Flood Forecaster Documentation and User's Guide

Scott J. Small

January 6, 2016

Contents

1 Forecaster Overview and Terminology

An *ASYNCH style simulation* is a simulation produced by the ASYNCH program. The simulation is the solution to a system of ODEs with forcings (such as rainfall, evaporation, reservoir outputs, etc). The outputs consist of hydrographs and other time series, peakflow information, and snapshots of every model state. Documentation for using ASYNCH exists. Much of the information contained there can by applied here.

A *forecast* is a special type of ASYNCH style simulation. The simulation begins with a rainfall forcing for a period of time (typically one hour). This forcing is typically some type of rainfall estimates (maybe the IFC product, or the NWS's MRMS). After this rainfall period, an intensity of 0 mm/hr is assumed, and the simulation continues for a set duration of time, called the *forecast window* (typically 10 days). In future developments, rainfall predictions may be incorporated into these forecasts. Forecasts are typically hydrographs, though other time series may be included.

A *forecaster* is a program which downloads rainfall data from a database, produces a forecast, and uploads the results to a database. Forecasters have the same general flow:

- A forecaster begins producing forecasts at a specified time. This involves downloading available rainfall estimates.
- Once a forecast is produced, a forecaster advances in time (typically one hour), and produces a new forecast with currently available rainfall estimates.
- A forecaster continues repeating this procedure until no rainfall estimates are available.

The most common reason rainfall estimates are unavailable is that the forecaster last made a forecast at a very recent time, and rainfall estimates have not been created yet. There are four different forecasters. Each differs in the output provided, as well as the flow of operation.

- ASYNCHPERSIS produces forecasts at select locations and maintains a history of these forecasts. Peakflow information at select locations is also produced for the most recent forecast.

- FORECASTER_MAPS produces forecasts at select locations, peak-flow data at select locations, and maps. A history of each is maintained.
- ASYNCHERSIS_END is similar to ASYNCHPERSIS, except the program terminates when no additional rainfall data is available.
- FORECASTER_MAPS_END is similar to FORECASTER_MAPS, except the program terminates when no additional rainfall data is available.

Some of these programs require additional inputs to run, depending upon the desired output. See Section 3. The forecasters ASYNCHPERSIS and FORECASTER_MAPS wait until rainfall estimates are available. These forecasters will only stop when requested by the user.

Every forecaster has a special forcing, referred to as the *forecast forcing*. This forcing (typically rainfall) consists of current estimates which are provided to the forecaster in realtime. The availability of this data drives a forecaster, and requires special attention.

A *forecaster group* is a collection of forecasters running on the same set of compute nodes. These are run in serial (i.e. one at a time) and are controlled by a Python script. Typically, the forecasters ASYNCHPERSIS_END and FORECASTER_MAPS_END are used in forecaster groups, as they terminate when no rainfall estimates are available.

Each forecaster in a forecaster group is repeatedly run until no forecaster makes progress (i.e. produces a forecast). When this occurs, they wait a set period of time (typically 10 minutes) before attempting to produce forecasts again.

2 The Forecaster Code

Will discuss where to find the repository of codes, needed libraries, how to compile, examples.

3 Forecaster Inputs

Forecasters require a large number of inputs to correctly produce forecasts. Many inputs can be pulled from a database with relative ease. Other inputs are best done through files. Most of these inputs only need to be modified when a significant change occurs (new parameters, new model, new database, etc).

3.1 Command Line Arguments

Each of the four forecasters takes a different collection of command line arguments.

ASYNCHPERSIS and FORECASTER_MAPS both take the arguments

- Filename of a global file (Section 3.2)
- Filename of a forecast file (Section 3.3)

ASYNCHPERSIS_END takes

- Filename of a global file (Section 3.2)
- Filename of a forecast file (Section 3.3)
- Start timestamp for all forcings (unix timestamp)
- End timestamp for the forecast forcing (unix timestamp)
- Filename for the exit file (Section 4.5)
- Timestamp for initial conditions (unix timestamp)

FORECASTER_MAPS_END takes

- Filename of a global file (Section 3.2)
- Filename of a forecast file (Section 3.3)
- Start timestamp for all forcings (unix timestamp)
- End timestamp for the forecast forcing (unix timestamp)
- Filename for the exit file (Section 4.5)
- Timestamp for initial conditions (unix timestamp)
- Flag to upload hydrograph files (if not present, assumed 0)
- Flag to upload snapshot files (if not present, assumed 0)
- Folder location of snapshot files to upload (only needed if previous flag is 1)

3.2 Global File

Running a forecaster requires a global file (see ASYNCH documentation). The global file is set up in a similar manner to a typical simulation with the ASYNCH program. a few important portions of the global file are highlighted here.

The forecast forcing should be pulled from a database (forcing flag 3). The end timestamp indicates the time at which forecasts will be produced. For the forecasters ASYNCHPERSIS and FORECASTER_MAPS, an ASYNCH style simulation begins from the start timestamp and ends at the end timestamp before forecasts are produced. The maxtime value (in first line of global file) must be the difference between the end and start timestamps, divided by 60 (to convert to minutes). For the forecasters ASYNCHPERSIS_END and FORECASTER_MAPS_END, the start timestamp must always match the end timestamp, and maxtime must always be 0.

The output time series must match the fields of the *hydroforecast_modelname* table described in Section 4.1. This is similar to writing output time series to a database with ASYNCH.

There are two options for the output peakflow function with the forecasters. ASYNCHPERSIS and ASYNCHPERSIS_END should use *Forecast*, while FORECASTER_MAPS and FORECASTER_MAPS_END should use *Forecast_Maps*. These functions will create the peakflow outputs defined in Section 4.2. Using the incorrect function will result in an error.

Forcings that are not the forecast forcing are assumed to be always available and are always active, regardless of their source (file, database, etc). If a forcing requires timestamps, the start timestamp should (probably) match the start timestamp of the forecast forcing. This is not actually necessary; each forcing will begin at their specified start timestamp. But for most problems, the start timestamps for every forcing will be the same. As usual, if the simulation or forecast goes beyond the end timestamp of a forcing, the forcing will be assumed 0. To insure a forcing is never deactivated (i.e. taken to 0), the end timestamp can be chosen as something large.

3.3 Forecast Files (.fcst)

These are ASCII files in unix format that provide additional information to the forecasters beyond what is available in the global file. Every forecaster must be run with a forecast file. The format of these files is

{model name}

{display results on IFIS flag (0 or 1)}

{index of forecast forcing in global file}

{number of precipitation values to use in a forecast}

{forecast window (in minutes)}

{database connection file for using forcing index table}

{halt filename}

#

The model name is attached to the name of every output table in a database. See Section 4. Setting the IFIS display flag to 1 causes the forecaster to call extra functions and perform additional queries to prepare the output results for use by IFIS. The index of the forecast forcing is simply a way to identify which forcing specified in the global file is used as the forecast forcing. These indices begin at 0. The next value is the minimum number of times with a forcing value (from the forecast forcing) which must be available before a forecast is made. This value is also the number of forcing values to use in each forecast. The forecast window is the length (in minutes) of the simulation for each forecast. A database connection file for using the forcing index table must be specified here in the forecast file. See Section 3.4 for information about what this file must contain. The last entry is the filename of the halt file used to determine when the forecaster should terminate. See Section 3.5.

Forecast files support commenting. A % symbol indicates the remainder of a line is to be ignored.

3.4 Forecast Forcing Index Table

The source for the forecast forcing data must come with an additional table, referred to as an *index table*. The structure of this table is very flexible, but should have as a minimum

```
CREATE TABLE tname
(
unix_time integer NOT NULL,
link_count integer DEFAULT (-1)
);
```

The table name and field names can be anything. They are used in the rainindex database connection file described below. The field *unix_time* contains a time in which rainfall data is potentially available. The field *link_count* is the number of links which have nonzero rainfall values. If this number is -1, then rainfall data is not available for that time.

The index table is used by the forecasters to determine if rainfall data is available a given time. The table is necessary because values of 0 rainfall are not necessarily stored. The forecasters need a method to distinguish between times with 0 rainfall everywhere and rainfall not yet available.

The forecaster accesses this table using a query from a database connection file (see ASYNCH documentation for a description of these files). This file is specified in the forecast file. The database connection file requires one query:

- Query to find the smallest timestamp in the index table greater than or equal to the next rainfall needed.
  - Inputs: Next timestamp of rainfall needed
  - Returned tuples: (unix_time)

If the query returns NULL, then no new rainfall data is available for the forecaster. Depending upon the implementation of index table, the *link_count* field may need to be checked for positive values to insure rainfall data is truly available.

3.5 Halt File

The halt file is used to properly terminate execution of a forecaster. This file is an ASCII file that contains a single value: either 0 or 1. Occasionally, a forecaster checks the contents of its halt file. As long as the value in the halt file is 0, the forecaster continues execution. If the value is 1, the forecaster produces the current forecast, and then begins proper shut down procedures.

The halt file is created by a forecaster, and the value in the halt file is always set to 0 initially. The user needs to modify the halt file when setting its value to 1.

3.6 Special Inputs for IFIS

If a forecaster will be used for displaying information on IFIS, an extra function must be called in the database with the output tables. This function gets stage values from the forecasted discharge rates. The function is currently called *get_stages_ifc01()*. The function name is hardcoded in the forecasters' code (check the main routine).

In addition, a PHP script is called after the stages are obtained. As of the writing of this document, the script is activated with a *wget* call to *_new_get_ifc_forecast.php* located at

http://ifisfe.its.uiowa.edu/ifc/php/acquisition/

This script moves the stages to a special table on the IFIS database. The script name and the location are hardcoded in the main routine of the forecasters.

The user probably does not need to create these, but may need to modify names in the code from time to time. Talk with Felipe or Radek about these.

4 Forecaster Outputs

Typically, output from a forecaster goes into a database. A specific table structure needs to exist for the outputs to be stored properly. Some database

functions must also exist. All output tables and functions must end with _*modelname*, where model name is specified in the forecast file.

Many output tables use a partitioned table structure. These tables are used to store historical data. One child table is needed for each day of storage. This creates a "moving window" of historical data. We call the number of days stored M (typically, M is 10 days). When data becomes more than M days old, it is deleted from the database. The deletion is done by the forecasters.

For some forecasters, output data is produced in a file to allow for checkpointing. This is typically done through the use of recovery files. See the ASYNCH documentation for details on the use of recovery files.

4.1 Hydrograph Tables

The hydrograph data from the most recent forecast is stored in a table with the definition

```
CREATE TABLE hydroforecast_modelname
(
link_id integer NOT NULL,
"time" integer NOT NULL,
discharge double precision,
baseflow double precision
);
```

This table is used for transferring forecasts to IFIS. The field *link_id* is the id for the hillslope or link. The field *"time"* is the unixtime of the *discharge* and *baseflow* values. Each of the discharges are measured in m^3/s. This table is created by the forecasters. After a forecast is produced, a call to the function

```
CREATE OR REPLACE FUNCTION
copy_to_archive_hydroforecast_modelname()
RETURNS void AS
$BODY$
INSERT INTO master_archive_hydroforecast_modelname(
link_id,time_utc,discharge,baseflow)
(SELECT link_id,to_timestamp("time"),discharge,baseflow
FROM hydroforecast_modelname);
$BODY$
LANGUAGE sql VOLATILE
COST 100;
```

occurs. This copies the time series from *hydroforecast_modelname* to *master_archive_hydroforecast_modelname*.

Every forecaster uses a partitioned set of tables archiving hydrograph forecasts. The master table should look like

```
CREATE TABLE master_archive_hydroforecast_modelname
(
link_id integer,
time_utc timestamp with time zone,
discharge double precision,
forecast_time integer,
baseflow double precision
);
```

while the child tables should take the form

```
CREATE TABLE archive_hydroforecast_modelname_num()
INHERITS (master_archive_hydroforecast_modelname);
```

The child tables are indexed from 0 to $M-1$ (this index should be used in place of *num* in the child table definition above). The field *forecast_time* is the unixtime when the forecast was made. The field *link_id* is the id for the hillslope or link. The field *time_utc* is the timestamp of the *discharge* and *baseflow* values. Each of the discharges is measured in m^3/s.

4.2 Peakflow Tables

The forecasters *ASYNCHPERSIS* and *ASYNCHPERSIS_END* keep only peakflow data for the most recent forecast. The table has the structure

```
CREATE TABLE peakforecast_modelname
(
link_id integer,
peak_time integer,
peak_discharge double precision,
start_time integer,
test_stage real
);
```

This table is truncated before any additional data is inserted. The field *link_id* is the id of the hillslope. The field *start_time* is the unixtime of when the forecast was made. The two fields *start_time* and *link_id* form a key for the table. The field *peak_time* is the unixtime of the when the peakflow occurs. The field *peak_discharge* is the value of the peakflow in m^3/s. The field *stage* is used for storing the stage corresponding to the peakflow. This value is not set by the forecasters. The table is created by the forecasters.

The forecasters *FORECASTER_MAPS* and *FORECASTER_MAPS_END* maintain a history of M days of peakflow data. These forecasters calculate peakflows over multiple intervals. Each such interval is called a *period*. The structure for the master table is

```
CREATE TABLE master_archive_peakflows_modelname
(
link_id integer,
peak_time integer,
peak_discharge double precision,
forecast_time integer,
period integer
);
```

The structure for the child tables is

```
CREATE TABLE archive_peakflows_modelname_num()
INHERITS (master_archive_peakflows_modelname);
```

Here *num* is the index of the table from 0 to $M-1$. The field *link_id* is the id of the hillslope. The field *forecast_time* is the unixtime of when the forecast was made. The field *period* indicates which leadtime the peakflow value is calculated. This is the unixtime of the beginning of the period. The three fields *forecast_time*, *link_id*, and *period* form a key for the table. The field *peak_time* is the unixtime of the when the peakflow occurs. The field *peak_discharge* is the value of the peakflow in m^3/s.

Both *FORECASTER_MAPS* and *FORECASTER_MAPS_END* calculate peakflow data over the same time periods:

- Beginning of the forecast to 1 hour
- 1 hour to 3 hours
- 3 hours to 6 hours
- 6 hours to 12 hours
- 12 hours to 1 day
- 1 day to 2 days
- 2 days to 3 days
- 3 days to 4 days
- 4 days to 5 days

4.3 Map Tables

The forecasters *FORECASTER_MAPS* and *FORECASTER_MAPS_END* maintain a history of M days of system states. The values in these tables are the states of every link in the network when the forecast is produced. These tables are model dependent. For model 262, the master table structure is

```
CREATE TABLE master_archive_maps_modelname
(
forecast_time integer,
link_id integer,
q double precision,
s double precision,
s_p double precision,
s_l double precision,
s_s double precision,
v_p double precision,
v_r double precision,
q_b double precision
);
```

while each child table has the form

```
CREATE TABLE archive_maps_modelname_num()
INHERITS (master_archive_maps_modelname);
```

The index of the child table (*num*) should range from 0 to $M - 1$. The field *forecast_ time* is the unixtime of when the forecast was made. The field *link_ id* is the id for the link or hillslope. The remaining fields are for each state in the model at a single hillslope.

4.4 Check Point Files

The forecasters ASYNCHPERSIS and FORECASTER_MAPS periodically produce recovery files. These files can be used as initial conditions to later forecaster runs. They are especially useful in the case of system failure. The two forecasters also produce a recovery file of the last system state when they terminate.

4.5 Exit Files

The two forecasters ASYNCHPERSIS_END and FORECASTER_MAPS_-END produce ASCII files called *check point files* when they terminate. These files contain the unix time for the next forecast to produce. They are intended to be used by the Python script in forecaster groups to determine if a forecaster actually produced any forecasts.

5 Starting a Forecaster or Forecaster Group

Forecasters can be run in a forecaster group, or independently. Regardless of whether a forecaster is in a group or independent, starting a forecaster requires setting up a global file (.gbl) and a forecast file (.fcst). Most of the information in the global file is identical to an ASYNCH style simulation.

5.1 Starting an Independent Forecaster

The inputs described in Section 3 must be set. In addition, the database tables and functions described in Section 4 should exist.

For the global file, two sections have a special meaning beyond their typical use. Care should be taken to insure these values are correct.

The *maxtime* is used in ASYNCH for specifying the total time to simulate. Similarly, ASYNCHPERSIS and FORECASTER_MAPS will run a simulation for maxtime (exactly like ASYNCH). When the simulation time reaches maxtime, each forecaster will begin producing forecasts as current forecast forcing values are available. Using a value of 0 for maxtime causes the forecasters to immediately begin producing forecasts. ASYNCHPERSIS_END and FORECASTER_MAPS_END cannot be run with an ASYNCH style simulation. Therefore the value of maxtime should always be set to 0 for these two forecasters.

In the global file, the forecast forcing has a start timestamp and an end timestamp, like any database forcing. For the forecasters, the end timestamp corresponds to the time when forecasts will be produced. The start timestamp corresponds to the starting timestamp for the ASYNCH style simulation. ASYNCHPERSIS_END and FORECASTER_MAPS_END ignore the start and end timestamps, so any value could be used here.

Note that the difference between the end timestamp and start timestamp (divided by 60) should always equal maxtime.

A forecaster can be launched similar to any MPI program:

mpirun -np {number of processes} {forecaster name} {arguments}

The forecaster used should be compiled according to Section 2.

5.2 Starting a Forecaster Group

The inputs described in Section 3 must be set. In addition, the database tables and functions described in Section 4 should exist. Each forecaster group has a Python script for starting and managing the forecasters in the group. The desired command line parameters should be set in the script.

Every forecaster group has a *times file*. This is an ASCII file in unix format and has the structure

{forecaster 1 start timestamp} {forecaster 1 end timestamp}

{forecaster 2 start timestamp} {forecaster 2 end timestamp}

{forecaster 3 start timestamp} {forecaster 3 end timestamp}

⋮ ⋮

The times file should have one line per forecaster in the forecaster group. These values are accessible in the Python script for launching the forecasters, and may be used as command line arguments for the forecasters. The

values in the times file are automatically updated by the Python script. *The times file should NOT be changed by the user while the forecaster group is operational.*

The Python script can be launched like any Python script. The number of MPI processes available should be passed as a command line argument. All forecasters in a forecaster group should be compiled according to Section 2.

6 Stopping a Forecaster or Forecaster Group

Forecasters and forecaster groups can be terminated like any program (ctrl-C, qdel, kill, etc). However, these methods are sloppy and can cause data loss. Stopping a forecaster or forecaster group properly is much cleaner, and will make restart much simpler. Forced stops should only be done in the situation of an error, or when a quick stop is needed.

6.1 Stopping an Individual Forecaster

Stopping any of the four forecasters can be done by setting the flag in the halt file (see Section 3.5). The forecaster will complete any forecast it is computing, and send the results to a database as normal. Once the forecast is made, the forecaster will begin its shutdown procedures.

The forecasters ASYNCHPERSIS and FORECASTER_MAPS will create an output recovery file (.rec) of the system state at the time the last forecast is made. This file can be used as initial conditions if the forecaster is to be restarted.

The forecasters ASYNCHPERSIS_END and FORECASTER_MAPS_END will eventually stop running, regardless of the contents of the halt file. If the flag in the halt file is set, these forecasters terminate execution as they normally do, although there may still be sufficient forcing data to continue with forecasts.

6.2 Stopping a Forecaster Group

Like an individual forecaster, forecaster groups have their own halt file. When the flag in a forecaster group's halt file is set, the Python script managing the group will complete the forecasts of each forecaster, then terminate.

7 Restarting a Forecaster or Forecaster Group

Occasionally, a forecaster or forecaster group may need to be restarted. This can happen due to many different reasons (change in the model, change in parameters, unexpected error, etc).

Restarting an entire forecaster group is typically easy. If a forecaster group is running, stop it using the method described in Section 6.2. Make any desired changes (global file, Python script, forecast file, times file). To begin running the forecaster group again, only the Python script for the group needs to be started.

Restarting an individual forecaster may require more effort. If running, the forecaster should be stopped (Section 6.1). Changes will (probably) be required to the global file (initial conditions and forcing timestamps, in particular) before the forecaster is started again.

8 Current Realtime Flood Forecasting Setup

This is being developed...

8.1 Data Center Virtual Machine

The realtime flood forecater displayed on IFIS is run at the data center operated by the University of Iowa ITS. Using the data center provides reliability and security. A virtual machine called *ifisff* is the host for the forecaster. It features 8 cores (Intel Xeon CPU), 24 GB memory, and 20 GB disk space. The virtual machine be accessed at the hostname

ifisff.its.uiowa.edu

The virtual machine is managed by Leandro Avila (leandro-avila@uiowa.edu). Approximately once per month, the virtual machine is restarted for software updates. Leandro will notify the owner of the virtual machine before a restart occurs. Be sure no floods are occuring at this time!

8.2 Database Tables

Input tables are available in a database on the ifisff virtual machine. Output tables are available in a database on the ifis virtual machine.

9 Errors

Typically, I check the end of the log file of each forecaster group each day. Any timestamps should be very recent. If a forecaster is executing a query, I continue checking every minute or so until it progresses. Make sure to check the output of each forecaster in a forecaster group for errors.

Occasionally, an error message will appear in an output log, or a forecaster will not behave as expected. Here are a few common errors I have encountered.

Problem Errors about database connections in the log file.

Causes Database is down, unavailable, or extremely busy. This is the most common problem. A query may also be incorrect.

How to fix Many times, nothing needs to be done to the forecasters. The problem is usually on the database side. Sometimes the connection problem is temporary, and the forecasters will fix everything on their own. Other times the database may need to be restarted, or queries need to be stopped. When this happens, the forecasters will usually recover on their own. If a query is incorrect (for example, a syntax error), the forecaster should be stopped and the query in the corresponding .dbc file fixed.

Problem The program is not making progress (i.e. a forecaster is running but the log file is not changing).

Causes Database is down, unavailable, or extremely busy. Bad model parameters. Bad rainfall values.

How to fix Check running queries on the databases to see if the forecaster's queries are actually running. In some rare cases, a forecaster will remain waiting for a query, even though the query has been killed. The forecaster will need to be killed in this situation. I believe this is an issue with libpq. If no queries are running, then the likely problem is the numerical solvers. Tiny step sizes often occur when unrealistic model parameters are used. Check that these parameters are correct (units, correct model). Also check for unrealistic (or unexpected) rainfall values, such as negative values, extremely high values, error code values instead of intensities.

Problem Too many or not enough initial conditions when using initial conditions from a database table.

Causes This usually occurs from an error in a previous forecaster run. I tend to get these errors when making changes to the forecasters (changing model parameters, input sources, etc).

How to fix If 0 initial conditions are present in the database table with initial conditions, the initial timestamp should be reduced to a time where initial conditions are available. If more initial conditions than needed are present, consider deleting the extra conditions, or modifying the timestamp for initial conditions.

Problem A long period of no rainfall is visible in the forecasts. However, rainfall did actually fall.

Causes This occurs when the rainfall subsystem is not functioning properly.

How to fix First, verify that nonzero rainfall intensities are not present in the database, but should be. If the values are not there, they must be reingested into the database table. Then the forecaster can be restarted from a timestamp before the missing rainfall data. If the time period with missing rainfall is large, consider running an ASYNCH style simulation to create initial conditions for the forecaster at a recent time.