/hsmb5/hydro/users/zcui/devl/src
```

Now you can checkout the HL-RDHM source files:

```
$ cvs checkout gorms
```

You will have a directory named `gorms` containing all HL-RDHM source files. You can edit, compile the source code it contains as the usually way.

3.2 Extract from a Tarball

Use the following command sequence to extract the tarball:

```
$ cd my/working/directory
$ gzip -d gorms.tar.gz
$ tar xvf gorms.tar
```

You will see the `gorms` directory in `my/working/directory`.

Chapter 4

Installation

The program can be configured and installed using the GNU autoconfiguring and installation tool — the *configure* script provided by the Autoconf.

1. Install the C++ Boost Library (see www.boost.org) if the C++ Boost Library is not installed, see www.boost.org for details.
   Note: HL-RDHM uses 4 Boost static libraries, libboost_date_time-gcc.a, libboost_filesystem-gcc.a, libboost_program_options-gcc.a, and libboost_regex-gcc.a. However, some versions build these libraries by adding a version number and a toolset version numebr into the file names. For example, libboost_date_time-gcc.a could be named as libboost_date_time-gcc34.a which indicates that the library was built by toolset gcc3.4. For more information, please check the Library Naming on http://www.boost.org/more/getting_started/unix-variants.html.

   In this case, for the HLRDHM building script to find these libraries, a symbolic link should be created to resolve the problem. For example,

   ```
   $cd $BOOSTDIR/lib
   $ln -s libboost_date_time-gcc34.a libboost_date_time-gcc.a
   ```

2. Install the GNU Scientific Library (GSL) if it is not installed, see www.gnu.org/software/gsl

3. NOTE: the installation has to be done in the K shell environment. Installation will fail in other shell.

4. Set the `BOOSTINCLUDEDIR` environmental variable. RDHM depends on the C++ Boost library. The `BOOSTINCLUDEDIR` environmental variable tells where the C++ Boost library include file can be found.

5. Set the `BOOSTDIR` environmental variable to the boost library include path, for example, in ksh shell, use the following command, or you can set it in your `.profile` file.

   ```
   $ export BOOSTINCLUDEDIR=/YOUR/PATH/TO/BOOST/include/boost-1_34
   ```

   where `include/boost-1_32` is a subdirectory under the installation path of Boost Library version 1.32

6. Set the `BOOSTDIR` environmental variable. RDHM depends on the C++ Boost library. The `BOOSTDIR` environmental variable tells where the C++ Boost library can be found. Set the BOOSTDIR environmental variable to the boost library path, for example, in bash shell, use the following command, or you can set it in your `.profile` file.

   ```
   $ export BOOSTDIR=/YOUR/PATH/TO/BOOST
   ```

7. Set the GSLDIR environmental variable to the GSL library path, for example, in bash shell,

   ```
   $ export GSLDIR=/YOUR/PATH/TO/GSL
   ```

8. Unpack the tar ball (rdhm.tar.gz) to your local machine. If it was checked out from CVS, cd to 'gorms'

9. Configure the source code

   ```
   $ cd gorms
   $ ./bootstrap
   $ ./configure --prefix=/whre/to/install
   ```

   The `configure` script accepts many command-line flags that modified its behaviors and the configuration of your source distribution. The `--prefix` option sets where the compiled binary file, header files to be installed. The default is `/usr/local/bin`. To obtain a list of all the options that are available, type

   ```
   $ ./configure --help
   ```

   For more information on Autoconf, please see the Autoconf manual[1].

10. Now you can compile the source code. Type:

    ```
    $ make
    ```

11. If everything is OK, you can install the compiled binaries with:

    ```
    $ make install
    ```

    The final binary executables are `rdhm`, `genpar` and `cellarea` in `/where/to/install/bin`.

Chapter 5

System Design and Organization

The system was designed to be an open, flexible system that can easily incorporate various parameterizations of distributed rainfall-runoff and routing model in a common programming framework.

There are ten modules: *Utilities*, *Domain*, *Model Base*, *Input/Output*, *C/FORTRAN Programming Interface*, *HL-RMS Algorithm*, *Calibration Driver*, *Driver*, *User Models* and *Calibration Algorithm*, which are organized into a layered structure (acyclic graph) as show in Figure 5.1. The upper layer modules dependent on the lower layer modules. The lower layer module doesn't know the existence of the upper layer modules. Within each layer, each module is independent. They don't know each other. Two modules, *User Models* and *Calibration Algorithm*, are open boxes which mean that new models can be added.

5.1 Utilities

Utilities contains basic data structures such as XMRG, HRAP, and common tools.
List of source files:

1. HRAP.h, HRAP.cpp
2. XMRG.h, XMRG.cpp
3. endian.c
4. fill_miss2d.c
5. linux.h
6. models.h
7. projection.h
8. com_header.h
9. write_grid2.h

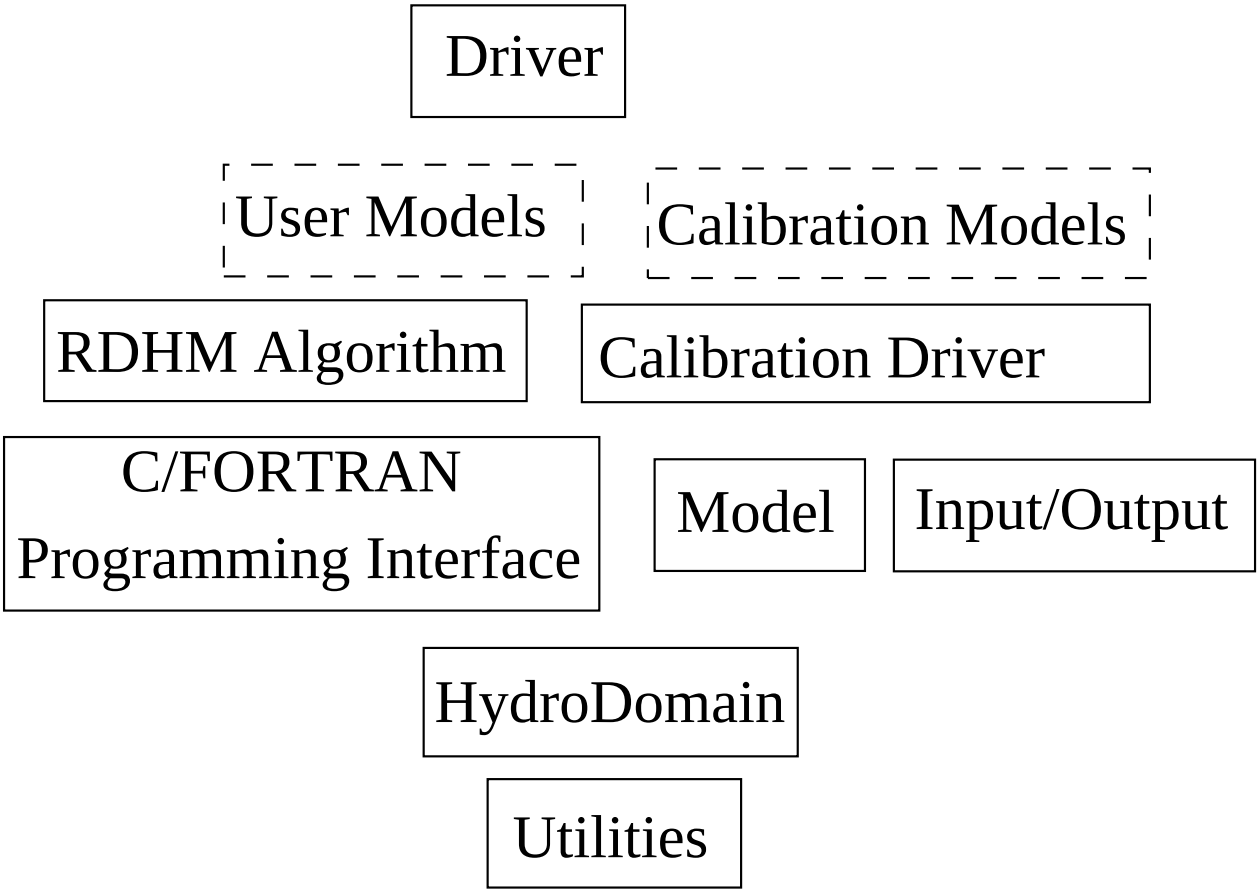

Figure 5.1: HL-RDHM modules organized in a layered structure

5.2 Domain

Domain defines and maintains a collection of pixels on which simulations will be preformed. It contains all the necessary data needed by various hydrologic models. List of source files in the *HydroDomain* directory:

1. PixelGraph.hpp
2. loadDomain.hpp
3. partially_traverse_tree.hpp
4. ConPixel.h
5. makeModelDataDescription.cpp makeModelDataDescription.h
6. parseOHDConn.cpp parseOHDConn.h

The Domain is implemented by the **PixelGraph** class. There is a **PixelGraph** object (**g**) accessible to users.

The PixelGraph object maintains all data needed during the simulation:

- contains all pixels
- maintains the connectivity of pixels
- pixel data can be retrieved
- pixel data can be set
- other information (simulation period, time step, basins, etc.)
- user defined data

There are two kinds of *Domain*, connected, and non-connected.

5.2.1 Connected Domain

In a connected *Domain*, each pixel has a flow direction to one of its neighbors (the down stream pixel). The connection is defined in a 'connectivity file', which defines the down stream pixel and other pixel information for each pixel.

5.2.2 Non-Connected Domain

In a non-connected *Domain*, the flow direction of each pixel is unknown. Pixels are not connected. Pixels are created by the program either from a given bounding box (left, right, top, bottom coordinates), or a bounding box and a mask grid layer. In the later case, only pixels with non-negative values in the bounding box are created. You can not do routing on non-connected domains.

5.3 Model Base

All models defined by users inherit the *Model Base* base class. List of source files:

1. ModelBase.hpp
2. default_model_func.h
3. default_model_func.cpp

5.4 Input/Output

Input/Output handles data Input/Output. List of source files:

1. accumulateGrids.h
2. accumulateGrids.cpp
3. getInputDeck.cpp
4. getFromDeck.hpp
5. getInputDataFromDeck.cpp getInputDataFromDeck.h
6. saveXmrgGrid.hpp
7. timeseriesOutput.hpp
8. saveXmrgGrid.hpp
9. writeTimeseriesHeader.cpp writeTimeseriesHeader.h

5.5 C/FORTRAN Programming Interface

C/FORTRAN Programming Interface provides various convenient functions for C/Fortran model developers to access various data stored in the *PixelGraph* object. List of source files:

1. ErrorCode.h
2. getSelectedVertices.hpp
3. getSelectedVerticesInconnOrder.hpp
4. getSelectionBoundary.hpp
5. partialBreadthFirstVisit.hpp
6. hlrms_interfaces.hpp
7. initPartialVisitColorMap.hpp
8. toConnOrder.hpp

5.6 HL-RMS Algorithm

Figure 5.2 shows the flow chart of the*HL-RMS Algorithm*. It consists of three stages: 1) before loop, 2) inside loop, and 3) after loop. Within each part, there are 'input', 'call model functions', and 'output' procedures. The 'call model functions' are actually provided by the **Model developer** who add their own models to the system. That is how the *HL-RMS Algorithm* can be customized to be flexible for various hydrologic models.

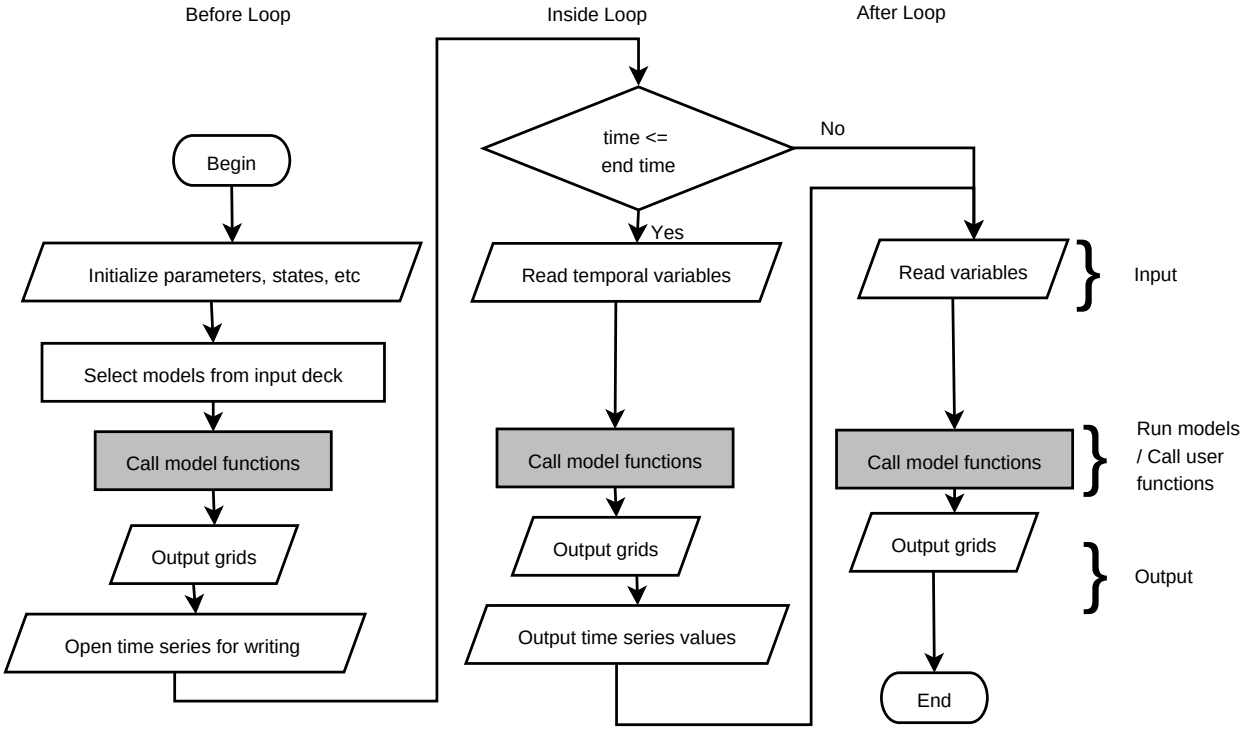

Figure 5.2: HL-RMS Algorithm Flow Chart

Before Loop reads non-temporal grid data such as parameters, initialize states, and calculates parameters to prepare the time loop for the model, output gridded data

Inside Loop reads temporal gridded data such as XMRG and air temperature, and do the simulation, output gridded data and time series

After Loop same as *Before Loop*

Model Developer must provide three functions to be called at the three gray point in the HL-RMS algorithm for the model he/she is adding. If the **Model Developer** intends to do nothing in one of them, a empty function can be provided.

5.7 Calibration Driver

Similar with the *HLRMS Algorithm*, the *Calibration Driver* calls a calibration algorithm specified in the input deck. The *Calibration Driver* flow chart is shown in Figure 5.3. The three gray boxes, 'before loop', 'inside

loop' and 'after loop' are the three model functions defined in the previous section of *HLRMS Algorithm*. The 'inside calibration loop function' box is the function defined by the calibration algorithm developer. It is invoked before the time loop by the calibration algorithm. However, the exact implementation of this function depends on the specific calibration algorithm.

The white dashed-line box represents the calibration algorithm implemented by developers. It contains a test to see whether or not the criteria is satisfied, the 'inside-calibration-loop' Function, and a time loop — the light gray box which is actually the middle part of the HLRMS algorithm shown in Figure 5.3.

The 'inside-calibration-loop' Function is useful when some initial conditions need to be reset before each calibration run. For example, reset the initial states before each repeated simulation.

5.8 Driver

This is simply the `main()` function, which calls either the *HL-RMS Algorithm* or *Calibration Driver* according to the selection in the input deck. The *HL-RMS Algorithm* or *Calibration Driver* in turn are customized by various user models or calibration algorithms. Figure 5.4 shows the *Driver* structure.

5.9 Other tools

The package also contains other two separate programs, `kwgenpar` and `cellarea`. These two programs share some common libraries with HL-RDHM. They are not part of HL-RDHM but tools to prepare data for HL-RDHM. They are here purely because they use some data structures from HL-RDHM. For detail information about these two programs, please see HL-RDHM User's Manual [7].

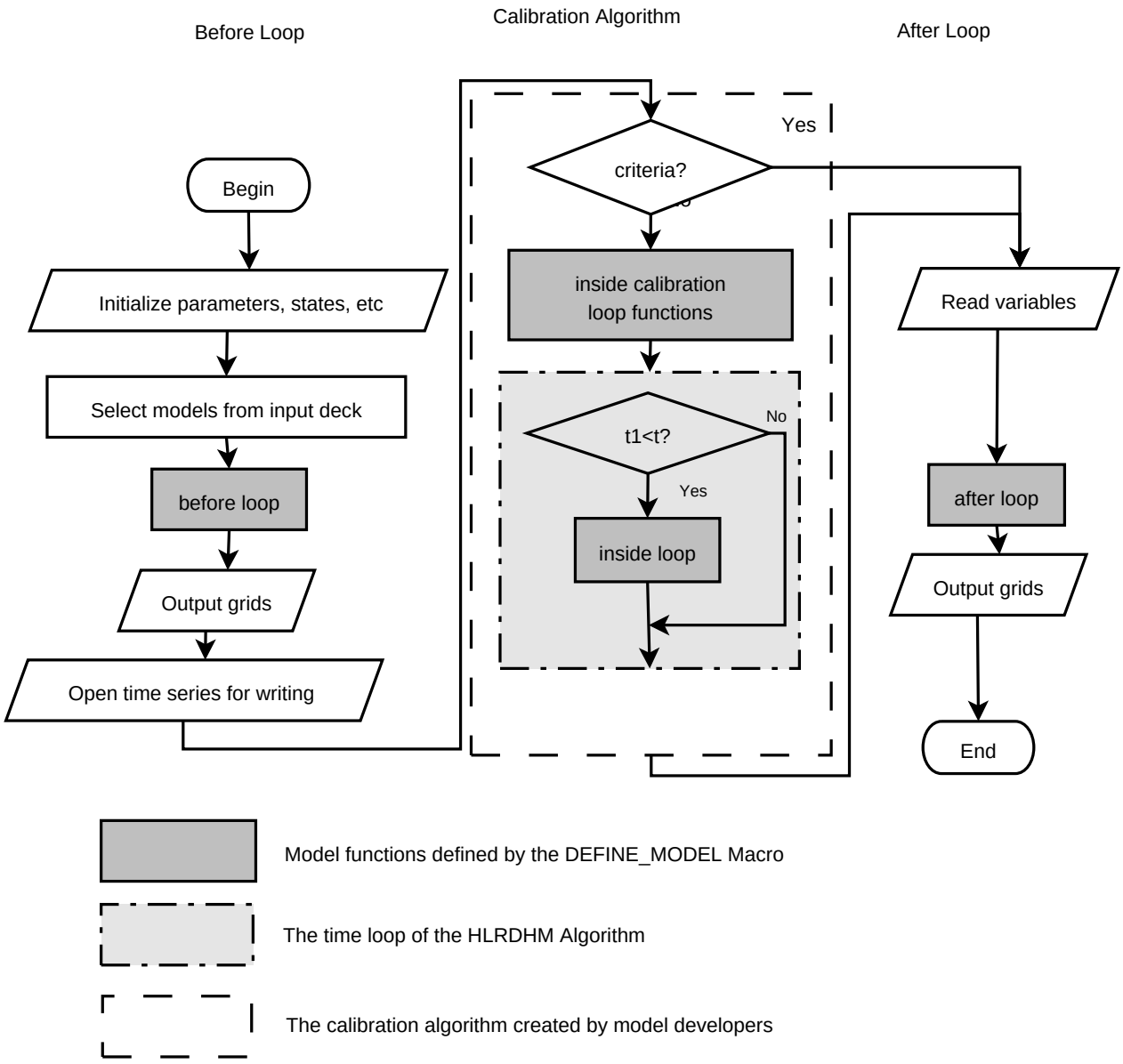

Figure 5.3: Calibration Driver Flow Chart

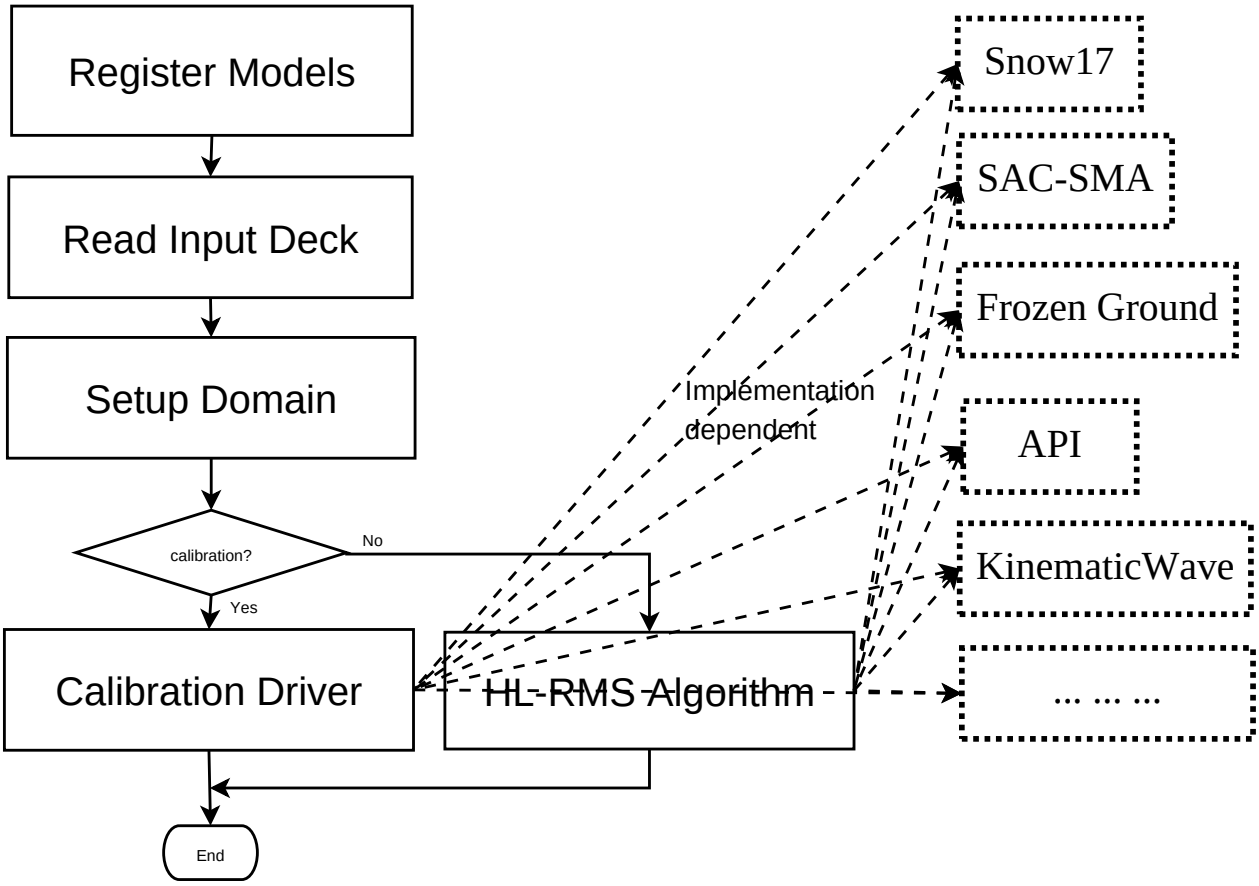

Figure 5.4: Driver calls HL-RMS algorithm or Calibration Driver, which are in turn customized by model developers

Chapter 6

Adding New Models

This *User Model* module contains sub-modules, which are provided by **Model Developers**, and therefore, it is expendable. To add new models, the **Model Developer** doesn't need to have knowledge about C++. However, Knowledge about C and Fortran are necessary. There are four steps to add a new models to the system.

1. Define the new model;
2. Describe model data;
3. Register the new model;
4. Update the building script;

6.1 Defining the New Model

The macro, `DEFINE_MODEL`, is provided to define new model. It needs four arguments, a name, and four function names. For example:

```
DEFINE_MODEL( MyModel, function1, function2,
                       function3, function4 );
```

The name, *MyModel*, will be used as the C++ class name of the model. The functions, *function1*,*function2*, and *function3* will be called by the *HL-RMS Algorithm* 'before', 'inside' and 'after' the time loop respectively as shown by the gray boxes in Figure 5.2. The *function4* is called the 'inside-calibration-loop' Function, which will be used by the *Calibration Driver*, if you don't program the calibration algorithm for your model, you can ignore this function.

Each of four functions must have the signature of:

```
int functionX( PixelGraph& g );
```

The *PixelGraph* is the class name for the simulation domain, and therefore, *g* is the object of the simulation domain, which contains all data needed by models.

Each function must also returns a value of *int* type to indicate success — zero or failure — non-zero.

For C/Fortran programmer, if you have model functions that are written in C/Fortran, then you can use the four functions as wrappers to call your specific C/Fortran functions.

Various convenient functions have been provided to access various data stored in the *PixelGraph* object *g*. These functions are defined in the *hlrms_interfaces.hpp* source file. Therefore, this file shall be included before using the functions. See the Appendix A for details on the interface.

6.2 Describing model data

Gridded data and time series input/output are handled by the *Input/Output* model. To do things correctly, the *Input/Output* must know some information about the data. The `struct ModelDataDescript` is designed for this propose. Its definition is:

```
struct ModelDataDescript {
   std::string name;
   std::string unit;
   std::string dimension;
           int accPolicy;
          int  maxMissing;
          bool fillMissing;
         float missingValue;
         float firstMissingValue;
         float noDataValue;
   std::string timeInterval;
}
```

name a text string, the name of this variable, it must be a unique name, shall not be conflict with names from other models, or the system.

unit a text string, the unit of variable

dimension a text string, the dimension of unit

accPolicy an int type, indicates how the grid value to be accumulated for each time step. It has pre-defined macros, SUM, AVERAGE, and UNKNOWN

- **SUM** Sum the value in each time interval for a time step, such as the hourly XMRG data
- **AVERAGE** Average the value in each time interval for a time step, such as the hourly air temperature grid
- **UNKNOWN** Use this value only you don't care how the value is going to be accumulated

maxMissing an int value, indicates what is the max number of missings counted by time interval of the gridded data

fillMissing a boolean value, whether fill the missing value in the gridded data file using the fill missing algorithm

missingValue a floating-point number, indicate what is the default value when a gridded data file is missing

firstMissingValue a floating-point number, some time we need use different default value for grid data missing during the first time step or during the following steps

noDataValue a floating-point number, because the XMRG like gridded data file doesn't contain the nodata value in it, it has to be specified here to retrieve the data correctly

timeInterval a text string in the format of 'HH:MM:SS.XXXX' for the time interval of the gridded data, it doesn't have to be hourly

After `DEFINE_MODEL`, use the `struct ModelDataDescript` to initialize the model's `inputBeforeLoop[]`, `inputInsideLoop[]`, and `inputAfterLoop[]` members, for example,

```
const Ohd_Hydro::ModelDataDescript
       FirstExample::inputBeforeLoop[] = {

  "sac_LZFPM",                  /* name */
  "mm",                         /* unit */
  "L",                          /* dimension */
  SUM,                          /* accumulating policy */
  0,                            /* max number of missing */
  DONT_FILLMISSING,             /* if fill missing */
  0.f,                          /* missing value */
  0.f,                          /* first missing value */
 -1.f,                          /* no data value */
  NO_TIME_INTERVAL,             /* non temporal */
```

6.3 Register the Model

The newly defined model shall be registered to the 'model database' such that it can be looked up by the name given in the input deck, that is linking the model in the input deck to the actual existing model in the program. The model is registered in the `main()` function using the `REGISTER_MODEL` macro, for example:

```
  ...
#include"MyModel.h"
  ...

  int main( int argc, char** argv )
  {
      ...
      REGISTER_MODEL( MyModel,
                      "MyModel" );
      ...
  }
```

The first argument of the macro is the class name defined by the `DEFINE_MODEL` macro, that is `MyModel`. The second argument is a text string identifier, which shall be the same as the operation name given by the input deck. Here, we use the same name for both class name and the string identifier although it is not a requirement.

6.4 Updating the Building Script

The GNU software developing tools, 'autoconf', 'automake' and 'gmake' are used to create Makefiles for the system. The objective of using GNU developing tools is to avoid the tedious process of programming Makefiles. The input file, 'configure.in' and 'Makefile.am', for 'autoconf' and 'automake' respectively, need to be updated whenever new source files are added such that the building tools be aware of the new source files. The procedures may involves editing the file 'configure.in' and the file 'Makefile.am'. These are easy steps though. See chapter 7 for details.

Chapter 7

A Step by Step Example

A very simple model is devised to show the steps of adding a new model in detail.

7.1 A Simple Model

This is not a real model, it is only for the propose of demonstrating how to add new model.

The model has the following features:

- name — FirstExample
- Two input parameters — *sac_LZFPM* and *api_AEIX*
- One calculated parameters — *calP*

$$calP = sac\_LZFPM + api\_AEIX$$

- One forcing input — *xmrg*
- One state — *ExSt*

$$ExSt = xmrg \times sac\_LZFPM$$

- One result — *res*

$$res = ExSt + calP$$

- To simplify the model, there is no 'inside-calibration-loop' Function.

7.2 Structure Design

First, we want to keep all the source files of this model in a separate directory to avoid messing up with other source files. Therefore we create a sub-directory in the `gorms` directory.

```
$ cd gorms
$ mkdir FirstExample
```

Second, we want to make a static library, called `libfirstexample.a` from which the source files are to be linked to the system.

7.2.1 Implement the Before-the-Loop Function

The Before-the-loop function can be used to calculate the parameter `calP` for other two parameters — `sac_LZFPM` and `api_AEIX`. We can name the function `FirstExampleBeforeLoop` and therefore, create a source file `FirstExampleBeforeLoop.cpp` for it.

FirstExample/FirstExampleBeforeLoop.cpp

```
/*
* filename: FirstExampleBeforeLoop.cpp
* description: calculate calP from
*              sac_LZFPM and api_AEIX
*/
#include<cstdlib>
#include"hlrms_interfaces.hpp"
using namespace Ohd_Hydro;

int FirstExampleBeforeLoop( PixelGraph& g )
{
     int error, i, npix;
     float *LZFPM, *AEIX, *calP;

     /* get number of pixels */
     if(error = get_npix( g, npix ) ) return error;

     /* allocate memory */
     LZFPM=(float*)malloc(npix*sizeof(float));
     AEIX=(float*)malloc(npix*sizeof(float));
     calP=(float*)malloc(npix*sizeof(float));

     /* get value from the PixelGraph object */
     if(error=get_value(g,"sac_LZFPM",LZFPM)) return error;
     if(error=get_value(g,"api_AEIX",AEIX)) return error;

     /* calculate calP */
     for( i = 0; i < npix; ++i ){
       calP[ i ] = LZFPM[ i ] + AEIX[ i ];
     }

     /* put calP to the PixelGraph object */
     if(error=put_value(g,calP,"calP")) return error;

     /* cleaning up */
     free( LZFPM );
     free( AEIX );
     free( calP );

```

```
40      /* must return an int */
41      return OK;
42  }
```

Following is the Line by Line explanation of the source file:

line 1 – 5 A brief description

line 6 This is the equivalent of the C *stdlib.h* header file, using *cxxxx* instead of *xxxx.h* header file in C++ according to the new ISO C++ standard

line 7 Include the `hlrms_interfaces.hpp` because we are going to use convenient functions defined in this file

line 8 Convenient functions and the *PixelGraph* class are defined in the *Ohd_Hydro* namespace, so we need add this line

line 10 As a requirement, the function signature must take one argument of type *PixelGraph* and returns an *int* as the error code

line 16 call the convenient function `get_npix` to get number of pixels

line 19 – 21 allocate memory

line 24 – 25 gets value from the *PixelGraph* object `g`

line 28 – 30 calculate `calP`

line 33 puts values of `calP` into the *PixelGraph* object `g`

line 36 – 38 cleaning up allocated memory

7.2.2 Implement the Inside-the-Loop Function

The Inside-the-Loop function is used to do the model calculation that is calculate the state `ExSt` and result `res` from `XMRG` and `calp`.

FirstExample/FirstExampleInsideLoop.cpp

```
1  /*
2  * filename: FirstExampleInside.cpp
3  * description: calculate ExSt, res from calP
4  */
5  #include<cstdlib>
6  #include"hlrms_interfaces.hpp"
7  using namespace Ohd_Hydro;
8
9  int FirstExampleInsideLoop( PixelGraph& g )
10 {
11     int error, i, npix;
```

```
12      float *LZFPM, *calP, *xmrg, *ExSt;
13      float *res;
14
15      /* get number of pixels */
16      if(error = get_npix( g, npix ) )  return error;
17
18      /* allocate memory */
19      LZFPM=(float*)malloc(npix*sizeof(float));
20      xmrg=(float*)malloc(npix*sizeof(float));
21      calP=(float*)malloc(npix*sizeof(float));
22      ExSt=(float*)malloc(npix*sizeof(float));
23      res=(float*)malloc(npix*sizeof(float));
24
25      /* get value from the PixelGraph object */
26      if(error=get_value(g,"sac_LZFPM",LZFPM)) return error;
27      if(error=get_value(g,"xmrg",xmrg)) return error;
28      if(error=get_value(g,"calP",calP)) return error;
29
30      /* calculate ExSt, res */
31      for( i = 0; i < npix; ++i ){
32         ExSt[ i ] = LZFPM[ i ] * xmrg[ i ];
33         res[ i ] = ExSt[ i ] + LZFPM[ i ];
34      }
35
36      /* update state ExSt */
37      if(error=put_value(g,ExSt,"ExSt")) return error;
38
39      /* put res to the PixelGraph object */
40      if(error=put_value(g,res,"res")) return error;
41
42      /* cleaning up */
43      free( LZFPM );
44      free( calP );
45      free( ExSt );
46      free( res );
47
48      /* must return an int */
49      return OK;
50  }
```

Following is the line by line explanation:

line 1 – 4 A brief description

line 5 This is the equivalent of the C *stdlib.h* header file, using *cxxxx* instead of *xxxx.h* header file in C++ according to the C++ standard

line 6 Include the `hlrms_interfaces.hpp` because we are going to use convenient functions defined in this file

line 7 Convenient functions and the *PixelGraph* class are defined in the *Ohd_Hydro* namespace, so we need add this line

line 9 As a requirement, the function signature must take one argument of type *PixelGraph* and returns an *int* as the error code

line 16 call the convenient function `get_npix` to get the number of pixels

line 19 – 23 allocate memory

line 26 – 28 gets value from the *PixelGraph* object `g`

line 31 – 34 calculate `ExSt` and `res`

line 37 update state `ExSt`

line 40 puts values of `res` into the *PixelGraph* object `g`

line 43 – 46 cleaning up allocated memory

7.2.3 Implement the After-the-Loop Function

There is nothing to be done after the time loop. We can use the default function `default_model_func`, which is an empty function that does nothing.

7.2.4 Implement the inside-calibration-Loop Function

Because there is no 'inside-calibration-loop' Function, we can use the default function `default_model_func` which is an empty function that does nothing.

7.3 Define the Model

Now define the model in a header file, `FirstExample.h`, to be included by the `main.cpp`. Also, we describe the model data using the `ModelDataDescript` struct here.

FirstExample/FirstExample.h

```
1  /*
2  * filename: FirstExample.h
3  * description: Define the FirstExample model
4  */
5  #include"hlrms_interfaces.hpp"
6  #include "default_model_func.h"
7
8  int FirstExampleBeforeLoop( PixelGraph& g );
9  int FirstExampleInsideLoop( PixelGraph& g );
10
```

```
DEFINE_MODEL( FirstExample,
              FirstExampleBeforeLoop,
              FirstExampleInsideLoop,
              default_model_func,
              default_model_func );

const Ohd_Hydro::ModelDataDescript
FirstExample::inputBeforeLoop[] = {

       "sac_LZFPM",      /* name */
       "mm",             /* unit */
       "L",              /* dimension */
       SUM,              /* accumulating policy */
       0,                /* max number of missing */
       DONT_FILLMISSING, /* if fill missing */
       0.f,              /* missing value */
       0.f,              /* first missing value */
       -1.f,              /* no data value */
       NO_TIME_INTERVAL, /* non temporal */

       "api_AEIX",       /* name */
       "mm",             /* unit */
       "L",              /* dimension */
       SUM,              /* accumulating policy */
       0,                /* max number of missing */
       DONT_FILLMISSING, /* if fill missing */
       0.f,              /* missing value */
       0.f,              /* first missing value */
       -1.f,              /* no data value */
       NO_TIME_INTERVAL, /* non temporal */

       "calP",            /* name */
       "mm^2",            /* unit */
       "L^2",             /* dimension */
       SUM,               /* accumulating policy */
       0,                 /* max number of missing */
       DONT_FILLMISSING,  /* if fill missing */
       0.f,               /* missing value */
       0.f,               /* first missing value */
       -1.f,              /* no data value */
       NO_TIME_INTERVAL,  /* non temporal */

       "ExSt",            /* name */
       "precent",         /* unit */
       "N/A",             /* dimension */
       AVERAGE,           /* accumulating policy */
```

```
57          UNLIMITED_MISSING, /* allow missing */
58          DONT_FILLMISSING,  /* don't fill missing */
59          0.f,               /* missing value */
60          -1.f,              /* missing value for
61          * the first time step
62          */
63          -1.f,              /* no data value */
64          "1:00:00"          /* 1 hr time interval */
65          };
66
67  const int FirstExample::numOfInputBeforeLoop = 4;
68
69  const Ohd_Hydro::ModelDataDescript
70  FirstExample::inputInsideLoop[] = {
71
72          "xmrg",
73          "mm",
74          "L",
75          SUM,
76          UNLIMITED_MISSING,
77          DONT_FILLMISSING,
78          0.f,
79          0.f,
80          -1.f,
81          "1:00:00" };
82
83  const int FirstExample::numOfInputInsideLoop = 1;
84
85  const Ohd_Hydro::ModelDataDescript
86  FirstExample::inputAfterLoop[]={};
87
88  const int FirstExample::numOfInputAfterLoop = 0;
```

Line by Line explanation is as following:

line 1 – 4 A brief description

line 5 Include the `hlrms_interfaces.hpp` because we are going to use definitions in this file

line 6 `default_model_func` is declared here

line 8 – 9 declare `FirstExampleBeforeLoop` and `FirstExampleInsideLoop`

line 11 – 15 Define the model class `FirstExample` using the `DEFINE_MODEL` Macro. The 'before-the-loop function' name is `FirstExampleBeforeLoop`, and the 'inside-the-loop function' name is `FirstExampleInsideLoop`, the 'after-the-loop function' is `default_model_func`

line 17 – 65 populate the `FirstExample` member `inputBeforeLoop[]` using the struct `ModelDataDescript` defined in the `Ohd_Hydro` namespace.

line 67 set the `numOfInputBeforeLoop` member to be 4, there are 4 variables need to be initialized before the time loop — `sac_LZFPM`, `api_AEIX`, `calP`, `ExSt`, that is the number of elements of the `inputBeforeLoop[]` member

line 69 – 81 populate the `FirstExample` member `inputInsideLoop[]` using the struct `ModelDataDescript` defined in the `Ohd_Hydro` namespace.

line 83 set the `numOfInputInsideLoop` member to be 1, there are only variable need to be initialized inside the time loop — `xmrg`, that is the number of elements of the `inputInsideLoop[]` member

line 85 – 86 populate the `FirstExample` member `inputAfterLoop[]` using the struct `ModelDataDescript` defined in the `Ohd_Hydro` namespace.

line 88 set the `numOfInputAfterLoop` member to be 0, there is no variables need to be initialized inside the time loop — `xmrg`, that is the number of elements of the `inputAfterLoop[]` member is 0

7.4 Register the Model

The newly defined model, *FirstExample*, shall be registered in the **Driver**, that is the `main()` function. Here, we show only a part of source file of `main.cpp`, because it's size and we just need to see only a small part to show how models are registered.

driver/main.cpp

```
1       ...
2     #include"FristExample.h"
3       ...
4     int main( int argc, char** argv )
5     {
6       ...
7        REGISTER_MODEL( FirstExample,
8                        "FirstExample" );
9        REGISTER_MODEL( MyModel,  "MyModel" );
10       REGISTER_MODEL( sac,  "sac" );
11       REGISTER_MODEL( frz,  "frz" );
12       REGISTER_MODEL( snow17,  "snow17" );
13       REGISTER_MODEL( api,  "api" );
14       REGISTER_MODEL( rutpix7,  "rutpix7" );
15       REGISTER_MODEL( rutpix9,  "rutpix9" );
16      ...
17    }
```

Line by line explanation

line 2 include the `FirstExample.h`, where the model *FirstExample* is defined

line 7 – 8 Register the model with the text identification of `FristExample`

line 9 – 15 Register other models

7.5 Update the Building Script

7.5.1 Create the FirstExample/Makefile.am

In the `FirstExample` directory, create the `Makefile.am` file.

FirstExample/Makefile.am

```
1  AM_CXXFLAGS                    = -Wall -I$(top_builddir)                \
2                                   -I$(BOOSTDIR)/include/boost-1_32       \
3                                   -I$(top_builddir)/HydroDomain          \
4                                   -I$(top_builddir)/ModelBase            \
5                                   -I$(top_builddir)/Utilities            \
6                                   -I$(top_builddir)/Interface            \
7                                   -I$(top_builddir)/Input                \
8                                   -I$(top_builddir)/Output               \
9                                   -I$(top_builddir)/hlrms                \
10                                  -I$(top_srcdir)                        \
11                                  -I.
12 lib_LIBRARIES                    = libfirstexample.a
13 libfirstexample_a_SOURCES        = FirstExampleBeforeLoop.cpp    \
14                                    FirstExampleInsideLoop.cpp
15
```

line 1 – 11 list the complier options and include paths that will be inserted to the `-I` flags of your compiler

line 12 specifies the name of the library we are building — `libfirsexample.a`

line 13-14 lists the source files that compose the library. Note that if the name of the library is `libfoo.a`, the prefix that appears in the variables that are related with the library is `libfoo_a_`

7.5.2 Update the driver/Makefile.am

Following is the complete content of the file `driver/Makefile.am`. Two lines, Line 24 and Line 39 were added. Line 24 was added to the `INCLUDES` variable to include `FirstExample`. Line 39 was been added to link to the library `libfirstexample.a`

driver/Makefile.am

```
1  MAINTAINERCLEANFILES = Makefile.in
2
3  AM_CXXFLAGS = -Wall                                                  \
4                    -I$(top_builddir) -I$(top_builddir)/HydroDomain    \
5                    -I$(top_srcdir)                                    \
```

```
                        -I$(BOOSTDIR)/include/boost-1_32          \
                        -I../loki                                 \
                        -I../HydroDomain                          \
                        -I../ModelBase                            \
                        -I../Utilities                            \
                        -I../Interface                            \
                        -I../Input                                \
                        -I../Output                               \
                        -I../myModel                              \
                        -I../hlrms                                \
                        -I../calib                                \
                        -I../api                                  \
                        -I../sac                                  \
                        -I../calsac                               \
                        -I../frz                                  \
                        -I../snow17                               \
                        -I../rutpix                               \
                        -I../calrutpix                            \
                        -I../FirstExample                         \
                        -I../thornthwaite                         \
                        -I../sacped                               \
                        -I../func_opt                             \
                        -I../sls                                  \
                        -I.

noinst_HEADERS  =

bin_PROGRAMS    = rdhm
rdhm_SOURCES    = main.cpp
rdhm_LDADD      = -L$(BOOSTDIR)/lib                                  \
                  -L/fs/opt/pgi/linux86/lib                          \
                  $(top_builddir)/sacped/libsacped.a                 \
                  $(top_builddir)/thornthwaite/libthornthwaite.a     \
                  $(top_builddir)/FirstExample/libfirstexample.a     \
                  $(top_builddir)/calrutpix/libcalrutpix.a           \
                  $(top_builddir)/rutpix/librutpix.a                 \
                  $(top_builddir)/api/libapi.a                       \
                  $(top_builddir)/snow17/libsnow17.a                 \
                  $(top_builddir)/sac/libsac.a                       \
                  $(top_builddir)/calsac/libcalsac.a                 \
                  $(top_builddir)/frz/libfrz.a                       \
                  $(top_builddir)/func_opt/libfuncOpt.a              \
                  $(top_builddir)/sls/libsls.a                       \
```

```
                $(top_builddir)/hlrms/libhlrms.a                    \
                $(top_builddir)/calib/libcalib.a                    \
                $(top_builddir)/Output/liboutput.a                  \
                $(top_builddir)/Input/libinput.a                    \
                $(top_builddir)/ModelBase/libmodel.a                \
                $(top_builddir)/Utilities/libutil.a                 \
                $(top_builddir)/HydroDomain/libdomain.a             \
                /usr/lib/libz.a                                     \
                /usr/lib/libm.a                                     \
                $(BOOSTDIR)/lib/libboost_date_time-gcc-1_32.a        \
                $(BOOSTDIR)/lib/libboost_filesystem-gcc.a            \
                $(BOOSTDIR)/lib/libboost_program_options-gcc.a       \
                $(BOOSTDIR)/lib/libboost_regex-gcc-1_32.a            \
                 -lg2c -lstdc++                                     \
                -lc -lgcc -lpgf90 -lpgf90_rpm1 -lpgf902 -lpgf90rtl  \
                -lpgftnrtl -lpgc
```

7.5.3 Update the Makefile.am

Now add `FirstExample` to the `SUBDIRS` variable in the Makefile.am of the top level (`gorms`) directory.

Makefile.am

```
auxdir                  = @ac_aux_dir@
AUX_DIST                = $(auxdir)/install-sh $(auxdir)/missing \
                          $(auxdir)/mkinstalldirs
AUX_DIST_EXTRA          = $(auxdir)/readline.m4 $(auxdir)/sys_errlist.m4 \
                          $(auxdir)/sys_siglist.m4
EXTRA_DIST              = bootstrap

AUTOMAKE_OPTIONS        = foreign
SUBDIRS                 = Utilities         \
                          HydroDomain       \
                          ModelBase         \
                          Input             \
                          Output            \
                          hlrms             \
                          sls               \
                          calib             \
                          sac               \
                          calsac            \
                          frz               \
                          api               \
                          snow17            \
                          rutpix            \
                          calrutpix         \
```

```
24                              FirstExample       \

25                            thornthwaite       \
26                            sacped             \
27                            func_opt           \
28                            driver             \
29                            kwgenpar           \
30                            cellareas
31  MAINTAINERCLEANFILES    = Makefile.in aclocal.m4 configure  \
32                            driver/config-h.in                \
33                            driver/stamp-h.in
34
35  ACLOCAL                 = aclocal -I $(auxdir)
36
37  dist-hook:
38          (cd $(distdir) && mkdir $(auxdir))
39          for file in $(AUX_DIST) $(AUX_DIST_EXTRA); do \
40            cp $$file $(distdir)/$$file; \
41          done
```

The directory name `FirstExample` is added on line 24. Note that it must be added before the `driver`, because `libfirstexample.a` must be built before the executable is built, i.e. the `driver`.

7.5.4 Update the configure.in

The `configure.in` is the input file of Autoconf. Autoconf creates Makefiles according to the content of `configure.in`. Because a new directory `FirstExample` has been created, and an extra Makefile needs to be created in that directory, the `configure.in` file needs to be updated too.

configrue.in

```
1  AC_INIT([driver/main.cpp], [2.0], [bug@address])
2  AC_CONFIG_AUX_DIR(config)
3  AM_CONFIG_HEADER(driver/config.h:driver/config-h.in)
4  AM_INIT_AUTOMAKE(rdhm, 2.0)
5
6  CFLAGS="-O3"
7
8  CXXFLAGS="-O3"
9  FCFLAGS="-O2"
10 AC_SUBST( CFLAGS )
11 AC_SUBST( CXXFLAGS )
12 AC_SUBST( FCFLAGS )
13 AC_PROG_CC
14 AC_PROG_CXX
15 AC_SUBST( CXX )
16 AC_PROG_FC( pgf90 )
```

```
17 AC_PROG_RANLIB
18
19 AC_OUTPUT(Makefile                                          \
20           driver/Makefile                                   \
21           HydroDomain/Makefile                              \
22           ModelBase/Makefile                                \
23           Utilities/Makefile                                \
24           Input/Makefile                                    \
25           Output/Makefile                                   \
26           hlrms/Makefile                                    \
27           calib/Makefile                                    \
28           sac/Makefile                                      \
29           calsac/Makefile                                   \
30           sls/Makefile                                      \
31           func_opt/Makefile                                 \
32           frz/Makefile                                      \
33           snow17/Makefile                                   \
34           api/Makefile                                      \
35           rutpix/Makefile                                   \
36           calrutpix/Makefile                                \
37            FirstExample/Makefile                                 \
38           thornthwaite/Makefile                             \
39           sacped/Makefile                                   \
40           kwgenpar/Makefile                                 \
41           cellareas/Makefile)
```

The added line 51 tells the Autoconf that a new Makefile should be created in the sub-directory `FirstExample`.

7.5.5 Create the autoconfig Script

Now you are ready to run the `bootstrap` script in the top level directory to update the `configure` script.

```
% bootstrap
```

When you open the shell script `bootsrap` using an editor, you will see that running the above command is equivalent to:

```
% autoreconf -fvi
```

Now the `configure` script is updated it is ready to build the executable — `rdhm`. Please see chapter 4 on how to use the `configure` script and the `make` program to build the executable.

Chapter 8

Adding New Calibration Algorithms

To add a new calibration algorithm, the developer has to have knowledge of C++. The structure of the *Calibration Driver* is shown in Figure 5.3. The added calibration algorithm is the middle part, the dashed-line box, in the figure. The detailed implementation in the dashed-line box dependents on the specific calibration algorithm. To create a new calibration algorithm, that is to implement the dashed-line box in Figure 5.3, three steps need to be taken.

1. Create a function that implements the calibration algorithm;
2. Define the calibration algorithm;
3. Register the calibration algorithm;
4. Update the building scripts;

8.1 Create a function that implements the calibration algorithm

This function is a C++ function that implements the specific calibration algorithm.
This function must have the signature of:

```
int calibrationFunctionName( PixelGraph& g,
                             DECK const& deck,
                             std::vector< HydroModelPtr >& selModels,
                             GridData1D& gd1d,
                             ModelPropertyMap& mptBeforeLoop ,
                             ModelPropertyMap& mptInsideLoop ,
                             ModelPropertyMap& mptAfterLoop);
```

At this step, please note that you may also want to create or update the 'inside-calibration-loop' function for the model you are going to calibrate.

8.2 Define the calibration algorithm

The newly created calibration has to be defined as a C++ class before HL-RDHM can use it. A Macro is provided to make this step easier. For example,

```
DEFINE_CALIB( CalibClass, calibrationFunctionName );
```

8.3 Register the calibration algorithm

The calibration algorithm can be registered by the Macro `REGISTER_CALIB` using a string identifier in the `main(int argc, char** argv)` function. After registration, the string identifier for the calibration algorithm is available for user to use in the input deck.
For example,

```
REGISTER_CALIB( CalibClass, "calibID" );
```

In this case, the string identifier `calibID` can be selected by the entry `calibration=calibID`.

8.4 Update the building scripts

Following similar steps in 7.5 to update the building scripts.

Appendix A

C/FORTRAN Programming Interfaces

The C/FORTRAN Programming Interfaces are a set of macros and convenient functions in C syntax to access various variables stored in the `PixelGraph` object.
All the convenient functions have the following format:

```
int xxxx( PixelGraph& g, ... )
```

It takes at least two arguments, one of them is the `PixelGraph` object. It returns a `int` type value with non-zero value indicating an error. This value can be compared with the error code in Appendix B to find out the type of the error.

All the convenient functions are defined in the
`Interface/hlrms_interfaces.hpp` source file.

Note that when a pointer type data is passed, make sure it is properly initialized. The convenient functions do not check the memory bound of the passed pointer. It is the user's responsibility to make sure the enough memory has been allocated.

A.1 getNumOfPixels

Get the number of pixels in basin 'id'.

SIGNATURE

```
int getNumOfPixels( GraphType& g,
       std::string const& id, SizeType& b )
```

IN

- g — the simulation domain object
- id — the basin id

OUT

- b — number of pixels for basin 'id'

RETURN

- OK
- BASIN_NOT_FOUND

A.2 getNumOfPixelsIncludingNested

Get the total number of pixels in basin 'id' including upstream basins.

SIGNATURE

```
int getNumOfPixelsIncludingNested(
              GraphType& g,
        std::string const& id, SizeType& b )
```

IN

- g — the simulation domain object
- id — the basin id

OUT

- b — total number of pixels for basin 'id' and its upstream basins.

RETURN

- OK
- BASIN_NOT_FOUND

A.3 get_npix

Get the total number of pixels in the simulation domain.

SIGNATURE

```
int get_npix(GraphType& g,SizeType& npix)
```

IN

- g — the simulation domain object

OUT

- npix — total number of pixels for all basins

RETURN

- OK
- BASIN_NOT_FOUND

A.4 get_nfpt

Get the total number of selected basins in the simulation domain.

SIGNATURE

```
int get_nfpt(GraphType& g,SizeType& nfpt)
```

IN

- g — the simulation domain object

OUT

- nfpt — total number of selected basins

RETURN

- OK
- ERROR

A.5 get_num_fpt

Get the basin sequence number for each pixel. The basin sequence number starts from 1.

SIGNATURE

```
int get_num_fpt(GraphType& g,SizeType* num_fpt)
```

IN

- g — the simulation domain object

OUT

- num_fpt — basin sequence number of each pixel, array of size `npix`.

RETURN

- OK
- ERROR

A.6 get_listfpt

Get the outlet pixel sequence number for each basin. The pixel sequence number starts from 1. A `PIXELS_ARE_NOT_CONNECTED` error will be return if the domain is not connected.

SIGNATURE

```
int get_listfpt(GraphType& g,SizeType* listfpt)
```

IN

- g — the simulation domain object

OUT

- num_fpt — outlet pixel sequence number, array of size `nfpt`.

RETURN

- OK
- ERROR
- PIXELS_ARE_NOT_CONNECTED

A.7 get_row_col

Get the row and col number of each pixel in the 2d array boundary of HRAP domain.

SIGNATURE

```
int get_row_col( GraphType& g, SizeType* row,
       SizeType* col, float const& cellSize )
```

IN

- g — the simulation domain object
- cellSize — the pixel size in HRAP unit

OUT

- row — row number of each pixel, array of size `npix`.
- col — column number of each pixel, array of size `npix`.

RETURN

- OK
- ERROR

A.8 get_idown

Get the down stream pixel sequence number of each pixel.
A `PIXELS_ARE_NOT_CONNECTED` error will be returned if the domain is not connected.

SIGNATURE

```
int get_idown(GraphType& g,SizeType* idown)
```

IN

- g — the simulation domain object

OUT

- idown — down stream pixel sequence number of each pixel, array of size `npix`.

RETURN

- OK
- PIXELS_ARE_NOT_CONNECTED
- PIXEL_NOT_FOUND

A.9 get_pixarea_KM2

Get the pixel area in KM^2 for each pixel.

SIGNATURE

```
int get_pixarea_KM2(GraphType& g,AreaType* pixarea)
```

IN

- g — the simulation domain object

OUT

- pixarea — area in KM^2 of each pixel, array of size `npix`.

RETURN

- OK
- UNIT_ERROR
- PROPERTY_NOT_FOUND;

A.10 get_length_M

Get the pixel drainage length in meters.

SIGNATURE

```
int get_pixarea_KM2( GraphType& g,
          LengthType* length )
```

IN

- g — the simulation domain object

OUT

- length — the pixel drainage length in meters array of size `npix`.

RETURN

- OK
- UNIT_ERROR
- PROPERTY_NOT_FOUND;

A.11 get_fspix_KM2

Get the upstream drainage area for each pixel in KM^2.
Returns a `PIXELS_ARE_NOT_CONNECTED` error if the domain is not connected.

SIGNATURE

```
int get_fspix_KM2( GraphType& g, AreaType* fspix )
```

IN

- g — the simulation domain object

OUT

- fspix — the upstream pixel drainage area in KM^2. array of size `npix`.

RETURN

- OK
- PIXELS_ARE_NOT_CONNECTED

A.12 get_fabove_KM2

Get the upstream drainage area for each basin in KM^2.
Returns a `PIXELS_ARE_NOT_CONNECTED` error if the domain is not connected.

SIGNATURE

```
int get_fabove_KM2( GraphType& g,
          AreaType* fabove )
```

IN

- g — the simulation domain object

OUT

- fabove — the upstream pixel drainage area in KM^2. array of size `nfpt`.

RETURN

- OK
- PIXELS_ARE_NOT_CONNECTED
- BASIN_NOT_FOUND

A.13 get_ndx

Get maximum number of 'ndx'.

SIGNATURE

```
int get_ndx( GraphType& g,T const& maxndx
LengthType const& dxmax,  T& ndx )
```

IN

- g — the simulation domain object
- maxndx — max ndx
- dxmax — max dx

OUT

- ndx — maximum number of 'ndx'

RETURN

- OK
- PIXELS_ARE_NOT_CONNECTED
- BASIN_NOT_FOUND

A.14 get_basin_sum

Get the summation of a value over a given basin or a given sub window in the unconnected domain

SIGNATURE

```
int get_basin_sum( GraphType& g,const char* id
const char* name, T& sum )
```

IN

- g — the simulation domain object
- id — the basin id
- name — the name of the value going to be summed

OUT

- sum — the sum of the given name over the basin

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.15 get_basin_sum_including_nested

Get the summation of a value over a given basin including its upstream basins or a given sub window in the unconnected domain

SIGNATURE

```
int get_basin_sum_including_nested
( GraphType& g,const char* id
const char* name, T& sum )
```

IN

- g — the simulation domain object
- id — the basin id
- name — the name of the value going to be summed

OUT

- sum — the sum of the given name over the basin and its upstream basins

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.16 get_basin_ave

Get the average of a value over a given basin or a given sub window in the unconnected domain

SIGNATURE

```
int get_basin_ave( GraphType& g,const char* id
const char* name, T& ave )
```

IN

- g — the simulation domain object
- id — the basin id
- name — the name of the value going to be summed

OUT

- ave — the average of the given name value over the basin

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.17 get_basin_ave_including_nested

Get the average of a value over a given basin including its upstream basins or a given sub window in the unconnected domain

SIGNATURE

```
int get_basin_ave( GraphType& g,const char* id
const char* name, T& ave )
```

IN

- g — the simulation domain object

- id — the basin id
- name — the name of the value going to be summed

OUT

- ave — the average of the given name value over the basin and its upstream basins

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.18 put_basin_value — the general version

Set the given basin or sub window in the unconnected domain a uniform value with the given value.

SIGNATURE

```
int put_basin_value( GraphType& g,const char* id
T const& value, ModelDataDescript const& des )
```

IN

- g — the simulation domain object
- id — the basin id
- value — value to be set to the basin
- des — the data description about the value

OUT

- g — the simulation domain object, in which the value is stored

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.19 put_basin_value — the XMRG version

Set the given basin or sub window in the unconnected domain a value from the given XMRG gridded data.

SIGNATURE

```
int put_basin_value( GraphType& g,const char* id
XMRG& value, ModelDataDescript const& des )
```

IN

- g — the simulation domain object
- id — the basin id
- value — the XMRG data to be set to the basin
- des — the data description about the data in the XMRG

OUT

- g — the simulation domain object, in which the value is stored

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND
- ERROR

A.20 multiply_basin_factor

Multiply the value of a given basin by a given factor.

SIGNATURE

```
int put_basin_value( GraphType& g,const char* id
const char* name, T const& factor,
              T const& noDataValue)
```

IN

- g — the simulation domain object
- id — the basin id
- name — the name of the multiplied value
- factor — the factor
- noDataValue — the nodata value, if the value is nodata value it will not be multiplied.

OUT

- g — the simulation domain object, in which the multiplied value is stored

RETURN

- OK
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND

A.21 get_values

Get values for all pixels for given names. The values are in a one dimensional array as the same format as in the old hlrms.
value = value1(Pix1), value1(Pix2), ... , value2(Pix1), value2(Pix2), ...

SIGNATURE

```
int get_values( GraphType& g,const char** names
size_t numOfNames,T* values)
```

IN

- g — the simulation domain object
- names — the list of variable names to be extracted
- numOfNames — the number of names in the list it will not be multiplied.

OUT

- values — one dimensional array of size $numOfNames \times npix$

RETURN

- OK
- ERROR
- POINTER_NOT_INITIALIZED
- PROPERTY_NOT_FOUND

A.22 get_value

Get the given variable by name for all basins.

SIGNATURE

```
int get_value( GraphType& g,const char* name
        T*  values)
```

IN

- g — the simulation domain object
- name — the variable name

OUT

- values — one dimensional array of size `npix`

RETURN

- OK
- ERROR
- POINTER_NOT_INITIALIZED
- PROPERTY_NOT_FOUND

A.23 put_values — the ModelDataDescript version

Store values for all pixels for given names to the domain object. The values are in a one dimensional array as the same format as in the old hlrms.
value = value1(Pix1), value1(Pix2), ... , value2(Pix1), value2(Pix2), ...

SIGNATURE

```
int put_values( GraphType& g,size_t numOfValues
T* values, ModelDataDescript* des)
```

IN

- g — the simulation domain object
- numOfValues — the number of different kind of values
- values — the one dimensional array contains all values
- des — the array of ModelDataDescript of the data in the one dimensional array of 'values'

OUT

- g — the simulation domain object, in which the values are stored

RETURN

- OK
- ERROR
- POINTER_NOT_INITIALIZED
- PROPERTY_NOT_FOUND

A.24 put_values — the value name version

Store values for all pixels for given names to the domain object. The values are in a one dimensional array as the same format as in the old hlrms.
value = value1(Pix1), value1(Pix2), ... , value2(Pix1), value2(Pix2), ...

SIGNATURE

```
int put_values( GraphType& g,size_t numOfValues
T* values, const char** names)
```

IN

- g — the simulation domain object
- numOfValues — the number of different kind of values
- values — the one dimensional array contains all values
- names — the array of variable names in the one dimensional array of 'values'

OUT

- g — the simulation domain object, in which the values are stored

RETURN

- OK
- ERROR
- POINTER_NOT_INITIALIZED
- PROPERTY_NOT_FOUND

A.25 put_value

Store values for all pixels for a given name to the domain object. The values are in a one dimensional array.

SIGNATURE

```
int put_values( GraphType& g,
T* values, const char* name)
```

IN

- g — the simulation domain object
- values — the one dimensional array contains all values
- name — the variable names in the one dimensional array of 'values'

OUT

- g — the simulation domain object, in which the values are stored

RETURN

- OK
- ERROR
- POINTER_NOT_INITIALIZED
- PROPERTY_NOT_FOUND

A.26 put_value_to_xmrg

Select a variable by name and save to the XMRG parameter file

SIGNATURE

```
int put_value_to_xmrg( GraphType& g,
const char* name, const char* filename,
 T const& noDataValue)
```

IN

- g — the simulation domain object
- name — the variable names in the one dimensional array of 'values'
- filename — the XMRG file name
- noDataValue — the nodata value in the resulting XMRG file

OUT

- A XMRG parameter file with a filename of 'filename'

RETURN

- OK
- ERROR
- PROPERTY_NOT_FOUND

A.27 get_value_from_xmrg

get selected value from an XMRG file

SIGNATURE

```
int get_value_from_xmrg( GraphType& g,
XMRG const& xmrg
 ModelDataDescript const& des)
```

IN

- g — the simulation domain object
- xmrg — the XMRG object
- des — the array of ModelDataDescript of the data in the one dimensional array of 'values'

OUT

- g — the updated simulation domain object

RETURN

- OK
- ERROR
- PROPERTY_NOT_FOUND

A.28 get_data_descript

Get its ModelDataDescript for a given variable name.

SIGNATURE

```
int get_data_descript( GraphType& g,
const char* name, ModelDataDescript& des)
```

IN

- g — the simulation domain object
- name — the variable name

OUT

- des — the ModelDataDescript of the variable

RETURN

- OK
- MODEL_DATA_DESCRIPT_NOT_FOUND

A.29 put_array_values

Store a array to each pixel.

SIGNATURE

```
int put_array_values( GraphType& g,
size_t arraySize, T** values,
       ModelDataDescript& des)
```

IN

- g — the simulation domain object
- arraySize — the size of each array
- values — the array of array for each pixel, size of `npix`
- des — the ModelDataDescript of the variable

OUT

- g — the simulation domain object, in which the 'values' is stored

RETURN

- OK
- PROPERTY_NOT_FOUND

A.30 get_array_values

Extract the value of one dimensional array type into a two dimensional array.

SIGNATURE

```
int get_array_values( GraphType& g,
const char* name, T** values,
       size_t* eachArraySize)
```

IN

- g — the simulation domain object
- name — the name of the variable

OUT

- eachArraySize — the size of each array
- values — the array of array for each pixel, size of `npix`

RETURN

- OK
- ERROR
- PROPERTY_NOT_FOUND

A.31 get_year_month_day_hour

Get the year, month, day, hour of current simulation time step.

SIGNATURE

```
int get_year_month_day_hour( GraphType& g,
T& year, T& month,T& day, T& hour)
```

IN

- g — the simulation domain object

OUT

- year — the year (yyyy)
- month — the month (mm, 1 – 12)
- day — the day (dd, 1 – 31)
- hour — the hour (hh, 0 – 23)

RETURN

- OK
- ERROR

A.32 get_start_year_month_day_hour

Get the start year, month, day, hour of the simulation.

SIGNATURE

```
int get_start_year_month_day_hour( GraphType& g,
T& year, T& month,T& day, T& hour)
```

IN

- g — the simulation domain object

OUT

- year — the year (yyyy)
- month — the month (mm, 1 – 12)
- day — the day (dd, 1 – 31)
- hour — the hour (hh, 0 – 23)

RETURN

- OK
- ERROR

A.33 get_end_year_month_day_hour

Get the end year, month, day, hour of the simulation.

SIGNATURE

```
int get_end_year_month_day_hour( GraphType& g,
T& year, T& month,T& day, T& hour)
```

IN

- g — the simulation domain object

OUT

- year — the year (yyyy)
- month — the month (mm, 1 – 12)
- day — the day (dd, 1 – 31)
- hour — the hour (hh, 0 – 23)

RETURN

- OK
- ERROR

A.34 get_time_step_minutes

Get the simulation time step in minutes.

SIGNATURE

```
int get_time_step_minutes( GraphType& g,
          T& timeStepMin)
```

IN

- g — the simulation domain object

OUT

- timeSetpMin — the time step in minutes

RETURN

- OK
- ERROR

A.35 set_time_step_minutes

Reset the simulation time step in minutes.

SIGNATURE

```
int set_time_step_minutes( GraphType& g,
          T const& timeStepMin)
```

IN

- g — the simulation domain object
- timeSetpMin — the time step in minutes

OUT

- g — the simulation domain object

RETURN

- OK
- ERROR

A.36 get_time_int_str

Get the simulation time step in a text string.

SIGNATURE

```
int get_time_int_str( GraphType& g,
          char*  timeIntStr)
```

IN

- g — the simulation domain object

OUT

- timeIntStr — the time step in a text string of HH:MM:SS.XXX format

RETURN

- OK
- ERROR

A.37 set_time_int_str

Reset the simulation time step using a text string in ISO format.

SIGNATURE

```
int set_time_int_str( GraphType& g,
          char*  timeIntStr)
```

IN

- g — the simulation domain object
- timeIntStr — the time step in a text string of HH:MM:SS.XXX format

OUT

- g — the simulation domain object

RETURN

- OK
- ERROR

A.38 add_data_descript

Add a ModelDataDescript object to the simulation domain.

SIGNATURE

```
int add_data_descript( GraphType& g,
          ModelDataDescript const& des)
```

IN

- g — the simulation domain object
- des — the ModelDataDescript object

OUT

- g — the simulation domain object, in which the ModelDataDescript object is stored

RETURN

- OK
- ERROR

A.39 remove_data_descript

Remove a ModelDataDescript object from the simulation domain.

SIGNATURE

```
int remove_data_descript( GraphType& g,
          std::string const& des_name)
```

IN

- g — the simulation domain object
- des_name — the name member of the ModelDataDescript object

OUT

- g — the simulation domain object, in which the ModelDataDescript object is stored

RETURN

- OK
- ERROR

A.40 add_name_index

Add a variable name to the name index database of simulation domain.

SIGNATURE

```
int add_name_index( GraphType& g,
          std::string const& name)
```

IN

- g — the simulation domain object
- name — the name of the variable

OUT

- g — the simulation domain object, in which the name is stored in the name index database

RETURN

- OK
- ERROR

A.41 remove_name_index

Remove a variable name from the name index database of simulation domain.

SIGNATURE

```
int remove_name_index( GraphType& g,
          std::string const& name)
```

IN

- g — the simulation domain object
- name — the name of the variable

OUT

- g — the simulation domain object, in which the name is removed from the name index database

RETURN

- OK
- ERROR

A.42 get_user_data

Get the `user-data` entry of the input deck into an array of text string.

SIGNATURE

```
int get_user_data( GraphType& g,
          char** user_data, int& lines)
```

IN

- g — the simulation domain object
- user_data — array of text string

OUT

- user_data — the array of text string that is the contents of the input deck user-data entry
- lines — size of the array of user_data

RETURN

- OK
- ERROR

A.43 get_llx

get the lower left corner x HRAP coordinate of the selected basin boundary.

SIGNATURE

```
int get_llx( GraphType& g, T& llx)
```

IN

- g — the simulation domain object

OUT

- llx — the lower left corner x HRAP coordinate of the selected basin boundary

RETURN

- OK
- ERROR

A.44 get_urx

get the upper right corner x HRAP coordinate of the selected basin boundary.

SIGNATURE

```
int get_urx( GraphType& g, T& urx)
```

IN

- g — the simulation domain object

OUT

- urx — the upper right corner x HRAP coordinate of the selected basin boundary

RETURN

- OK
- ERROR

A.45 get_lly

get the lower left corner y HRAP coordinate of the selected basin boundary.

SIGNATURE

```
int get_lly( GraphType& g, T& lly)
```

IN

- g — the simulation domain object

OUT

- lly — the lower left corner y HRAP coordinate of the selected basin boundary

RETURN

- OK
- ERROR

A.46 get_ury

get the upper right corner y HRAP coordinate of the selected basin boundary.

SIGNATURE

```
int get_ury( GraphType& g, T& ury)
```

IN

- g — the simulation domain object

OUT

- ury — the upper right corner y HRAP coordinate of the selected basin boundary

RETURN

- OK
- ERROR

A.47 put_independent_basin_values

put values for a given independent basin, include nested upstream basins, even the basins are selected

SIGNATURE

```
int put_independent_basin_values(
 GraphType& g,
 const char* id,
std::vector< T > const& values,
ModelDataDescript const& des)
```

IN

- g — the simulation domain object
- id — the basin id
- values — the values in bread first search order
- des — the variable description

OUT

- g — the simulation domain object

RETURN

- OK
- ERROR
- PROPERTY_NOT_FOUND
- BASIN_NOT_FOUND
- BASIN_IS_NOT_INDEPENDENT

A.48 getNestedBasinPixelNumber

get number of pixels in a given basin including upstream basins.

SIGNATURE

```
int getNestedBasinPixelNumber(
 GraphType& g,
 const char* id,
SizeType& b)
```

IN

- g — the simulation domain object
- id — the basin id

OUT

- b — the number of pixels including upstream basins.

RETURN

- OK
- BASIN_NOT_FOUND
- PIXELS_ARE_NOT_CONNECTED

A.49 save_value

Save an arbitrary value to the simulation domain object.

SIGNATURE

```
int save_value(
 GraphType& g,
 const char* name,
T const& value)
```

IN

- g — the simulation domain object
- name — the identifier of the value, which is used to retrieve the value later.

OUT

- g — the simulation domain object

RETURN

- OK
- ERROR

A.50 retrieve_value

Retrieve the saved arbitrary value from the simulation domain object.

SIGNATURE

```
int retrieve_value(
 GraphType& g,
 const char* name,
T& value)
```

IN

- g — the simulation domain object
- name — the identifier of the value, which is used to retrieve the value later.

OUT

- value — the retrieved value

RETURN

- OK
- ERROR

A.51 get_selected_basins

Get selected basin ids in a string list

SIGNATURE

```
int get_selected_basins(
 GraphType const& g,
 char** basinIds)
```

IN

- g — the simulation domain object

OUT

- basinIds — the basinIds in a list

RETURN

- OK
- ERROR

A.52 get_vertex_subwin

For a given pixel, get the name of the basin or subwindow the pixel belongs to

SIGNATURE

```
int get_vertex_subwin(
vertex_descriptor const & v,
 char* sub)
```

IN

- g — the simulation domain object
- v — the vertex

OUT

- sub — the basin id which the vertex belongs to

RETURN

- OK
- ERROR

Appendix B

Error Code

The error codes are defined in `Interface/ErrorCode.h` header file.

Error Number	Error Type
0	OK
-1	ERROR
1	PROPERTY_NOT_FOUND
2	PROPERTY_SIZE_ERROR
3	BASIN_NOT_FOUND
4	POINTER_NOT_INITIALIZED
5	PIXELS_ARE_NOT_CONNECTED
6	PIXEL_NOT_FOUND
7	UNIT_ERROR
8	MODEL_DATA_DESCRIPT_NOT_FOUND
9	BASIN_SIZE_IS_ZERO
10	BASIN_IS_NOT_INDEPENDENT

Bibliography

[1] *The GNU Autoconf Manual*, http://www.gnu.org/software/autoconf/manual/autoconf-2.57/autoconf.html

[2] *The C++ Boost Library*, http://www.boost.org

[3] *GNU Automake*, http://www.gnu.org/software/automake

[4] *GNU Make*, http://www.gnu.org/software/make

[5] Koren, K., Reed, S., Smith, M., Zhang, Z., Seo, D., 2004, Hydrology laboratory research modeling system (HL-RMS) of the US national weather service, *Journal of Hydrology*, 291, 297–318

[6] Office of Hydrologic Development, 2005, Sacramento Model Enhancement To Handle Implications of Frozen Ground on Watershed Runoff, *National Weather Service/Office of Hydrologic Development, ALGORITHM DESCRIPTION DOCUMENT*

[7] Seann Reed, Victor Koren, Zhengtao Cui, Fekadu Moreda, and Lee Cajina, Hydrology Laboratory-Research Distributed Hydrologic Model (HL-RDHM) User Manual v. 2.0