Technical Documentation Version 7.4

Batch Mode

Center for Advanced Decision Support for
Water and Environmental Systems (CADSWES)
UNIVERSITY OF COLORADO BOULDER

Contents

Contents

Chapter 1
Batch Mode and RCL

Introduction

This section describes methodology to run RiverWare from the command line and optionally execute a RiverWare Command Language (RCL) script. This Batch Mode allows you to run RiverWare without seeing the user interface and allows for automation of many processes including advancing the run dates, running DMIs to bring in new data, running the model, generating output, and saving the model.

Running RiverWare from the Command Line

Running RiverWare from the command line provides an alternative to clicking on the RiverWare icon from the Windows system. In addition, RiverWare supports certain commands directly from the command line like loading models and rulesets and certain other operations. To access the command line, in the Windows Start menu Search field, type cmd.
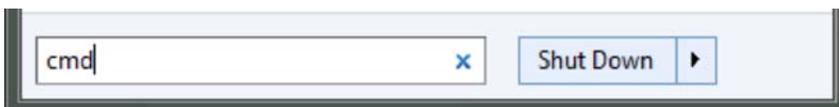
Then select the option for **cmd.exe**.
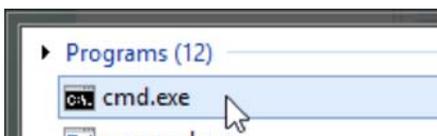
A window similar to that shown in Figure 1.1 will open. To start RiverWare from this window, enter the full RiverWare path (depending on your PATH configuration).

Figure 1.1 Starting RiverWare from the Windows Command Line

A complete list of command line options can be viewed by invoking RiverWare with the `--help` command line argument. On the Windows command line, type:

```
riverware.exe --help
```

When RiverWare is executed from a command line window, it is run as an independent process from the command shell, so exiting the command window will not automatically terminate RiverWare. The Windows task manager can be used to terminate batch runs of RiverWare.
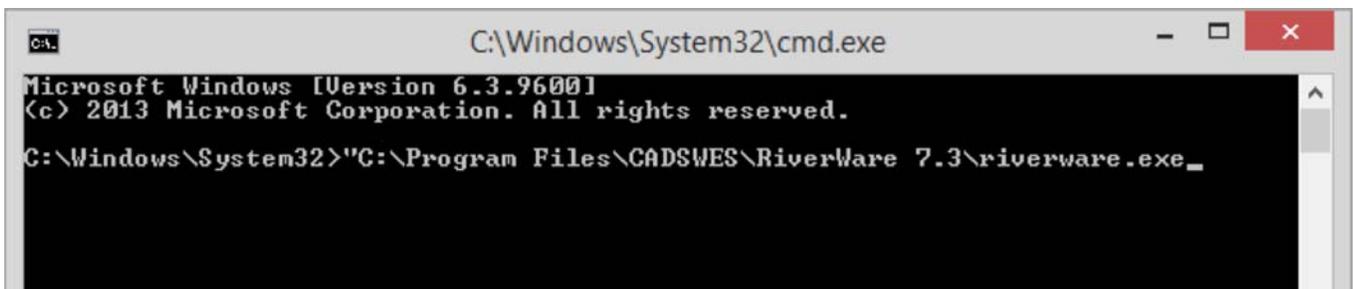
Sometimes, the user wishes to run one or more models as automatic tasks (as Windows "Scheduled Tasks"). Setting up the runs using batch mode and the command line options can be useful in this approach. For example, a user may wish to run a series of models overnight when the machine is more available. One command line option that is useful is the `--log <file>` switch. This option sends any diagnostic output to the specified file.

Running RiverWare in Batch Mode

The RiverWare Command Language (RCL) facility allows models to be run in batch mode. While in this mode, RiverWare does not open a user interface, but instead runs in background mode.

In batch mode, the user provides RiverWare with a script file, which contains the commands RiverWare executes. Commands are entered through the RCL file, rather than through the user interface. The commands are described in detail in the following document.

The syntax to invoke RiverWare in batch mode from a command prompt is:

```
riverware --batch <script file>
```

The following example will invoke the batch script file called `BasinOperationsScript.rcl` and will write the diagnostics output to file called `BasinOperationsDiagnostics.txt`.

```
riverware --batch "C:\MyRiverWareFiles\BasinOperationsScript.rcl" --log "C:\MyOut-
puts\BasinOperationsDiagnostics.txt"
```

The `<script file>` defines how the batch run should proceed. In the file, you specify commands to load a model, load a ruleset if necessary, start a run, and save the results. In the simplest case, you will load a model, run a simulation, save the model, and exit batch mode. The following RiverWare Command Language commands support these actions:

```
# Load the model
OpenWorkspace <Model Name>
# run the simulation
StartController
# save the model
SaveWorkspace <Model Name>.saved.mdl
# Close the opened model and exit RiverWare
CloseWorkspace
```

This is a very simple example. The full complement of key words and commands is presented in the following section.

RiverWare Command Language (RCL) Overview

Key

- Angled brackets (<>) indicate a required parameter.

- Brackets ([]) indicate an optional parameter.

- A vertical bar "|" within angled brackets or brackets indicates a choice of parameters. For example, "<#RunInfo|#MRM>" indicates that either "#RunInfo" or "#MRM" must be provided as a required parameter.

- Parameters starting with "!" are tokens which should appear exactly as is.

- Parameters starting with a "#" are RiverWare names which should appear exactly as is.

- A "#" sign at the beginning of the line represents a comment. Anything occurring after the "#" will be ignored.

- Parameters which contain embedded white space characters must be enclosed in curly braces {}, or quotes " ".

- Anywhere there is a file referenced below, the user can specify the path using either a full reference or using environment variables (see next section). On Windows, path names should use "/" or '\\'. Also, the file may reference a shared network: RCL accepts '\' but it has to be escaped with '\', so the RCL syntax is as follows: \\\\ComputerName\\ShareName\\...

Environment Variables

Environment variable are "system variables" that allow portability of models across machines and platforms.

Setting Environment Variables

If you wish to set environment variables and then reference them in the RCL script, they can be set using the TCL command "set env" as follows:

```
set env(variable) value
```

Then the variable can be accessed within the RCL script using the $env(variable) syntax. These variables are only available within the script, RiverWare has already inherited the environment before this is set and are then not available. To set variables to be available to RiverWare, use the SetEnv command; see "SetEnv" on Page 11. This command has the following syntax:

```
SetEnv <variable> <value>
```

These variables are then available in RiverWare but not available within the script. Most likely, you will want to set both variables within the script. Consider the following example:

```
# for use in the script:
set env(DMI_DIR) C:/DMI/data
# for use by RiverWare:
SetEnv DMI_DIR $env(DMI_DIR)
```

```
# open the model using the TCL environment variable syntax:
OpenWorkspace $env(DMI_DIR)/model.mdl.gz
```

Then within RiverWare, you can reference DMI_DIR using the $DMI_DIR syntax in DMI's, MRM or other places.

Using Environment Variables

Environment variables set outside of the RCL script can be referenced using the form:

* $VARIABLE

* $(VARIABLE)

* ${VARIABLE}

VARIABLE is a letter followed by zero or more letters, digits or underscores. If the variable is followed by a letter, digit or underscore it must be "quoted" with () or {}.

Special Parameters

*   {date time}

    A date and time of the format {MM/DD/YYYY HH:MM} or {MM-DD-YYYY HH:MM}, where the year includes the century and the hour is 24-hour based.

    Note: A date and time, which includes embedded white space, must be enclosed in braces {}.

*   {delta time}

    A timestep or time duration of the format {count units}, where units are "HOURS," "DAYS," "WEEKS," or "MONTHS."

    Note: A delta time, which includes embedded white space, must be enclosed in braces {}.

RCL Commands Reference

Following are the RCL commands in alphabetical order.

*   **CloseWorkspace**

    CloseWorkspace

Clears the workspace.

>   **Note:** The CloseWorkspace command does not close RiverWare. RiverWare closes once the execution of the complete batch script is complete. Multiple OpenWorkspace and CloseWorkspace commands within the same script will be executed within a single RiverWare session.

- ## CreateSlotCache

  `CreateSlotCache`

Creates a slot cache that contains the current values of all series slots on the workspace, in the range of the current controller. See "Slot Cache" in *Workspace* for a description of the cache.

> **Note:** The slot cache is under development. Please contact riverware-support@colorado.edu for more information and the current status of this feature.

- ## EnableInfoDiag

  `EnableInfoDiag <1|0>`

Enable Informational Diagnostics in the diagnostics manager dialog. When the argument is 1, diagnostics will be enabled. When the argument is zero, then the diagnostics will be disabled.

- ## ExecuteScript

  `ExecuteScript <Script Name>`

Executes the specified script as defined in the Script Manager. See "Script Management" in *Script Management* for details on RiverWare Script Management. The script name should be enclosed in curly braces {} if it contains embedded white space.

Example:

`ExecuteScript {Run Operations Model}`

- ## ExportOptAnalysisAsCSV

  `ExportOptAnalysisAsCSV <file name> [<section name> true|false true|false]`

Exports the Optimization Analysis information as a comma separated values file. The arguments are as follows:

- `<file name>` is the name of the file to write. If the file path contains embedded white space, it should be enclosed in curly braces {}.

- `<section name>` are the (optional) sections to export. Defaults to all sections. The section names are as follows and correspond to combinations of the Optimization sections; see "Export as CSV" in *Optimization* for details.
  ConstraintAll
  ConstraintNew
  ConstraintPhysical
  ConstraintPolicy
  ConstraintPrior
  None
  SolutionAndPolicy
  SolutionGoalSummary
  SolutionInfo
  SolutionSingle
  VariableAll

`VariableOther`

`VariableSlot`

- The first optional `true|false` specifies whether to skip export of constraint and variable rows associated with fully satisfied constraints or solutions. Defaults to true, i.e., do not export these rows for fully satisfied solutions.

  Note: If this argument is present, all previous arguments must also be present.

- The second optional `true|false` specifies whether to skip export of constraint and variable rows associated with the solution to the postponed problem, which is relevant only for runs in which a seed value was used to postpone one or more solutions. Defaults to true, i.e., do not export these rows for postponed problem solutions.

  Note: If this argument is present, all previous arguments must also be present.

Example:

```
ExportOptAnalysisAsCSV {C:\MyOutputs\MyOptAnalysis.csv} {PolicyConstraints} {true}
```

- ### GetRunInfo

  ```
  GetRunInfo <#RunInfo|#MRM <MRM configuration>>
  [!InitDate] [!EndDate]
  [!Duration] [!Step] [!Controller]
  ```

Prints single run information (#RunInfo) or multiple run information (#MRM) to standard output. If #MRM is specified, you also need to specify an MRM configuration <MRM configuration>.

- `!InitDate` prints the run's initial date.

- `!EndDate` prints the run's end date.

- `!Duration` prints the run's duration.

- `!Step` prints the run's timestep.

- `!Controller` prints the current controller.

If no parameters are specified, all information is printed.

Following are examples of the syntax:

```
GetRunInfo #RunInfo !InitDate
GetRunInfo #RunInfo !EndDate
```

- ### GetSlot

  ```
  GetSlot <Object.Slot> [<date time>]
  ```

Returns the series or scalar slot's value, in user units, as a Tcl variable. `Object.Slot` should be enclosed in curly braces `{}` if it contains embedded white space.

- `<date time>` is only for series slots and must be enclosed in curly braces `{}`, as it contains embedded white space - for example, `{05-15-2010 12:00}`.

The command is used when writing batch mode scripts which make use of the Tcl scripting language. GetSlot returns the value in a Tcl variable, so sample usage includes:

```
set v [GetSlot {Object.Slot} {05-05-2010 12:00}]
SetSlot {Object.Slot} {05-05-2010 12:00} $v
```

The Tcl variable can be used in arithmetic expressions, conditionals, etc:

```
set v [expr 0.5 * $v]
if {$v > 1.0} {
...
} else {
  ...
}
```

Further scripting help with the TCL language is beyond the scope of this reference but can be found in online searches.

- ### HDB_EnsembleID

  ```
  HDB_EnsembleID <HDB dataset name> <Ensemble ID> ...
  ```

Sets the Ensemble ID for an HDB dataset that is configured to have its ID selected when MRM is started.

- `<HDB dataset name>` is the name of the HDB dataset that is configured to have its ID selected when MRM is started. If the name contains embedded white space, it must be enclosed in braces `{}`.

- `<Ensemble ID>` is the Ensemble ID to set on the dataset.

IDs can be set for more than one dataset by adding addition pairs of names and IDs.

- ### HDB_ModelRunID

  ```
  HDB_ModelRunID <HDB dataset name> <Model Run ID> ...
  ```

Sets the model run ID for an HDB dataset that is configured to have its ID selected when its DMI is invoked.

- `<HDB dataset name>` is the name of the HDB dataset that is configured to have its model run ID selected when its DMI is invoked. If the name contains embedded white space, it must be enclosed in braces `{}`.

- `<Model Run ID>` is the Model Run ID to set on the dataset.

IDs can be set for more than one dataset by adding addition pairs of names and IDs.

- ### InvokeDMI

  ```
  InvokeDMI <DMI name> [-UuserParam=value]
  ```

Invokes a DMI.

- `<DMI name>` is the name of the DMI to invoke. If the name contains embedded white space, it must be enclosed in braces `{}`.

- `[-UuserParam=value]` sets the user parameter "userParam" to "value".

Example:

```
InvokeDMI {Operations Data Input DMI}
```

- **InvokeDssDMI**

  `InvokeDssDMI <DMI name> !DssFile <DSS file> !DssFPart <DSS F part>`

Invokes a DSS DMI.

- `<DMI name>` is the name of the DSS DMI to invoke. If the name contains embedded white space, it must be enclosed in braces {}.

- `<DSS file>` is the name of the DSS file. If the file name contains embedded white space, it must be enclosed in braces {}.

- `<DSS F part>` is the DSS F part. If the F part contains embedded white space, it must be enclosed in braces {}.

- **ListDMI**

  `ListDMI [!Type <Input|Output>] [!Dataset <dataset type>]<output file>`

Writes the names of the DMIs which match the specified criteria to the output file.

- `[!Type <Input|Output>]` limits the output to either Input or Output DMIs.

- `[!Dataset <dataset type>]` limits the output to DMIs which use only the specified dataset type.

- `<output file>` is the file the DMI list is written to.

Examples:

`ListDMI C:/MyOutputs/DMIListFile.txt`

  Writes the names of all DMIs to the file DMIListFile.txt.

`ListDMI !Type Input C:/MyOutputs/DMIListFile.txt`

  Writes the names of all Input DMIs to the file DMIListFile.txt.

`ListDMI !Type Input !Dataset DSS C:/MyOutputs/DMIListFile.txt`

  Write the names of all Input DMIs which use only DSS datasets to the file `DMIListFile.txt`.

- **LoadConstraints**

  `LoadConstraints <file path> [!Append]`

Loads the constraint set into the workspace. By default, constraint sets which are already loaded into the workspace are cleared.

- `<file path>` is the path to where the constraint set resides on disk.

- `!Append` does not clear constraint sets which are already loaded into the workspace.

- **LoadOptSet**

  `LoadOptSet <file path>`

Loads the optimization goal set into the workspace.

- `<file path>` is the path to where the optimization goal set resides on disk.

Example:

```
LoadOptSet C:/MyRiverWareFiles/MyOptGoalSet.opt
```

- **LoadRules**

```
LoadRules <file path>
```

Loads a ruleset into the workspace.

- `<file path>` is the path to where the ruleset resides on disk.

Example:

```
LoadRules C:/MyRiverWareFiles/MyRuleset.rls
```

- **OpenGlobalSet**

```
OpenGlobalSet <file path>
```

Opens a global function set.

> **Note:** This must be called before LoadRules if the ruleset references functions in the global set.

- `<file path>` is the path to where the set resides on disk.

Example:

```
OpenGlobalSet C:/MyRiverWareFiles/MyGlobalFunctions.gfs
```

- **OpenWorkspace**

```
OpenWorkspace <file path>
```

Loads a model into the workspace.

- `<file path>` is the path to where the model resides on disk.

Example:

```
OpenWorkspace C:/MyRiverWareFiles/BasinOperationsModel.mdl
```

- **Output**

```
Output <outputDeviceName>
```

Generates an output file that is specified in the Output Manager of the model. See "Output Manager" in *Output* for details about the available output devices and how they can be configured. Plot pages cannot be generated from a RCL script.

- `<outputDeviceName>` is the name of the output device saved in the Output Manager of the model.

Examples:

```
Output OperationsForecastReport
Output FloodStorageAllocationChart
```

- **RequireVersion**

```
RequireVersion <major version>.<minor version>[.<patch level>]
```

Specifies the RiverWare version required to execute the batch script. If the RiverWare version is earlier than the required version, the command issues an error message and stops the script file's execution.

- `<major version>.<minor version>[.<patch level>]` are the major version, minor version and optional patch level of RiverWare necessary to execute this batch script.

Example

```
RequireVersion 7.3.1
```

If the version of RiverWare is earlier than 7.3.1, the batch script will abort with an error message.

- **ReorderGoalSet**

  ```
  ReorderGoalSet <order slot>
  ```

This command allows you to reorder items (goals, groups, or certain statements) within the currently loaded optimization goal specified set. The desired order is specified by the single column order slot, a table slot, each of whose rows corresponds to the name of an item in the goal set. The values in the single column of the order slot are interpreted as integers giving the desired order of that object with respect to other set items referred to in the table. This command is similar to the Reorder RPL Set Script action; see "Reorder RPL Set" in *Script Management*.

Example:

```
ReorderGoalSet BasinDataObj.GoalPriorities
```

The command refers to the table slot shown in Figure 1.2.

Figure 1.2 Table slot with priorities to reorder the goal set
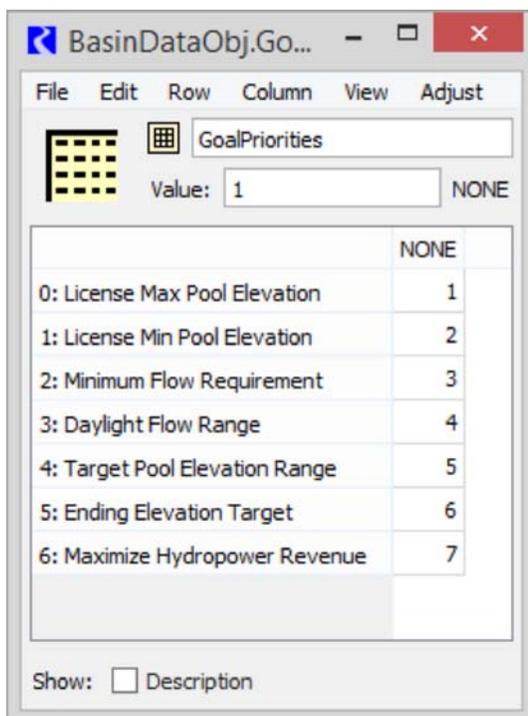
- **SaveWorkspace**

  `SaveWorkspace [file path]`

Saves the model which is currently loaded into the workspace.

- `[file path]` is the path to where the model is to be saved on disk. If the file path is not specified, the `OpenWorkspace` file path is used.

Examples:

`SaveWorkspace`

Saves the model to the same file name from which it was opened

`SaveWorkspace C:/MyOutputs/SavedModelFile.mdl`

Saves the model to the file specified

- **SetDiagFile**

  `SetDiagFile <diagnostic file>`

Sets the location of the diagnostic output file.

- `<diagnostic file>` is the path to where the diagnostic file will be written.

- **SetEnv**

  `SetEnv <variable> <value>`

Sets an environment variable for use within the RiverWare run.

See "Environment Variables" on Page 3 for details on setting and using environment variables.

- **SetRunInfo**

  `SetRunInfo <#RunInfo|#MRM <MRM configuration>>`
  `[!InitDate <date time>]`
  `[!StartDate <date time>]`
  `[!EndDate <date time>]`
  `[!Duration <delta time>]`
  `[!Controller <controller type>]`

Sets single run information (#RunInfo) or multiple run information (#MRM). If #MRM is specified, you also need to specify an MRM configuration `<MRM configuration>`.

- `!InitDate` sets the run's initial date to `<date time>` of format {MM-DD-YYYY HH:MM}.

- `!StartDate` sets the run's start date to `<date time>` of format {MM-DD-YYYY HH:MM}.

  Note: If you specify both `!InitDate` and `!StartDate` with the same `SetRunInfo` command, the argument that appears second will take precedence.

- `!EndDate` sets the run's end date to `<date time>` of format {MM-DD-YYYY HH:MM}.

- `!Duration` sets the run's duration to `<delta time>` of format {Number TimeUnits}.

- `!Controller` sets the run's controller to `<controller type>`. The controller type must be one of the controllers available in the Run Control dialog:

    – `Simulation`

    – `Rulebased Simulation`

    – `Optimization`

      Note: Controller types that contain embedded white space must be enclosed in braces `{}`.

Following are samples of the syntax:

    `SetRunInfo #RunInfo !InitDate {04-21-2011 24:00} !EndDate {04-26-2011 24:00}`

    `SetRunInfo #RunInfo !StartDate {04-22-2011 24:00}`

    `SetRunInfo #RunInfo !EndDate {04-26-2011 24:00}`

    `SetRunInfo #RunInfo !Duration {200 Days}`

    `SetRunInfo #RunInfo !Controller {Rulebased Simulation}`

    `SetRunInfo #MRM "Paleo Runs"`

- ### SetSlot

    `SetSlot <Object.Slot> [<date time>] <value>`

Sets the series or scalar slot's value to the value, assumed to be in user scale and units. `Object.Slot` should be enclosed in curly braces `{}` if it contains embedded white space.

- `<date time>` is only for SeriesSlots and must be enclosed in curly braces `{}` as it contains embedded white space - for example, `{05-15-2010 12:00}`.

See , for details on possible usage.

- ### SetTrace

    `SetTrace <1|0>`

Enable or disable trace messages from the RCL interpreter. The trace messages are prefaced with "TRACE:" and written to standard output; they can be captured in a file with the `--log <log file>` command line option.

- ### SlotList

    `SlotList <output file>`

Writes information about all slots in the model to the output file.

- `<output file>` is the file the slot list is written to.

The information is written as comma-separated values to the output file; the `--help` command line option lists the output fields.

- ### SlotListDMI

    `SlotListDMI <DMI name> <output file>`

Writes information about the slots to/from which the DMI imports/exports data; the information is written as comma-separated values to the output file. The information includes:

- The slot's name.

- The slot's priority (determined by its dataset association).

- The slot's begin date "mm-dd-yyyy".

- The slot's begin time "hh:mm:ss".

- The slot's end date "mm-dd-yyyy".

- The slot's end time "hh:mm:ss".

- The dataset associated with the slot .

- The dataset's type, currently DSS or HDB. If the Dataset is DSS, then the slot's DSS path "/A/B/C/D/E/F/" is also provided as another comma separated value.

- **StartController**

  ```
  StartController [!MRM <MRM configuration>]
  [firstTrace=N]
  [numTrace=M] ctlFile=file]
  ```

Starts the current RiverWare controller.

- `!MRM` will start the Multiple Run Management controller instead of the single run controller, using the specified MRM configuration. RCL currently does not support many commands for defining multiple runs, so models should already contain the multiple run definitions.

The following commands override the values in the MRM configuration in the model file (`[!MRM <MRM configuration>]` must be used if you wish to specify these parameters):

- `[firstTrace =N]` sets the index sequential initial offset

- `[numTrace=M]` sets the index sequential number of runs

- `[ctlFile=file]` sets the output control file.

- **SyncObj**

  ```
  SyncObj [!Acct] [!ExDiffTS]
  [!StartDate <start date>]
  [!EndDate <end date>]
  ```

Synchronize slots to either the Run Control or the specified dates.

- `[!Acct]` includes accounting slots.

- `[!ExDiffTS]` excludes slots whose timestep size differs from the Run Control timestep size.

- `[!StartDate <start date>]` synchronizes slots to the specified start timestep, rather than the Run Control initial timestep.

- [!EndDate <end date>] synchronizes slots to the specified end timestep, rather than the Run Control finish timestep.

Examples:

SyncObj

Synchronizes all series slots to the Run Control initial timestep and finish timestep.

SyncObj !ExDiffTS !StartDate {04-23-2013 12:00} !EndDate {04-26-2013 12:00}

Synchronizes series slots to the specified start and end timesteps and excludes slots that have a timestep size that differs from the Run Control timestep size.