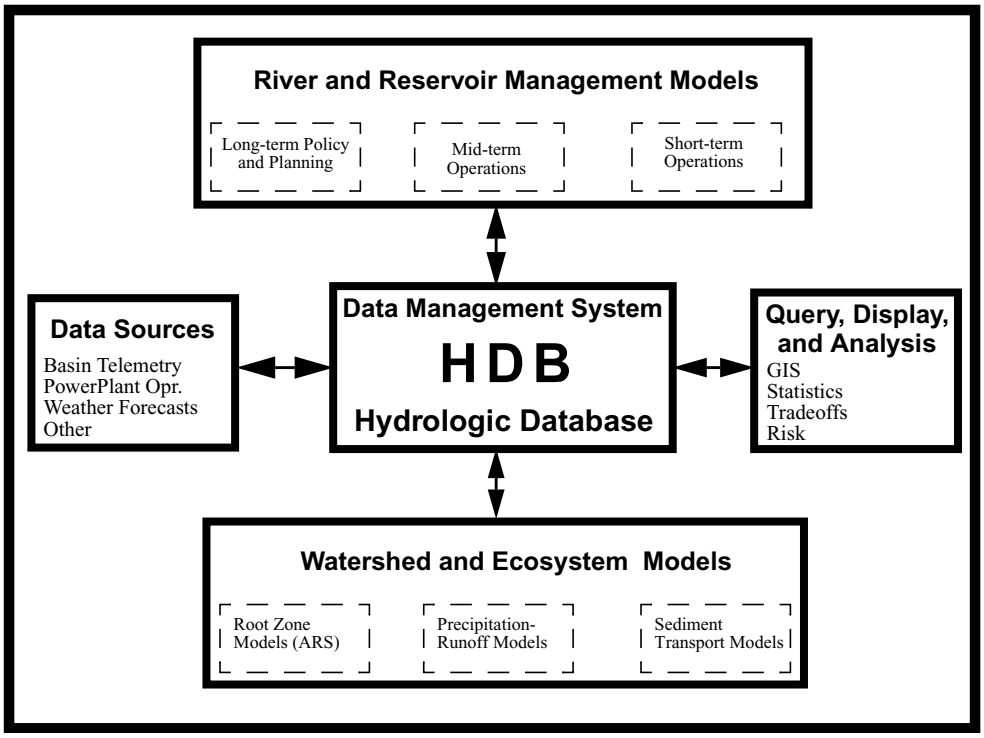
Hydrologic Database (HDB): RiverWare DMI Applications User Guide

Next Page



The Center for Advanced Decision Support for Water and Environmental Systems (CADSWES)
University of Colorado at Boulder
College of Engineering and Applied Science
Department of Civil, Environmental and Architectural Engineering
Campus Box 421
Boulder, CO 80309-0421
Phone: (303) 492-3972 Fax: (303) 492-1347
E-mail: inquiries@cadswes.colorado.edu
http://cadswes.colorado.edu

CADSWES

Section 1 Intended Audience

This document is intended for all users of HDB who are transferring data between RiverWare and HDB with the RiverWare Data Management Interface (DMI) applications.

Next Page

Previous Page

Section 2 Application Overview

The HDB-RiverWare DMIs provide functionality to exchange data between HDB and RiverWare models. The Input DMI allows data to be retrieved from HDB and to be loaded into the object slots of a RiverWare model. The output DMI allows object/slot data from RiverWare to be stored in HDB.

The HDB-RiverWare DMIs have a graphical interface when the information exchanged between HDB and RiverWare pertains to modeled data. In HDB model runs are identified by a unique model_run_id. The interface component of the HDB-RiverWare DMIs allows a user to search and browse the database to associate the data to be exchanged with one or more model_run_ids.

Next Page

Previous Page

Section 3 Permissions

The DMI code runs as the user named app_user and invokes the role named app_role to permit queries and insert, update, and delete operations on table r_base and the model tables.

Next Page

Previous Page

Section 4 Invoking the Application

The RiverWare DMIs are stand-alone execuatables named riverwareDMI.In and riverwareDMI.Out. They can be called directly for testing purposes if passed the appropriate arguments, but generally they are called through shell scripts that are invoked from within the RiverWare program. The shells call riverwareDMI.In for input DMIs that move data from HDB to RiverWare or riverwareDMI.Out for output DMIs that move data from RiverWare to HDB. The following shell scripts are supplied by CADSWES with HDB:

Next Page

Previous Page

DMI Shell Scripts	Description
RW.24m.InputDMI.sh	generic 24-month study input DMI.
RW.24m.OutputDMI.sh	generic 24-month study output DMI.
RW.BHOPS.InputDMI.sh	input DMI for RiverWare replacement of the daily BHOPS model
RW.BHOPS.OutputDMI.sh	output DMI for RiverWare replacement of the daily BHOPS model.
RW.UC.24m.OutputDMI.sh	output DMI customized for UC 24-month study modeling.
RW.LC.24m.OutputDMI.sh	output DMI customized for LC 24-month study modeling

From RiverWare a shell script is invoked as a DMI from the Utilities/DMI Selector menu. For an in-depth discussion of the RiverWare Data Management Interface see Section 9 of the RiverWare Simuation Training Manual.

Unlike earlier versions of the RiverWare DMIs, a ModelId must now be selected in the Executable Parameters window of the DMI's edit dialog for the DMI to execute correctly (see "ModelId Parameter" on page 11). The ModelId parameter must be configured by the user for RiverWare as a User keyword = value Pair. This model id determines what mappings are used to translate between RiverWare objects and slots, and HDB sites and datatypes.

Section 5 Inputs

Command Line

RiverWare DMIs are not typically called from the command line, but can be for testing purposes. If called from the command line, parameters must be supplied that are normally passed from the RiverWare DMI interface. These include the default parameters provided by RiverWare plus the user-supplied parameter for model id:

- control file name
- working directory
- initial date of run
- initial time of run
- end date of run
- end time of run
- timestep of run
- model id

Next Page

Previous Page

Files

For the Output DMI, data files are created by RiverWare that are then used as the data source for loading data to tables in HDB. One file is created for each Object/ Slot combination that results from control file specifications for RiverWare.

Dialog Boxes

Depending on the type of data being loaded to or from HDB, a DMI interface may appear to assist in the selection of model_run_ids that can be associated with RIverWare data in HDB. These interfaces are written in Tcl/Tk.

Database Driver Tables

Driver tables are required in HDB to allow for the proper mapping of data between RiverWare and HDB. These are discussed in detail in “Database Driver Tables” on page 6.

Environment Variables

The environment variables INPUT_DMI_PATH and OUTPUT_DMI_PATH must be set to the directory (full path) that contains the DMI executables and the executables for the Tcl/Tk interfaces. The environment variable RIVERWARE_DMI_DIR is not required, but when specified contains the path where the data files created by the DMIs are placed.

Next Page

Previous Page

Section 6 Outputs

Data Files

For the Input DMI, data files are created from HDB for input into RiverWare. One file is created for each Object/Slot combination that results from control file specifications made for RiverWare.

Next Page

Previous Page

Log and Error Files

When executing, the DMIs write essential verification and diagnostics information to both stdout and stderror. When triggering the DMIs from RiverWare, this information is echoed in the “DMI Diagnostics” window.

Note: Due to the different buffering of stdout and stderror, the order in which the information appears in the DMI Diagnostics window does not necessarily correspond to the order in which the verification and diagnostics messages are generated.

Database

The Output DMI from RiverWare can insert data into the “r_base” or the “m_” tables and can create model_run_id entries in the ref_model_run table. The Input DMI has no outputs to the database.

Section 7 Using Application Functionality

Functional Description

Next Page

Previous Page

The RiverWare DMI applications are stand-alone executables that are initiated through RiverWare by the user. Because RiverWare uses ascii files, the output DMI reads ascii data files prepared by RiverWare and inserts the data into the database. Similarly, the input DMI prepares the ascii data files with data fetched from HDB. RiverWare then reads the ascii files and uses them to initialize a model. See the RiverWare Simulation Training Manual for more information on the flow of information in RiverWare DMIs.

For each particular model, users specify which Object/Slot combinations will be handled by the DMI (i.e., transferred between the database and RiverWare) via user-specified control files. These files contain a list of Object/Slot combinations (or wildcards representing many Object/Slot combinations) which will be exported or imported through the DMI. Each model may have an associated user-specified control file; if necessary, there may be separate user-specified control files for the input and output DMIs for a particular model, depending on which Object/Slot combinations are to be input and output for a run. The format of these user specified control files is described in the RiverWare documentation.

RiverWare parses the user-specified control file, resolves all wild-cards, and creates a file named ‘metaControl’. This file resides in `$RIVERWARE_DMI_DIR/<dminame>` or, if `RIVERWARE_DMI_DIR` is not defined in the current user’s environment, in `/tmp/<dminame>`. Unlike the user-specified control file, which can have Object/Slots grouped by using wildcards, ‘metaControl’ is a list of the exact Object/Slots for which data will be transferred. Each Object/Slot entry in ‘metaControl’ is accompanied by a number of keyword/value combinations which specify relevant parameters, such as object type, the data file name, the units for the Object/Slot, as well as any optional parameters; see “Valid Optional Parameters and Parameter Values” on page 12 for a list of valid optional parameters in the user specified control files, and their meanings. The DMIs read ‘metaControl’ and perform the appropriate data transfer—the output DMI reads each data file generated by RiverWare and inserts values into the database; the input DMI fetches values from the database and creates a data file for each Object/Slot; these values will be read by RiverWare.

RiverWare <--> HDB Mappings

In order for RiverWare and HDB to exchange data, there must be some mapping between the data representations used in the two systems. That is, RiverWare is comprised of a network of objects with slots; HDB is a set of relationships involving sites and datatypes. RiverWare objects correspond roughly to HDB sites, and

RiverWare slots correspond roughly to HDB datatypes. It the responsibility of the RiverWare/HDB user to determine which Objects/Slots for a particular RiverWare model correspond to which Site/Datatypes, and to maintain and/or create the appropriate mappings. Also, RiverWare units must be mapped to HDB units. Finally, each installation of HDB which acts as a data source or sink for any given DMI must be identified. All of these mappings are stored in tables within HDB and the tables are maintained by the user through the metadata application.

- `ref_dmi_data_map`: mappings between RiverWare Object/Slots and HDB site_datatype_ids by RiverWare model id;
- `hdb_model`: mappings between particular RiverWare models and HDB model_ids;
- `hdb_dmi_unit_map`: mappings between RiverWare unit names and HDB unit_ids;
- `ref_db_list`: list of database sites to which to connect.

Next Page

Previous Page

ref_dmi_data_map

This table holds the mapping between RiverWare Object/Slots for a particular RiverWare model and HDB site_datatype_ids.

- **model_id:** HDB `model_id` for the RiverWare model.
- **object_name**: RiverWare name for object.
- **data_name**: RiverWare name for slot.
- **site_datatype_id**: HDB `site_datatype_id` for this object/slot.

hdb_model

This table maps particular RiverWare models to HDB model_ids.

- **model_id**: HDB `model_id` for the RiverWare model.
- **model_name**: Name of the RiverWare model.
- **cmmnt**: Comment to describe the model.

hdb_dmi_unit_map

This table holds the mapping between RiverWare unit names and HDB unit_ids; it includes the scale by which the HDB unit must be multiplied to arrive at the RiverWare unit.

- **pr_unit_name**: RiverWare name for unit.
- **unit_id:** HDB `unit_id` for this unit.
- **scale:** Integer value by which the HDB unit must be multiplied to arrive at the

RiverWare unit. For instance, thousand acre feet in HDB maps to the RiverWare unit acre-feet, with a scale of 1000.

ref_db_list

This table holds a list of the database sites, both local and remote, to which the DMI must connect to retrieve data.

- **session_no**: Positive integer used to identify the session for this database connection. Session_no = 1 must be set to the local database. For example, in Salt Lake City, session_no 1 must point to the Upper Colorado Hydrologic Database.
- **db_site_db_name**: Character string giving the name of the database for this session, e.g., “uchdba.”
- **db_site_code**: Character abbreviation used to identify this database, e.g., “UC”.
- **maxid:** Explained in “Keeping Records on Model_Run_Ids” on page 22.
- **max_sync_id**: Explained in “Keeping Records on Model_Run_Ids” on page 22.

Next Page

Previous Page

ModelId Parameter

A ModelId parameter must now be passed as a user-defined keyword=value pair from RiverWare when executing the DMIs. This parameter is a number that must represent a valid model_id defined in the hdb_model table in HDB. The model_id is used in the ref_dmi_data_map table to determine how RiverWare objects and slots translate to HDB sites and datatypes. The use of the model_id allows objects and slots with the same names in different models to map to different sites and datatypes in HDB.

How to create the keyword=value pair will shortly be changed to allow setup from the GUI within RiverWare. For now, an entry must be made in the RiverWare resource file (riverwareDB) to allow use of the user-defined keyword. The riverwareDB file should reside in the same directory ($RIVERWARE_HOME) as the RiverWare executable file. It is a text file and must contain the following syntax to set up the ModelId parameter:

```
<obj_type>DMI.numOpts: 1
<obj_type>DMI.key1: ModelId
<obj_type>DMI.values1: 12, 11, 8, 15
```

The first line means we are defining one keyword=value pair. The second line names the keyword for pair 1. The third line is a comma-separated list of potential values for the keyword in pair 1. In this case these numbers should be model ids that have been defined in the hdb_model table in HDB.

After the riverwareDB file has been set up, the user can create DMIs in the RiverWare interface and choose a value for the ModelId keyword for the DMI in the

Executable Parameters window of the DMI's edit dialog. Note that existing DMIs will not be affected by setting up a new keyword, so associating the keyword with existing DMIs requires that they be recreated in RiverWare. After a keyword value is selected for a DMI, that user-defined keyword value pair will be passed to the input or output DMI executable whenever the DMI is initiated from RiverWare.

DMI Control Files

Next Page

Users of the input and output DMIs need to be aware of the format of the user-specified control files, and of the optional parameters which the RiverWare<->HDB DMIs are able to process

Previous Page

Valid Optional Parameters

The optional parameters provided by RiverWare can be specified on each line in the control file. These parameters and the format of the control file is discussed in the RiverWare Simulation Training Manual.

The currently supported parameters are:

- file=<data file> - The data file the slot will import from or export to. This is typically set to ~/%o.%s to represent a file named object.slot in the default directory.
- units=<units> - The units of the data the slot will import or export. <units> must match exactly one of the units defined in RiverWare units file. The output DMI maps this value to the corresponding HDB unit representation, then verifies that the unit is appropriate for writing to HDB. If it is not, then this Object/Slot is not processed. The input DMI does not utilize this flag.
- scale=<scale> (float) - The scale of the data the slot will import or export.

Note: If either units or scale is not specified in the control file, then it defaults to the current setting in the RiverWare model. Since the setting in the model can change from invocation to invocation, it is strongly recommended that the control file always contain units and scale (even if the scale is 1.0).

- import=<fixed>|<resize> - The type of import. <fixed> leaves the size of the data series unchanged; <resize> resizes the data series to fit the data. The default is <fixed>
- flags=<true>|<false> - The flags keyword specifies whether or not SeriesSlot flag values will be exported or imported with the data. The default is <false>.
- precision=<display>|<model> - The precision of the data the slot will import or export. The <display> precision is that specified in the slot's Configuration dialog. The <model> precision is 12, the same precision as all internal calculations. The default is <display>.

Valid User Parameters and Parameter Values

A number of user parameters can be passed via the control file. These parameters are not processed by RiverWare, but are read by the DMI executable where they can modify the processing of the data. For HDB the following flags are available for use on each line of a control file:

- !load_model_data= : Indicates whether or not to select and load forecasted data from HDB into RiverWare. Used only for input DMI (model initialization) control files.

  Valid values: “Y” or “N”.

  Default: “N”

- !num_hist_timesteps= : The number of timesteps of historical (observed) data to load from HDB into RiverWare.

  Valid values: any positive integer.

  Default: 1

- !destination= : Either r or a model_run_id. The r indicates that data from the output DMI should be written as historic data to the r_base table. This option is used for RiverWare models that calculate information from observed data. A model_run_id indicates that data should be written to the (m_) tables using the given model_run_id. If all objects.slots in the control file have this flag set, the DMI user interface is skipped.

  Valid values: r or any positive integer that represents a valid model_run_id.

  Default: N/A

- !model_source= : The HDB model_run_id which will be used as the source of modeled data for the current Object/Slot. Used only by the input DMI and for Object/Slots where !load_model_data=Y.

  Valid values: any positive integer which represents a valid HDB model_run_id OR any string of 6 characters or less which the user will later map to a real model_run_id.

  Default: N/A. If not supplied, the current Object/Slot will use the default model_run_id as its source.

- !hist_source= : The HDB model_run_id which will be used as the source of observed data for the current Object/Slot. This is an override of the default, which is simply to use the data from the HDB observed (“r_”) tables. Used only by the input DMI.

  Valid values: any positive integer which represents a valid HDB model_run_id OR any string of 6 characters or less which the user will later map to a real model_run_id.

  Default: N/A. If not supplied, the current Object/Slot will use “r_” tables as the source of the data.

The following section describes in more detail the behavior of the DMI with respect to the last two flags.

Next Page

Previous Page

!model_source, and !hist_source

Valid Values Explained

As mentioned above, these flags can have a numeric value, e.g., "!model_source=12". Any numeric value specified is assumed to be the actual value of the model_run_id, and no other processing or mapping is needed.

The !model_source and !hist_source flags can also have any mnemonic character code, 6 letters or less, which identifies a historical or model data source. A distinct list of the codes used in any given control file will be generated when the input DMI is invoked. Each code will then be displayed as a line in the interface table, and the user must then specify the defining characteristics of the model_run_id to which this code must be mapped. Note that the first character in a character code cannot be numeric.

Next Page

Previous Page

Why Are These Flags Needed?

These flags are collectively known as the model source flags. The use of model source *codes* (e.g., !model_source=NWS) in the user specified control files gives users the ability to *conceptually* define a `model_run_id` without having to pin down the actual number in the control file. When the DMI runs, the user is asked to map the character code to an actual numeric ID by filling out a tabular form in a graphical user interface (GUI). In this way, users can retrieve data from a different model_run_id every month, if necessary, without ever editing the user-specified control file.

For instance, when running the 24 month study in the Colorado River Basin, most modeled values must come from the previous month's model run; this run has a different `model_run_id` every month. By specifying the correct parameters for the default (DEF) model run in the input DMI interface, the user can readily grab the right data from the database. Similarly, some slots are loaded from recently-generated inflow forecasts; the *model_run_id* for these inflows changes monthly. By specifying "`!model_source=NEW`" and then supplying the correct parameters for the NEW code in the GUI, the correct model_run_id will be accessed and the values can be pulled in. Finally, there is one `model_run_id` which is constant over time for the 24 month study. Rather than specifying a character code and being prompted at every input DMI invocation, the user specifies the actual `model_run_id` in the control file ("`!model_source=12`) and is never prompted further.

Using the DMIs

The Input DMI

Treatment of Modeled vs. Observed Data

The input DMI reads data from HDB for the object/slot combinations specified in metaControl and writes the values out to data files, one file per object/slot combination. RiverWare reads the data files generated by the DMI and uses the

specified values as input for the model.

The input DMI treats modeled (forecasted) data separately from historic (initial condition) data, because the two types of data have very different roles as model input and because they are stored in different tables in HDB. Model data can be of two types: 1) Data that the user specified for a previous model as conditions for the model to meet as it is solved, or 2) Model output calculated by RiverWare as the model was solved. The two types of model data are both stored in m_ tables and are not differentiated in any way in the database-- the last model that was run determines whether the data is user-specified or model output. User-specified data for an object/slot should be loaded from HDB into later model runs so that the user will not have to type in all the values every time the model is run. However, calculated model output should not be loaded by the DMI because this is what the model is trying to *solve* for, and loading it will result in an over-determined model. Historic data can be loaded for any object/slot without causing an over-determined model, and having the data available enables users to view it with the SCT interface. The historical data can come from the real (r_) tables, or model data tables (m_), using values generated by previous model runs.

Next Page

Previous Page

Every object/slot listed (explicitly or though wild cards) in the user-specified control file has historic data loaded by the input DMI. However, only those object/slot combinations with the keyword-value pair "!load_model_data=Y" will have model data loaded by the DMI. This enables the user to have independent switching of model data on the object/slot level.

The Input DMI Interface

As explained in a previous section, the use of model source codes in control files, combined with a graphical user interface, simplifies the process of specifying model_run_ids to be used as the source of input data.

When the Input DMI is invoked, the control file is parsed and any *non-numeric* model source codes (i.e., !hist_source and !model_source flags with character string values) are put into a list; this list will be displayed in the interface. For every distinct code in the list, users must specify the defining characteristics of the model_run_id represented by the code. For instance, maybe the code NEW is used to represent the most-recently generated hydrologic inflows. Users must input specifications which will point to the model_run_id holding this data. If specifications are too broad (too many matches) or too restrictive (zero matches), users will be prompted further, until one and only model model_run_id has been found for each code.

Interacting With the Main Interface

The main interface to the Input DMI is a single window broken into two frames. On the left hand side is a table in which users input defining characteristics of any model source codes used in the control file. There is one column for almost every column in the ref_model_run table (it makes no sense to query on some of the character columns, so they've been excluded). There is one row for every unique model source code found in the control file. Additionally, if there are Object/Slots for which !load_model_data=Y and no model_source was specified, the first row of the table will have a model source code of DEF; here users must describe the default model_run_id to be used. Arrow keys, "Return", and the mouse can all be used to

activate cells of the input table.

On the right hand side is a list of model_ids and model_names which are stored in HDB. This list is simply to support the user in filling in the model_id column.

At the bottom of the window are two buttons: “OK” will initiate retrieval of model_run_ids which match the input specifications, and “EXIT DMI” will exit the DMI and return control to RiverWare.

To fill in the defining characteristics properly:

- Specify at least one column for each model run code as a minimum search condition.
- Use date formats which are accepted by the database; for instance, “19-DEC-1996”.
- Never change the contents of the Model Source Code column.

Next Page

Previous Page

Additionally:

- It is helpful to specify the Model Id in order to immediately narrow the search.
- The Time Step field should not be used for specification, as it is a long character string
- Provide defining parameters as precise as possible to narrow the search.
- If the Model Run Id is known, specify it and nothing else.

Press “OK” when all defining characteristics have been input.

Interacting with the Auxiliary Interfaces

There are three cases to be handled once “OK” is pressed and a search for model_run_ids begins.

1. There is more than one Model Run Id for a particular model source code.
2. There are no Model Run Ids for a particular model source code.
3. There is exactly one Model Run Id for a particular model source code.

For Case 1, all Model Run Ids which match the model source code in question are displayed (with their attributes) in a table. Every Model Run Id occupies a row in the table, and each row is numbered, starting with “1”. In another window, users choose the desired Model Run Id by typing in the associated row number and pressing “OK”. If none of the Model Run Ids are correct, then users should type “0”; they will re-specify this Model Source Code when Case 2 situations are handled. This process continues until all Model Source Codes with more than one matching Model Run Id have been handled.

For Case 2, again a window similar to the initial specification window appears, except only those model source codes with no matching Model Run Ids are displayed. The user must re-specify the defining characteristics for these model source codes and press “OK” to re-initiate the search for matches.

The windows which handle Cases 1 and 2 will be recursively invoked until every model source code is matched with a single Model Run Id. When this happens (Case 3), a window notifies the user that the mapping is complete. Pressing “OK” will allow the DMI to continue; “EXIT DMI” will terminate the DMI and return control to RiverWare.

Output DMI

Treatment of Modeled vs. Observed Data

Next Page

The output DMI reads data from text files generated by RiverWare and inserts the values into HDB.

Previous Page

The output DMI does not insert any data into HDB that falls on or before the start date of the RiverWare model run. The !destination flag indicates if output DMI data goes to r_base or to model tables in HDB. Data is inserted at the database location appropriate for the Site/Datatype.

If data is going to the model tables in HDB, a user interface as described below is employed to assist the user.

The Output DMI Interface

The purpose of the Output DMI interface is to allow users to interactively find or create the model_run_id with which data output to HDB will be associated.

Interacting With the Main Interface

The main window is similar to the main Input DMI window (Section 5.2.2.1 on page 16 and Figure 3 on page 15), with the exception that all columns of the ref_model_run table are represented. Arrow keys, “Return”, and the mouse can all be used to activate cells of the input table.

At the bottom of the window are three buttons: “Search” will initiate retrieval of model_run_ids which match the input specifications; “Create” will create a new model_run_id based on the input specifications; and “EXIT DMI” will exit the DMI and return control to RiverWare.

When creating a new model_run_id:

- Fill in the Model Run Name, Model Id and Run Date columns; they cannot be NULL.
- Do not use “ ‘ ” in character strings.
- Use date formats which are accepted by the database; for instance, “19-DEC-1996”.

When searching for an existing model_run_id:

- Specify at least one column as a minimum search condition.
- Use date formats which are accepted by the database; for instance, “19-DEC-1996”.

- Never change the contents of the Model Source Code column.
- Specify the Model Id in order to immediately narrow the search.
- Do not use the Time Step field, as it is a long character string
- Provide defining parameters as precise as possible to narrow the search.
- If the Model Run Id is known, specify it and nothing else.

Press “Search” or “Create” when all searching/defining characteristics have been input. If creation is successful, the new model_run_id will be inserted into the ref_model_run table at all HDB installations in the region, and the new model_run_id will be displayed back to the DMI user. If the creation fails (as it will, for instance, if the Run Date is invalid), the user will be notified, and a window similar to the main window will be displayed. The user must then redefine the new model_run_id.

Next Page

Previous Page

When a model_run_id is searched for, zero, one, or more than one matches may be found. See “Interacting with the Auxiliary Interfaces” on page 18 for a description of how these cases are handled.

The “Exit DMI” button is available from every interface.

Synchronization of Model_Ids and Model_Run_Ids

Introduction

Related HDB installations must be able to exchange data; this requirement has influenced numerous database schema and system architecture decisions during the life of HDB.

The ability to share modeled data is part of this requirement; as a result, model_ids and model_run_ids must be kept synchronized at all HDB installations wishing to share modeled data. The synchronization of model_ids has a fairly simple solution; it is maintained through the Meta Data Application. All changes and additions to the hdb_model table must be made through this application.

The synchronization of model_run_ids is not so simple, namely because users must be able to create them on-the-fly, i.e., from within a RiverWare DMI, and this is not compatible with the architecture of the Metadata Application. This section discusses the method by which model_run_ids are kept synchronized

Overview of Model_Run_Id Synchronization

The goal of model_run_id synchronization is to have the exact same set of model_run_ids at all HDB installations which recognize each other. There can be no duplicate IDs, and all IDs must represent the exact same model run. The following bullet items describe, at a high level, how model_run_id synchronization is achieved.

- A one-time synchronization of all model_run_ids is done manually.
- Whenever a model_run_id is created through the Output DMI, it is inserted at all recognized HDB installations. Synchronization is maintained.
- When an HDB user creates a model_run_id outside of the DMI, it is inserted only on the local database. Synchronization is lost.
- Whenever the Input or Output DMI is invoked, the model_run_ids of all recognized installations are synchronized with those of all others. The result is that when the DMI runs, all possible model_run_ids, including those created remotely, are available for use. This is satisfactory, as currently only the DMIs need access to model_run_ids not created locally. If, in the future, other applications require synchronized IDs, they must first call the synchronization function.

Next Page

Previous Page

Note: The synchronization is quite quick, and does not noticeably affect DMI performance.

Reserved Model Run ID Ranges

The easiest way to keep model_run_ids synchronized is to ensure that no one ID will be used at more than one HDB installation. That is, if model_run_id 5 is created by an Upper Colorado, ID 5 should not be able to be used for a model run created at Lower Colorado.

To prevent this duplication of model_run_ids, a range of valid IDs is reserved for each HDB installation. A constant, ID_RANGE, has been defined in both the DMI source code (`dmi_utils.h`) and the database procedures `update_new_modelrun_id` and `gen_new_site_record`. It is currently set to 100.

For instance, Upper Colorado was initially assigned IDs 1 - 100, and Lower Colorado IDs 101 - 200. When Upper Colorado uses ID 100, the next model_run_id it will be assigned is 201; likewise, when Lower Colorado uses ID 200, it will be assigned ID 301. All model_run_id management is done within the DMI code and database procedures, and is invisible to the user. Any number of HDB installations and any number of model_run_ids per installation can be handled.

Keeping Records on Model_Run_Ids

The function which actually synchronizes model_run_ids needs to know:

- for each installation, the maximum model_run_id being used;
- for each installation, the maximum model_run from every remote installation which has been ported to the local installation; that is; up to what point are the model_run_ids synchronized?

These records are maintained in the ref_db_list table. The two columns which hold this information are:

- **maxid:** Integer indicator of model_run_id which will be assigned when next model run is created at this HDB site. Used by procedures and library functions; updated only when a new model_run_id is created at the

installation. Filled in only for `session_no` = 1 (local site).

- **max_sync_id:** Integer indicator of maximum `model_run_id` for this `db_site_db_name` which has been synchronized with (ported to) the local DB site (`session_no` = 1). In other words, this column is filled in only for `session_no` > 1, and indicates the maximum remote `model_run_id` which has been ported to the local site. Used and updated by synchronization function which runs at DMI startup.

Synchronization Example

Next Page

Assume that the following version of the ref_dmi_db_site_list table sits at Upper Colorado. Lower Colorado is the only other recognized HDB installation.

Previous Page

session_no	db_site_db_name	db_site_coee	maxid	max_sync_id
1	slcdg1::uchdb	UC	47	
2	hoover::lchdb	LC		101

The `maxid` column only has a value for the local site, Upper Colorado. This value 47 means that the *next* available **`model_run_id`** at UC is 47.

The `max_sync_id` column only has a value for the remote HDB site(s). For instance, the value of 101 here means that all LC `model_run_ids` up to 101 have also been ported to Upper Colorado, and that, the next time the synchronization function runs, it will start with Lower Colorado **`model_run_id`** 102.

Assume that this next table sits at Lower Colorado. Upper Colorado is the only other recognized installation.

session_no	db_site_db_name	db_site_coee	maxid	max_sync_id
1	hoover::lchdb	LC	1087	
2	slcdg1::uchdb	UC		43

The `maxid` column indicates that the *next* available **`model_run_id`** at LC is 108. (Hence model_run_ids 102 through 107 will be added at Upper Colorado.)

The `max_sync_id` column indicates that all UC `model_run_ids` up to 43 have also been ported to Lower Colorado, and that, the next time the synchronization function runs, it will start with Upper Colorado **`model_run_id`** 44, adding IDs 44 through 46 to Lower Colorado.

TCL

TCL is free software which must be installed on (or available to) every system which will run the RiverWare DMIs. The executables tclsh7.4 (or higher) and wish4.0 (or higher) are required; they can be installed anywhere, but **`/usr/`**

`local/bin` is the recommended location. These scripts are available from Sun, either via FTP or WWW.

If TCL is not installed on the machine running the DMI, but rather is being pointed to via a crossmount, then two environment variables must be defined in the user's .cshrc file. First, find the /lib directory in which the two TCL libraries, tcl7.4 and tk4.0, reside. Add these two lines to the user's .cshrc file:

```
setenv TCL_LIBRARY /<libraryPath>/tcl7.4
setenv TK_LIBRARY /<libraryPath>/tk4.0
```

The path to the compiled TCL executables (whether on the local or remote machine) must be in every DMI user's PATH environment variable (for instance, /usr/local/bin).

Next Page

Previous Page