Contents

[1 Preface 2](#_Toc32930270)

[2 Installation of RTC-Tools 3](#_Toc32930271)

[3 Developing an RTC-Tools model for optimization 4](#_Toc32930272)

[3.1 The folder structure of an RTC-Tools model 4](#_Toc32930273)

[3.2 The hydraulic layer 4](#_Toc32930274)

[3.3 The data/IO layer 10](#_Toc32930275)

[3.4 Control layer 11](#_Toc32930276)

[4 Results of the optimization model for Hume Dam 14](#_Toc32930277)

[5 The RTC-Tools simulation model 19](#_Toc32930278)

[5.1 Preface 19](#_Toc32930279)

[5.2 The hydraulic layer 19](#_Toc32930280)

[5.3 The control layer 19](#_Toc32930281)

[5.4 Results of the simulation model for Hume Dam 20](#_Toc32930282)

[6 Use of PyCharm to modify the python code 21](#_Toc32930283)

[7 Delft-FEWS 23](#_Toc32930284)

[7.1 Analyze the dataflow into and out of the model 23](#_Toc32930285)

[7.2 Implement a Delft-FEWS preprocessing module for the optimization model 23](#_Toc32930286)

[7.3 Implement a Delft-FEWS preprocessing module for the simulation model 24](#_Toc32930287)

[7.4 Implement the RTC-Tools 2 Delft-FEWS time series mapping 24](#_Toc32930288)

[7.5 Compose the ModuleDataSet File 25](#_Toc32930289)

[7.6 Compose the ModuleParameter file 26](#_Toc32930290)

[7.7 Implement a general adapter module to run the RTC-Tools optimization model 27](#_Toc32930291)

[7.8 Implement a general adapter module to run the RTC-Tools simulation model 27](#_Toc32930292)

[7.9 Compose workflows for simulation and optimization model 28](#_Toc32930293)

[7.10 Visualizing the results 28](#_Toc32930294)

[7.11 Extending the ModifierTypes 29](#_Toc32930295)

[7.12 Add RTC-Tools section to topology 30](#_Toc32930296)

[8 Testing RTC-Tools in Delft-FEWS 31](#_Toc32930297)

[8.1 Testing - URBS Forecast 31](#_Toc32930298)

[8.2 Test – Run RTC-Tools in simulation mode 32](#_Toc32930299)

[8.3 Test – Run RTC-Tools in optimization mode 34](#_Toc32930300)

[8.4 Ensemble mode 37](#_Toc32930301)

RTC-Tools model Hume dam

Notes of the model development

# Preface

This document is a step-by-step guideline to build a simple reservoir model with RTC-Tools, shown with the Hume dam model as example.

Chapter 2 of this guideline document starts with installation instructions for RTC-Tools on a desktop. This is required for both the development of RTC-Tools models, as for running RTC-Tools from Delft-FEWS.

The next part of this guideline documents discusses how to develop RTC-Tools models.

Chapter 3 explains how to build an RTC-Tools model for optimization. Here the structure of an RTC-Tools model is introduced. Optimization results are shown for two different settings of operational goals.

Chapter 5 explains how to build an RTC-Tools model for simulation.

Chapter 6 describes the use of PyCharm to modify of develop a model.

The next part of this documents discusses how to link an existing RTC-Tools model to Delft-FEWS, run it and test it.

Chapter 0 explains how to connect the RTC-Tools model to Delft-FEWS.

Chapter 8 includes test scripts for running RTC-Tools within Delft-FEWS

We end this guideline with known issues in the development of the Hume dam model.

Chapter **Error! Reference source not found.** summarizes known issues that have come up during the development of the Hume dam model.

# Installation of RTC-Tools

The Delft-FEWS configuration delivered contains a module data set with Python and RTC-Tools binaries included.

For development of new RTC-Tools models or the extension of existing RTC-Tools models, the following installation procedure is recommended.

Before installing RTC-Tools, Python 3.7 must be installed on the computer. In our example, we assume that the installation folder is

c:\Python37\

To install RTC-Tools, carry out the following steps:

1. Navigate with a tool such as ‘Windows PowerShell’ to the folder of your project, say <Project\_Folder>.
2. Create a Python virtual environment by executing the following command in the installation directory.

c:\Python37\python –m venv venv

1. Activate the virtual environment that has been created in the folder ‘*venv*’ with the following command:

‘.\venv\Scripts\activate’

1. Install RTC-Tools with the PIP method:

‘pip install rtc-tools rtc-tools-channel-flow’

If not present, the „Visual C++ Redistributable for Visual Studio 2015“ must be installed on the computer. It can be downloaded here:

https://www.microsoft.com/en-us/download/details.aspx?id=48145

We recommend installing pandas for data processing as well:

‘pip install pandas’

A Modelica editor can be useful for development purposes. We recommend the Open Modelica Connection Editor and this editor will be used in the explanations. It can be downloaded here:

<https://openmodelica.org/>

A Python development environment for the Python is advised when developing or modifying a model. We recommend use of PyCharm and more details about its use are given in  [section](#_Use_of_pycharm) 6.

# Developing an RTC-Tools model for optimization

## The folder structure of an RTC-Tools model

An RTC-Tools model contains of three layers:

* The hydraulic layer. Here the schematization of the water system is defined in Modelica language.
* The data/IO layer with input and output data.
* The control layer. This layer consists mainly of Python code, and here the control of the water system is defined. Python code is also used to connect control, hydraulics and data.

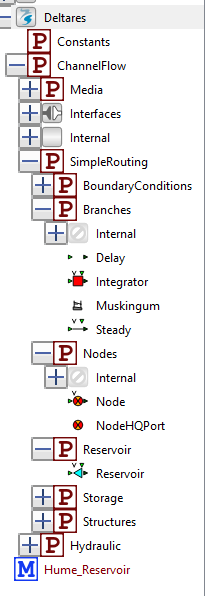
Typically, an RTC-Tools model is organized in the following folders:

* input. This folder contains input time series and parameters as managed by the Fews-application.
* model. This folder contains the the hydraulic layer, i.e. the schematization of the water system, captured in Modelica file(s). This folder also holds csv files with additional information on the reservoirs, such as volume-level-relations, min/max values and information on flow delays.
* output. RTC-Tools writes output to this folder.
* src. This folder contains all Python source code, which is mainly the control layer, but also additional tools can be stored here.

Additionally, this folder can contain your virtual environment folder when you made it here (as recommended)

## The hydraulic layer

### Preparations

* Open the Open Modelica Connection Editor
* Load the Deltares library “Open Channel Flow” from:

*‘*\venv \Lib\site-packages\rtctools\_channel\_flow\modelica\Deltares\package.mo*’.*

### Schematize the water system with Modelica objects from the Deltares library

The hydraulic layer is the schematization of the water system. RTC-Tools uses the Modelica language for this. RTC-Tools comes with a library for open channel flow models. Start with ‘File🡪New Modelica Class’and give the new class the name ReservoirModel. Use the building blocks from this Deltares Modelica library and drag generic model objects to the canvas.

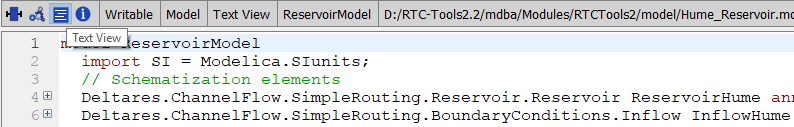
When drawing the network, take the following principles into account:

* Objects of node type alternate with Branch objects.
* Start with simple objects first and change to advanced objects later.
* Start with a simple model and make sure that it runs, and extend it later.

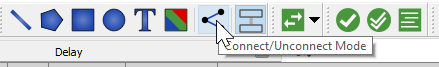
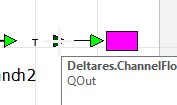
Save the Modelica file with a meaningful name under the folder

model

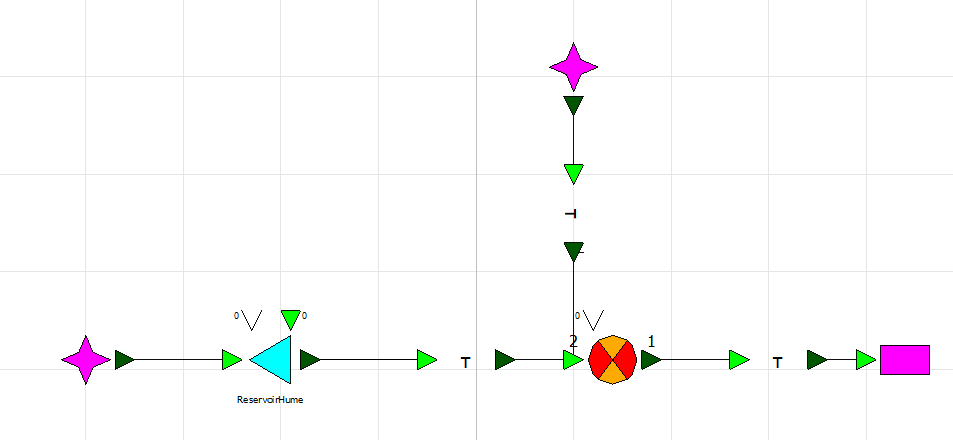
Switch to Text view to make editions in the Modelica file.



To connect model objects, choose the Connect mode and connect the building blocks:

For the Hume dam model (i.e. a single reservoir with one downstream tributary and a downstream Terminal node, with fixed delays inbetween) the network should look like this:

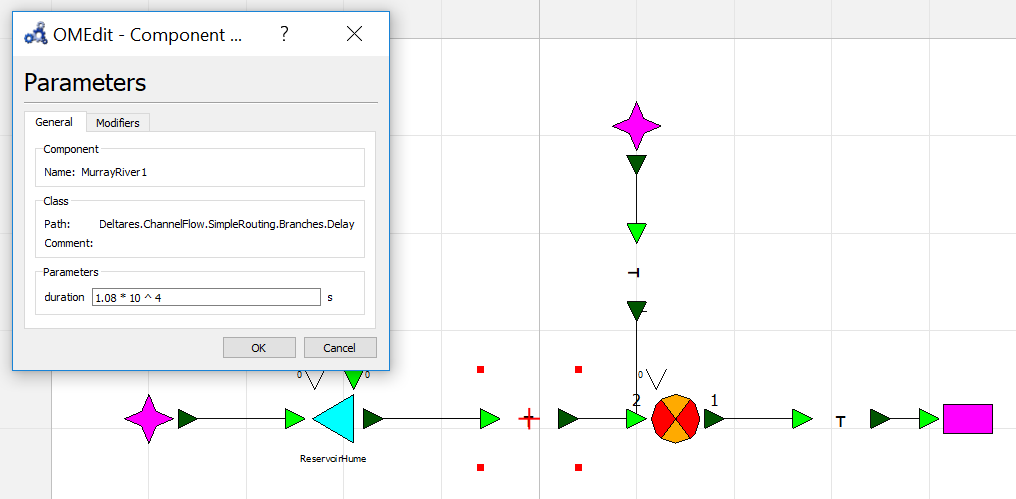


The Hume Dam model consists of the following elements, from left to right:

* An inflow node “InflowHume”. This node represents a flow boundary condition.
* A reservoir node “ReservoirHume”. This model represents the reservoir. Behind this node is the reservoir equation.
* A delay branch “MurrayRiver1”. This branch represents section of the Murray River downstream of the reservoir. A delay equation is behind this branch type, which delays a time series by full hours.
* A node “KiewaInflow”. This node represents the location where the Kiewa river flows into the Murray river. The Kiewa river is modelled with an inflow node and a delay branch.
* Another delay branch for the river section between Kiewa Inflow and Doctor’s point.
* A terminal “TeminalDoctorsPoint”. This node represents the downstream end of the model.

Between the elements listed above connections have been established. These are visible on the canvas as lines from dark green to light green triangles.

Fill in the time lag information for the delay branch. Select a model object, right-click and choose “Parameters”. The time lag must be a multiple of the time step (one hour in this case) but specified in seconds.



### Add input and output variables to the Modelica model

Input and output variables tell Modelica what data it can expect as input and what it must provide as output. Modelica distinguishes the following variable types:

* Input, fixed = true: This type is for variables from external time series. The time series values are known beforehand.
* Input, fixed = false: This is (from a Modelica point of view) an input variable that changes during the computation. Variables of type fixed = false are computed within the control layer of an RTC-Tools model during the simulation or with optimization. If this time series should appear in the output file (timeseries\_export.csv), corresponding Python-Code must be programmed in the control layer.
* Output: Output time series are computed by Modelica equations. The control layer can use output time series. Output time series are written into the output file by RTC-Tools.

Switch to text mode and define input variables of type fixed=true for all boundary conditions. In our case, these are the inflow boundary conditions for the reservoir and the Kiewa river. RTC-Tools uses the variables as time series IDs. The timeseries IDs used for boundary conditions are free to choose, as these can be mapped to Fews-timeseries IDs using the rtcDataConfig.xml file. Choose meaningful names.

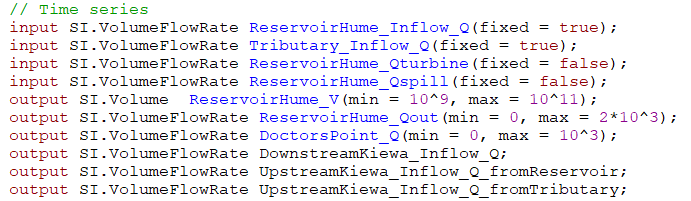
Define input variables of type fixed=false for all control parameters. These are the turbine flow and the spill flow of the reservoir.

Define output variables for parameters of interest, e. g. the volume of the reservoir.

The Python script of the control layer assumes that the following ID convention is applied for controlled/computed timeseries: <locationname>\_<parameter>, with at least the following timeseries ids.

* <reservoirname>\_V
* <reservoirname>\_Qspill
* <reservoirname>\_Qturbine
* <reservoirname>\_Qout
* <dowstreamflowtargetlocation>\_Q

The block with the variable definition should now look like this:



### Add equations for the variables to the Modelica file

The variables that have been defined in the previous time steps must now be assigned to the model objects where appropriate. This is done with the help of mathematical equations.

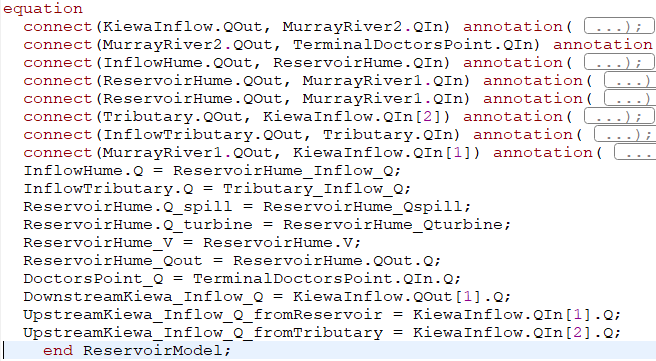
Look for the equation statement in the Modelica file (Text View). Below the equation statement some connect statements are already listed. These have been generated within step 3.2.2 during the connection of the model objects in the Diagram View. Technically, these connections are equations, too.

The principle of the assignment of a variable to a model object is the following:

* <Model object>.<Variable>.<Quantity> = <TimeSeries ID>

In the statement ReservoirHume.Qout.Q, ReservoirHume is the name of the reservoir node (see above), QOut the variable (reservoir outflow), and Q the quantity (flow).

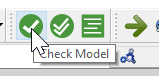
Here is an example with equation statements for the Hume dam model:



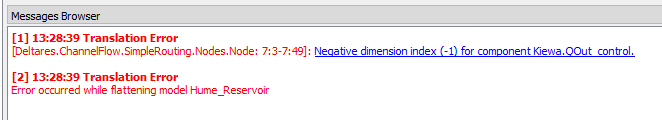
Some model objects can have more than one variable. In this case the index must be added in brackets. For the node KiewaInflow, KiewaInflow.Qout[1].Q means the first inflow (and not the second).

### Verify the Modelica model schematization and make manual editions

Verify the schematization with the “Check model” tool:



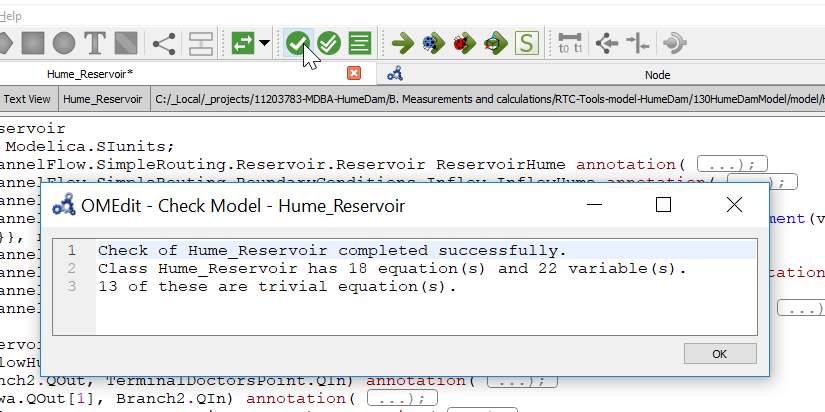
Probably the message window shows errors in red.



Walk through the errors and fix them. Things that usually need to be edited manually are:

* For nodes, the number of incoming and outgoing connections must be added. Example: Deltares.ChannelFlow.SimpleRouting.Nodes.Node Kiewa (nin = 2, nout = 1) has two ingoing connections and one outgoing connection.
* For connections with nodes that have multiple items, the index must be specified. Example: The connection connect(Tributary.QOut, Kiewa.QIn[2]) connects Tributary.QOut with inflow number two of node Kiewa (Kiewa.Qin[2]).

In the end, the check result should look as follows:



### Balance the equation system of the Modelica model

The model check provides the number of equations and the number of variables. Equations and variables must have the same number.

If the equation system is not balanced, the Modelica model must be modified accordingly. Check that there are input time series definitions for

* Boundary conditions for inflow (BoundaryConditions.Inflow, fixed = true).
* Control variables (Reservoir.Q\_turbine, Reservoir.Q\_spill, fixed = false).

### Adding Reservoir properties

The control layer (i.e. the model script) has a generic setup for reservoir models, assuming one downstream flow target location per reservoir.

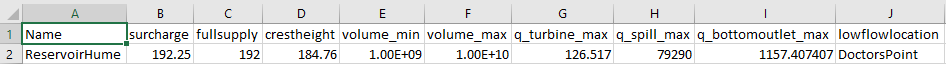
This generic setup is driven from two types of csv files containing reservoir information:

* Reservoir.csv containing specific reservoir properties (one file contains all reservoirs)
* Volumelevel.csv containing the volume-level relationship for one or more reservoirs

Reservoir.csv (semi-colon separated) holds the following properties:

* name (string, reservoir name is used in the Modelica file)
* surcharge (numeric, surcharge level)
* fullsupply (numeric, full supply level)
* crestheight (numeric, crest height) (relevant for simulation mode only)
* volume\_min (numeric, minimum reservoir volume)
* volume\_max (numeric, minimum reservoir volume)
* q\_turbine\_max (numeric. Maximum flow through turbines)
* q\_spill\_max (numeric, maximum flow over spillway)
* q\_bottomoutlet\_max (numeric, maximum flow through bottom gates)
* lowflowlocation (string, reference to downstream flow target location as used in Modelica file)

For Hume Reservoir, the contents looks like:



Volumelevel.csv (semi-colon separated) holds the volume-level relation for the reservoir. Multiple files are allowed. The following header should be used:

* ReservoirName
* Storage\_m3
* Elevation\_m

For Hume Reservoir, the content looks like:



## The data/IO layer

Input and output of RTC-Tools is basically time-series oriented but may also contain a parameter file. RTC-Tools supports two data formats:

* FEWS-PI (XML files)
* CSV

If RTC-Tools runs within a Delft-FEWS system, the FEWS-PI format should be used. For standalone applications it can make sense to choose the CSV format, because CSV files can be edited and visualized easily in Microsoft Excel or Veusz. There is also a conversion tool pandas4RTC.py that converts FEWS-PI files to CSV for quick model result checks.

The time series for import must contain all time series that are specified as input fixed = true in Modelica. The time series ID (in CSV file: the column name) must correspond to the name of the variables in Modelica.

When using the CSV format make sure that the first column contains the time steps and has the date format in yyyy-mm-dd hh:mm:ss.

For time series in XML format, the file rtcDataConfig.xml contains the mapping of ids between the RTC-Tools time series definition (time series ID) and the Fews ; locationID and parameterID as used in the file timeseries\_import.xml, i.e. the time series file which Fews exports. When adding output time series in the Modelica file, their time series ID should also be added to this rtcDataConfig.xml file e.g., when interested in the time series “*arbitrary\_inflow*” at location “*loc\_\_a*”, add “*input SI.VolumeFlowRate arbitrary\_inflow;*” to the Modelica file (above the equations section) and add

<timeSeries id="*arbitrary\_inflow* ">

<PITimeSeries>

<locationId>loc\_\_a</locationId>

<parameterId> *arbitrary\_inflow\_output\_name* </parameterId>

</PITimeSeries>

</timeSeries>   
to rtcDataConfig.xml. Now a time series which is known in the Modelica and python files as “*arbitrary\_inflow*” will show up in timeseries\_export.xml as “*arbitrary\_inflow\_output\_name*”.

Within the ReservoirModel setup as used for the Hume Reservoir, the rtcParameterConfig.xml file is used to transfer information about the goals to be switched on and off in the control layer.

The structure of the parameter identifiers typically is:

* <reservoirname>\_use\_<goal\_type> [Boolean]
* <reservoirname>\_priority\_<goal\_type> [intValue]

e.g.

<parameter id="ReservoirHume\_use\_surcharge\_max">

<boolValue>True</boolValue>

</parameter>

<parameter id="ReservoirHume\_priority\_surcharge\_max">

<intValue>4</intValue>

</parameter>

The different goal types supported in the control layer are explained in more detail in the next section.

## Control layer

### Generic setup for simple reservoir models with downstream flow target

The control layer, i. e. the Python code, delivered with the Hume reservoir model, is generally applicable for single and multi-reservoir systems with zero or one downstream target flow locations (like doctor’s Point) per reservoir. The code parses the CSV datafiles and the settings in the rtcParameterConfig.xml to loop over the reservoirs and add the associated goals. This same control layer can be applied for other models, as long as the naming conventions explained in the previous sections are adopted.

The optimization model uses sequential goal programming to achieve a prioritized sequence of goals, by preserving the previous goal while dealing with the next prioritized goal. Goals and constraints make use of the goal definitions. This model makes only use of path-goals, which applies the goal to each point in time instead of to specified points in time. We use this type of goal for operational goals that apply for each time step, e. g. maximum and minimum flows for a certain location. Goals can also be formulated generally, for example to minimize the cumulative discharge during the simulation period.

Goals can be exceeded, but constraints cannot. If we set a maximum discharge goal for a certain location (e. g. Doctor’s point), it can be met most times during a year, but it might be exceeded during a flood event. RTC-Tools tries to meet a goal as good as possible. Constraints should only be used for physical limits like a minimum water volume of 0 in a reservoir to ensure that the volume does not become negative. If a constraint cannot be met, the optimization terminates with an infeasible solution error.

The simple reservoir model used for Hume Reservoir can handle the following types of goals (per reservoir) at a specified priority:

* Maximum surcharge level
* Maximum Full supply level
* Discharge max and min at a downstream target location (Doctor’s point for the Hume reservoir model)
* A user defined target for total reservoir release Q\_out
* Minimize Q\_spill.
* Minimize Q\_out.
* Smoothen Q\_turbine

The data describing those goals is held in the rtcParameterConfig.xml file.

The simple reservoir model can handle the following types of constraints (per reservoir):

* Volume max, volume min
* Q\_turbine\_max (where P=eta\*rho\*g\*h\*Q)
* Q\_spill\_max

The data describing those constraints is specified in the reservoir-properties csv file.

Note that optimization basically does not support if-then-else logic. For these conditions more advanced methods like mixed integer optimization must be applied. Consequently, there is no condition in the simple ReservoirModel that checks if the water level has reached the crest level of the dam. Bottom outlet flow and spillway flow are combined in the Q\_spill variable in the optimization model.

### The control layer code explained in detail

The control layer of the simple ReservoirModel setup typically does not need any modification. However, in case one intends to build more complex control layers, insight is needed how this control layer can be defined. This section provides the necessary detail.

The RTC-Tools Application Programming Interface (API) offers various methods, which can be overridden by the control layer. This override mechanism is the standard mechanism used to add the control functionality. Note that the full API () of RTC-Tools is described at <https://rtc-tools.readthedocs.io/en/stable/>.

To build a new control layer we recommend not to start from scratch, but to take the control layer version of the simple Reservoir Model as example and modify it.

The control layer of an RTC-Tools model for optimization is structured as follows:

* Import statements. Here some RTC-Tools libraries are referred to, but also standard Python libraries. In case of the simple ReservoirModel we also import a dedicated class for reservoir properties and delays.
* Goal definitions. Goal definitions are classes that contain a generic description for goals and are used to define site-specific goals.
* The main class with the RTC-Tools optimization model.
* The run statement that executes the main class.

It is recommended to use the same name for the main class as the one used in the Modelica file as this is the default situation. In this case the main class is indicated with

**class** ReservoirModel( … ):

and the text within parentheses indicates the super-classes of this class.  
Any python class is initialized with the \_\_init\_\_ method. This method can remain empty, but in the simple ReservoirModel, this method is called with file references pointing to the reservoir properties csv file and the volume-level csv tables. The file references are handed over from the command line in the \_\_main\_\_ section of the code.

def \_\_init\_\_(self, \*args, reservoirs\_csv\_path: str, volume\_level\_csv\_paths: List[str], \*\*kwargs)

Within the \_\_init\_\_ method the following actions are carried out:

* Read the volume-level relation files for all reservoirs
* Read the csv data reservoir properties
* For each reservoir, populate the reservoir class with these properties and convert water level criteria into volume criteria using linear interpolation.

The main class contains method overrides for pre- and post- methods:

def pre(self)

def post(self)

Within the pre method the following actions are carried out:

* Setting of initial volume which is taken from a time series.

In the post method the following actions are carried out:

* Per reservoir, convert the volumes computed back to water levels using linear interpolation of the associated volume-level table.
* Call the super-method to write data to PI-xml (using the headers as available in timeseries-import.xml

Additional actions can be written to these post- and pre- methods in a similar manner.

Furthermore, there are methods to be filled with all goals:

def path\_goals(self)

def path\_constraints(self)

In the path\_goals method, the following actions are carried out, looping over all reservoirs:

* Per reservoir, identify, based on the rtcParameterConfig.xml, if the goal type needs to be added
* Obtain the priority and add the goal type using the static or transient data specified in the reservoir properties, the rtcParameterConfig.xml or the timeseries\_import.xml.

In the path\_constraints method, the following actions are carried out, looping over all reservoirs:

* Check whether initial state (volume) is within bounds (min/max volume as specified in reservoir properties file)
* Add min/max volume constraint
* Add max. Q turbine constraint
* Add max Q spill constraint
* Add constraint to minimize the absolute value of the Turbine flow derivative

The simple ReservoirModel also uses an override of the solver\_options to use an LP solver (clp) instead of the default NLP solver (ipopt). In addition is has a private method to write the goal definition and bounds definition to csv files for debugging purposes. The debug switch is set in the \_\_init\_\_ section, while the debug method is called in the path\_constrainst method.

The \_\_main\_\_ of the python script parses command line arguments and hands them over to RTC-Tools.

The command line arguments are:

--reservoirs\_csv\_path (required): Path reference to the csv file with reservoir properties

--volume\_level\_csv\_paths (required): one or more path references to csv files with volume level lookup tables

--solver (optional): option to choose solver clp (LP, default) or ipopt (NLP)

# Results of the optimization model for Hume Dam

Figure 1 shows a computation result for the model Hume dam where the observed total release from a historical event is used as user-defined target for total reservoir release. Figure 2 shows the discharge for the downstream locations at Kiewa inflow and Doctor’s point.

* The reservoir volume stays within bounds according to the applied goals. Observed volume and computed volume differ from each other. This might be related to the fact that the volume in the reservoir is not measured directly.
* The user-defined release is followed, with exception of the initial time step.
* Turbine flow is used when possible up to the maximum admission.
* The maximum discharge at Doctor’s point is exceeded most of the time.

The goals for this result are listed in Table 1.

Table 1 Goal configuration with historic reservoir release as goal. Units are m³ for volume and m³/s for flows.

|  |  |  |
| --- | --- | --- |
| Goal | Priority | Active |
| Vmin = 2∙10^9,  Vmax = surcharge volume, derived from the surcharge level via the volume-level table | 4 | Yes |
| Vmin = 2∙10^9,  Vmax = full supply volume | - | No |
| Qout with observed data or user-defined release scheme | 10 | Yes |
| Doctor’s point:  Qmin = 13.889 m³/s  Qmax = 289.325 m³/s | 11 | Yes |
| Minimize Qspill, smooth Qspill | 12 | Yes |
| Smooth Qturbine | 100 | Yes |

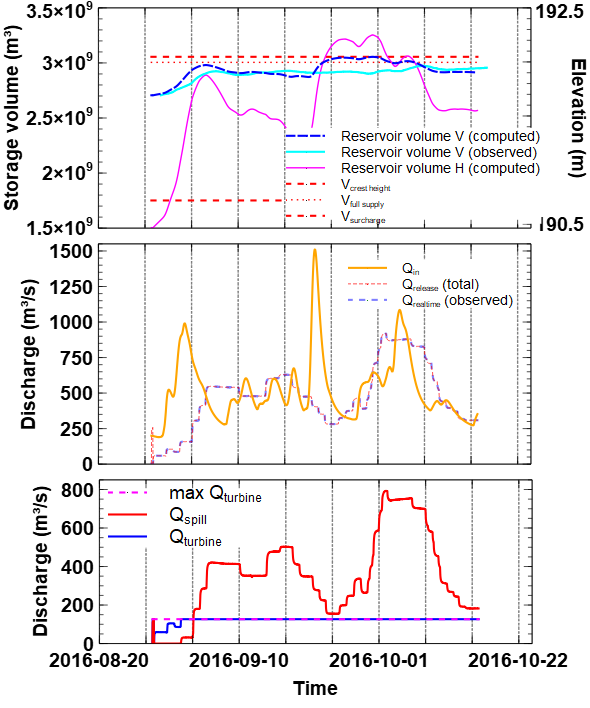


Figure 1 Optimization results of model “Hume dam” with user-defined reservoir outflow as goal for reservoir volume and water level (top), inflow and total outflow (middle) and turbine flow and spill (bottom)

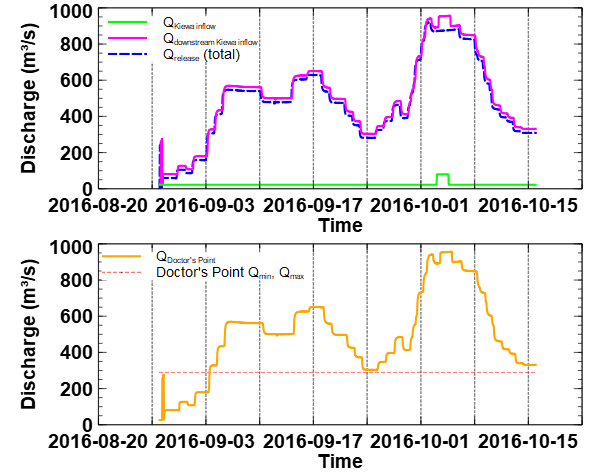


Figure 2 Discharge at Kiewa inflow (top) and Doctor’s point (bottom) with user-defined reservoir release scheme as goal

Figure 3 and Figure 4 show results where the historic reservoir release has not been applied as a goal.

* The reservoir volume reaches the maximum in the end of the simulation period.
* The total release deviates from the historic scheme and has a more continuous pattern.
* Total release starts with high values already in the beginning. The second inflow peak (very large) is already taken into account in the beginning of the forecasting period.
* The little inflow peak from the Kiewa tributary is damped at Doctor’s point, but still recognizable.
* Discharge exceeds the maximum value at Doctor’s point, but there is no distinct peak.
* End of the world behavior: Reservoir release is increased in the last time steps of the forecast period, and this release does not reach Doctor’s point within the simulation period due to the travel time of water in the downstream reaches (delay branch). The model makes use of the travel time and does not care what happens after the end of the simulation period.

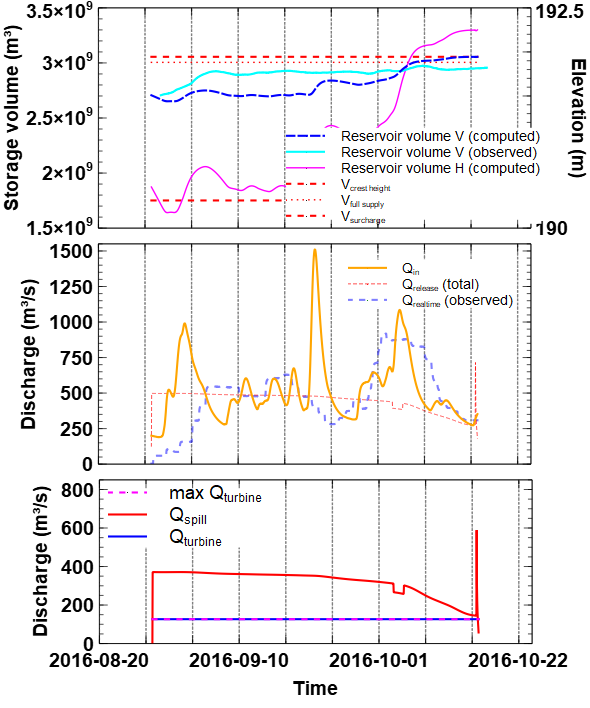


Figure 3 Optimization results of model “Hume dam” with discharge limits at Doctor’s point as primary goal for reservoir volume and water level (top), inflow and total outflow (middle) and turbine flow and spill (bottom)

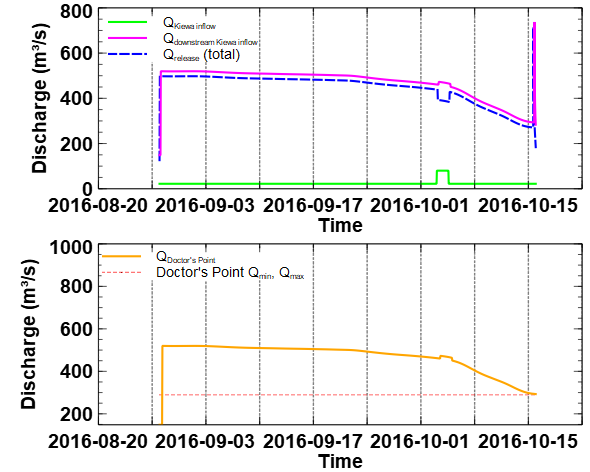


Figure 4 Discharge at Kiewa inflow (top) and Doctor’s point (bottom) without user-defined reservoir release scheme as goal

The goals for this result are listed in Table 2.

Table 2 Goal configuration with historic reservoir release as goal. Units are m³ for volume and m³/s for flows.

|  |  |  |
| --- | --- | --- |
| Goal | Priority | Active |
| Vmin = 2∙10^9,  Vmax = surcharge volume | 4 | Yes |
| Vmin = 2∙10^9,  Vmax = full supply volume | - | No |
| User defined Qout with observed data | - | No |
| Doctor’s point:  Qmin = 13.889  Qmax = 289.325 | 11 | Yes |
| Minimize Qspill | 12 | Yes |
| Smooth Qturbine | 100 | Yes |

# The RTC-Tools simulation model

## Preface

Simulation and optimization mode differ significantly in the way the control variables (the release of a reservoir) are determined.

In optimization mode, the control scheme (control variable over time) is determined for the whole simulation period. The control scheme is not known beforehand but an optimization algorithm is used to find the optimal control scheme such that all goals are met as good as possible. To evaluate if a goal is met or not, the results of the hydraulic model are evaluated against the goals. The hydraulic model computes not only volumes and discharges, but also derivatives with respect to the control variable, which helps the optimizer to proceed towards an optimal control scheme.

In simulation mode, the model proceeds from time step to time step. Each time step a control logic is applied, taken only external data or model data from previous time steps into account. Control can also be defined as external time series. The delays are implemented in very different ways between these two modes. Consequently, two variants of an RTC-Tools model are needed if both simulation and optimization mode are to be supported.

## The hydraulic layer

The hydraulic layer of the simulation model Hume dam differs from the one for an optimization model by

* The use of a Modelica block *FixedDelay* for the delay branches
* The use of a reservoir node *Reservoir\_withBottomOutlet*.

The FixedDelay has been introduced to model the time lag in simulation mode. The delay is specified in the Python code (control layer).

The Reservoir\_withBottomOutlet has another outflow component to account for the bottom outlet of Hume dam. In optimization mode we only distinguish between spill and turbine flow, and Q\_spill accounts for the discharge through the spillway and the discharge through the bottom outlet. In simulation mode it is possible to model a conditional use of spillway and bottom outlet, so it makes sense to distinguish between the three components here.

## The control layer

In simulation mode there are no goals. Control is determined with the help of conditions. Methods used in the simulation variant are

def \_\_init\_\_

def initialize

def get\_output\_variables

def update

def post

The *\_\_init\_\_* method is used to populate the reservoir class with data and also read the delays.

Within the *initialize* method the initial values are set and vectors for the FixedDelay are prepared. Under *get\_output\_variables* method additional output variables are defined for V, Qspill, Qturbine and Qbottomoutlet. The control logic is coded in the *update* method by looping over all reservoirs and allocating the reservoir release series (input) over the turbine, spill and bottom-outlet. Note that the Modelica equation set ensures that the water balance is maintained and that the volume of the reservoir is updated given the inflows and release. In the *post* method, the volume of the reservoir(s) is converted back to water levels.for all time steps. In addition, units are set based on naming conventions. Variables with ‘.Q’or ‘\_Q’ in the name are assigned the unit m3/s. Variables with ‘.V’or ‘\_V’ in the name are assigned the unit m3.

## Results of the simulation model for Hume Dam

Figure 5 shows results from a simulation run with the historic reservoir outflow.

* Observed and computed volume differ, probably because the volume has not been measured directly.
* The observed reservoir release has been set as control, consequently, observed and computed release are identically.
* Reservoir release is distributed on spillway, bottom outlet and turbines according to simple logic.

The simulation results can be used to validate a model with historic data or to compute the reservoir stage for a certain control scheme.

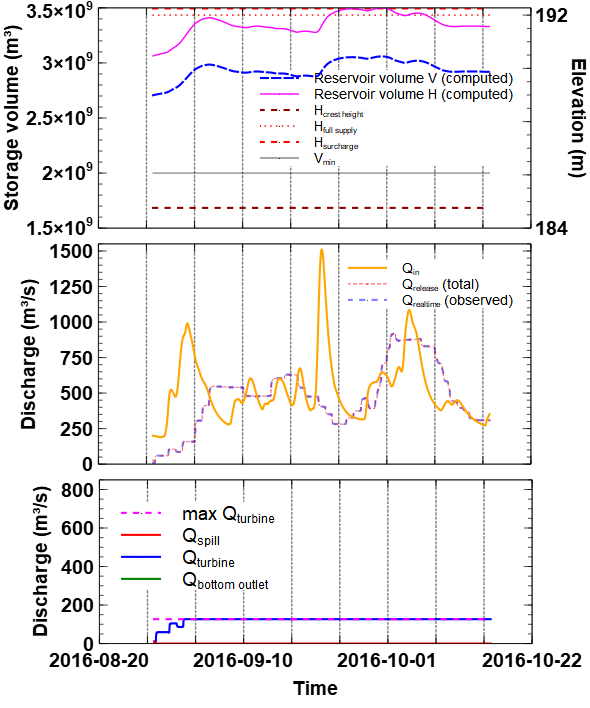


Figure 5 Simulation results of model “Hume dam” with user-defined reservoir outflow for reservoir volume and water level (top), inflow and total outflow (middle) and turbine flow and spill (bottom)

# Use of PyCharm to modify the python code

As stated in section 2, it is recommended to use PyCharm when adjusting the python code of the control layer , as it easily allows you to set breakpoints, which makes execution of the program halt at these point in debug mode. This allows you to see useful information at these points such as all variables in memory (with their values) and the current thread that is followed in python. Furthermore, you can easily go through the code line-by-line from the breakpoint onward.

In this explanation, PyCharm Community Edition 2019.1.4 is used which can be downloaded from [https://www.jetbrains.com/PyCharm/download/other.html](https://www.jetbrains.com/pycharm/download/other.html).  
Having installed the software and opened this application, choose ‘Create New Project’ from the welcome screen and fill in the location of the folder you are using for your project. Instead of choosing ‘*Create*’, click on the small triangle located under the Location-box, then choose ‘*Existing interpreter*’. Next to the address bar, you will find a ‘…’ button and pressing it will bring up a new window. Choose ‘Virtualenv Environment’ and press again on a ‘…’ button. Locate the previously made virtual environment folder you made in [the installation section](#_Installation_of_RTC-Tools). It should contain several subfolders. From within this folder, chose ‘*Scripts\python.exe*’. Then finally press ‘*Create*’ and ‘*Yes*’.  
Note that PyCharm will first need to index the virtual environment before it can be used, which might take a small amount of time. Meanwhile, you can use ‘*open*’ from ‘*File*’ in the menu-bar and open the python file you want to edit e.g., ‘Reservoir\_Optimization.py’.  
 Clicking on at the right-hand-side of a line-number will create a break point (red circle). In ‘*Run*’ at the menu-bar, you can find buttons to run or debug your python file. Choosing ‘debug’ will run the python-script until this breakpoint and useful information can be found in the docked windows at the bottom half of the PyCharm window.

An example of a useful change might be the following: Say you have modified your Modelica file and have saved it under a different name, both in terms of file name (e.g. Hume\_Reservoir\_optimizationmodel.mo) and model name in the first and last lines of the text in this Modelica file), e.g., ‘*Hume\_Reservoir\_optimizationmodel*’. Now you have two options to incorporate this in the python file:

1. Change every reference of *‘ReservoirModel*’ to ‘*Hume\_Reservoir\_optimizationmodel*’ in your python file, which can be error-prone.
2. Change the default behavior of RTC-tools:

When the name of the Modelica model (i.e. the first/last line in the Modelica file) differs from the name of this class , say “*ReservoirModel”*, the default behavior can be overridden by adding the following lines:

**def** \_\_init\_\_(self, \*\*kwargs):  
 kwargs[**"model\_name"**] = **"Hume\_Reservoir\_optimizationmodel"** super().\_\_init\_\_(\*\*kwargs)

right beneath the line which opens this main class (with the correct indentation i.e., four more than in the line which started this main class).

# Delft-FEWS

This chapter describes the steps needed in order to implement an RTC-Tools model into Delft-FEWS starting from the model prepared stand alone in the previous paragraphs.

## Analyze the dataflow into and out of the model

### Identify which timeseries are input to the RTC-Tools model.

In an RTC-Tools model, all boundary conditions are defined in the Modelica file as: “input … (fixed = true)”.

In case of the Hume Dam optimization model, the input time series are: ReservoirHume\_Inflow\_Q, Tributary\_Inflow\_Q (i.e. (Kiewa), ReservoirHume\_Release\_Q and the initial conditions ReservoirHume\_V\_init.

In case of the Hume Dam simulation model, the input series are: ReservoirHume\_Inflow\_Q, Tributary\_Inflow\_Q (Kiewa), Tributary\_Q\_delayed (Kiewa delayed), MurrayRiver1\_Q\_delayed (release delayed), MurrayRiver2\_Q\_delayed (confluence delayed), and the initial conditions ReservoirHume\_V\_init, ReservoirHume\_Qturbine\_init, ReservoirHume\_Qspill\_init

### Determine where these timeseries are generated within Delft-FEWS.

In the case of Hume Dam, forecast flows series are generated by the URBS model in ‘upper\_murray\_ToHumeDam\_URBS\_Forecast’, both for the Hume Dam inflow and the Kiewa tributary. The initial conditions come from observations i.e. ‘ProcessObserved\_FloodOps’. The release time series is manually entered. Fixed delays are applied to the release series and the tributary series to route the water to the downstream flow target location of Doctor’s Point. In case no URBS model is available, manual entry of inflow series could be a backup for Hume Dam and Kiewa.

## Implement a Delft-FEWS preprocessing module for the optimization model

The preprocessing for the optimization model should ensure that the following timeseries are provided without gaps, from T0-1 to the end of the forecast.

* ReservoirHume\_Inflow\_Q
* Tributary\_Inflow\_Q
* ReservoirHume\_Release\_Q

Each of those series is prepared the following transformations:

* Merge manual inflow forecast over observed data (if any) over URBS forecast (if any)
* Conduct linear interpolation
* Conduct constant extrapolation in both directions
* Apply default value as backup

In addition, this module should generate an initial volume, initial turbine flow and initial spill flow taking the observation with a default value as backup. The spill and turbine flow are included to guide the optimization and prevent extreme fluctuations in the first timesteps.

These transformations are implemented in module PrepareRTCTools\_HumeDam\_Optimization.

Since the model run needs to run with a single URBS forecast as the ensemble URBS forecast, all observed timeseries are identified with ensembleId ‘main’ such that this data can be integrated in the ensemble workflow.

## Implement a Delft-FEWS preprocessing module for the simulation model

The preprocessing for the simulation model should ensure that the same inflow and release timeseries, as used in the optimization model, are available without gaps from T0-1 to the end of the forecast. In addition, the following timeseries need to be prepared.

* Tributary\_Q\_delayed (i.e. Kiewa inflow delayed with 6 timesteps)
* MurrayRiver1\_Q\_delayed (release delayed with 3 timesteps)
* MurrayRiver2\_Q\_delayed (combined flow at confluence delayed with 5 timesteps)

Each of those series is prepared with the following transformations:

* Apply the time shift to the Kiewa inflow (t+6), resp. Hume dam release (t+3)
* Add up these delayed flows and apply a time shift to Doctor’s Point (t+5)

These transformations are implemented in module PrepareRTCTools\_HumeDam\_Simulation.

Since the model run needs to run with a single URBS forecast as the ensemble URBS forecast, all observed timeseries are identified with ensembleId ‘main’ such that this data can be integrated in the ensemble workflow.

## Implement the RTC-Tools 2 Delft-FEWS time series mapping

The RTC-Tools timeseries are defined in the Modelica model. The model uses a single identifier to a single timeseries which is a variable at a specific location. The Delft-FEWS PI-series format makes a distinction between location and parameter (and qualifier). To bridge the two different identification methods, RTC-Tools has introduced a mapping file called rtcDataConfig.xml (see Figure 7.1).

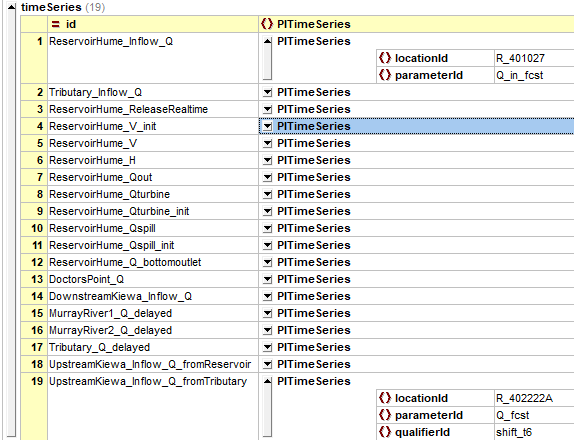


Figure 7.1 Example of the rtcDataConfig.xml to connect PI-time series to Modelica

## Compose the ModuleDataSet File

The RTC-Tools binaries and associated python installation is held in the module data set fil modules\_bin\_RTCTools.zip. This file needs to be extracted once only, either using the autoExportModuleDataSet option in the clientConfig .xml, or by adding a node in the ‘Floodops🡪Prepare System’ section of the workflows tree, with associated workflow and module instance.

|  |  |
| --- | --- |
| This section describes the composition of the ModuleDataSet file containing the full RTC-Tools model: upper\_murray\_humeDam\_RTCTools.zip.  The zip file contains the default RTC-Tools folder structure, extended with a debug folder, a src folder and two batch files, one for the simulation model and one for the optimization model. |  |

* debug folder:
  + At run-time the model will dump debug files in this folder if the debug-switch is turned on in the model script
* input folder contains:
  + file rtcDataConfig.xml, the data mapping file between Fews-timesries (location/parameter/qualifier) and RTC-Tools time series (Modelica time series ids) data
  + at run-time, Delft-FEWS will export the timeseries\_import.xml with boundary conditions and parameters.xml file with goal-settings to this folder
* model folder contains:
  + Hume\_Reservoir.mo (Modelica model for optimization)
  + Hume\_Reservoir\_\_simulation.mo (Modelica model for simulation)
  + delays.csv: data file defines delays per location for simulation model
  + reservoirs.csv: data file defines reservoir properties (e.g. min/max volume, flow rates)
  + volumelevel.csv: data file specifies volume-level relation per reservoir
  + contains sub-folder deltares\_extensions with Modelica object extensions for the simulation model: FixedDelay.mo and Reservoir-withBottomOutlet.mo
* output folder contains:
  + results and diagnostics file generated at run-time
* src folder contains python scripts:
  + Reservoir\_Optimization.py, script called to run optimization model
  + Reservoir\_Simulation.py, script called to run simulation model
  + Reservoir.py: support scripts to parse and return reservoir and delay information
  + Pandas4RTC.py, support script to transfer xml-results to csv when debug=True
* File runRTC2\_opt.bat and runRTC2\_sim.bat:
  + Batch file to start model in virtual environment with associated path references to reservoirs.csv, delays.csv and volumelevel.csv files

Table 7.1Content batch file to start simulation model

|  |
| --- |
| call .\venv\Scripts\activate && python   .\src\Reservoir\_Simulation.py --reservoirs\_csv\_path   .\model\reservoirs.csv --volume\_level\_csv\_paths   .\model\volumelevel.csv --delays\_csv\_path .\model\delays.csv   > .\RTC23sim\_stdout.txt 2>&1  exit /b "%errorlevel%" |

## Compose the ModuleParameter file

The goals for the optimization model are defined in a module parameter file (see Figure 7.2). The values in the file are inserted via location attributes. The identifiers of the parameters are constructed as a prefix of the reservoir name (as held in the reservoirs.csv data file) with a postfix referencing the goal or the priority.

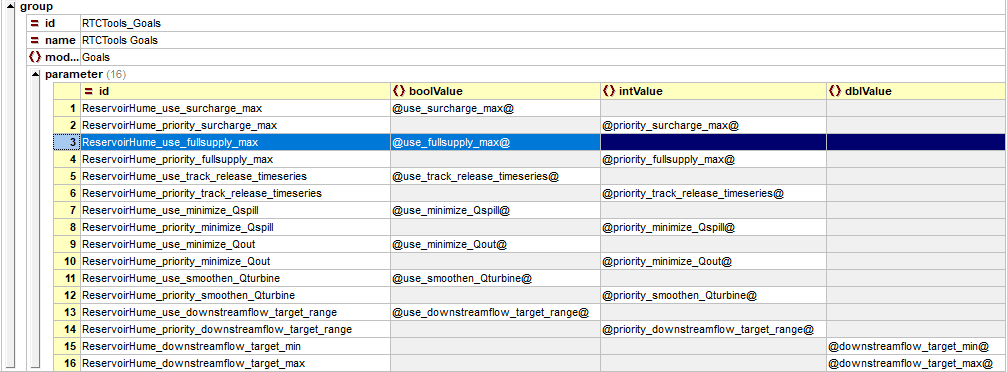


Figure . Module parameter file

## Implement a general adapter module to run the RTC-Tools optimization model

The RTC-Tools optimization model is executed via the upper\_murray\_RTCTools\_Optimization module. This general adapter module conducts the following activities:

* Export the boundary timeseries (inflow forecasts, proposed reservoir release), the initial state timeseries (V, Qspill, Qturbine) and empty series for the results#
* Exports the ModuleDataSet of the model (upper\_murray\_HumeDam\_RTCTools.zip)
* Export the Parameters.xml file by looping over the template to populate the attribute values
* Export the activation template file (held in the virtual environment) into a activate.bat file via the exportCustomFormatRunFileActivity to replace the virtual environment path placeholder by the value of global property RTCTOOLS\_VIRTUAL\_ENV.
* Call the batch file to execute the model
* Import the resulting forecast timeseries: Q\_release, Q\_spill, Q\_turbine, V\_forecast, Q(Kiewa) shifted, Qrelease shifted, Q@confluence, Q@Doctor’sPoint

# The empty timeseries for the results are exported such that RTC-tools receives the proper unit information in the header.

## Implement a general adapter module to run the RTC-Tools simulation model

The RTC-Tools optimization model is executed via the upper\_murray\_RTCTools\_Simulation module. This general adapter module conducts the following activities:

* Export the boundary timeseries (inflow forecasts, proposed reservoir release), the delayed observed flows for Hume Dam, Kiewa and Doctor’s Point and initial state timeseries (V, Qspill, Qturbine)
* Exports the ModuleDataSet of the model (upper\_murray\_HumeDam\_RTCTools.zip)
* Export the activation template file (held in the virtual environment) into a activate.bat file via the exportCustomFormatRunFileActivity to replace the virtual environment path placeholder by the value of global property RTCTOOLS\_VIRTUAL\_ENV.
* Call the batch file to execute the model
* Import the resulting forecast timeseries: Q\_release, Q\_spill, Q\_turbine, V\_forecast, Q(Kiewa) shifted, Qrelease shifted, Q@confluence, Q@Doctor’sPoint

## Compose workflows for simulation and optimization model

The RTC-Tools models are both executed in deterministic mode (using URBS Official forecasts) and in ensemble model (using the ENSEMBLE forecast) using different settings in the ensemble loop (see Figure 7.3 and Figure 7.4). Both modes are deployed as ensemble run

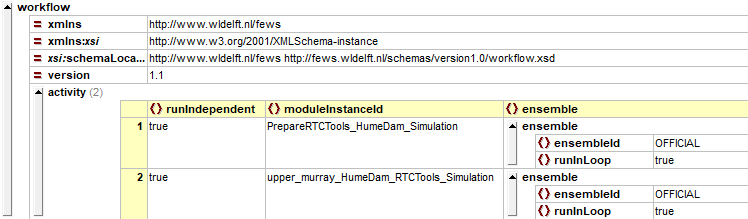


Figure 7.3 Simulation workflow definition for the (deterministic) URBS Official forecast (UpperMurray\_RTCTools\_HumeDam\_Sim\_Official.xml)

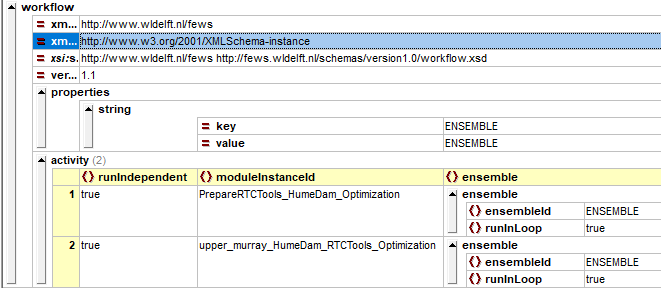


Figure 7.4Optimization workflow for the URBS ENSEMBLE forecast (UpperMurray\_RTCTools\_HumeDam\_Opt\_Ensemble.xml)

Note that module instance identifiers and workflow identifiers should be properly registered as forecast in the respective descriptors files.

## Visualizing the results

To visualize the results, a display has been composed with 5 sub-plots:

* Flows around the Dam (inflow, release, computed outflow). Note that the computed outflow may deviate from the release if the reservoir volume is reaching its extremes.
* Turbine, spill (and bottom outlet) flow
* Volume and water level
* Tributary inflow (Kiewa)
* Flow at downstream target location (Doctor’s Point)

For the ensemble workflow, the legend becomes large. Currently only the first three sub-plots are shown, although these could be swapped with other sub-plots.

## Extending the ModifierTypes

In the ModifierTypes.xml file, two Modifier Groups have been added, one for the simulation and one for the optimization (see Figure 7.5). Both reference the same time series modifier, while the Optimization group also references the location-attribute modifier for the RTC-Tools goal settings (see Figure 7.6).

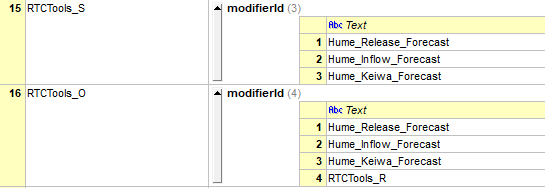


Figure 7.5Modifier groups defined for the two Hume dam models

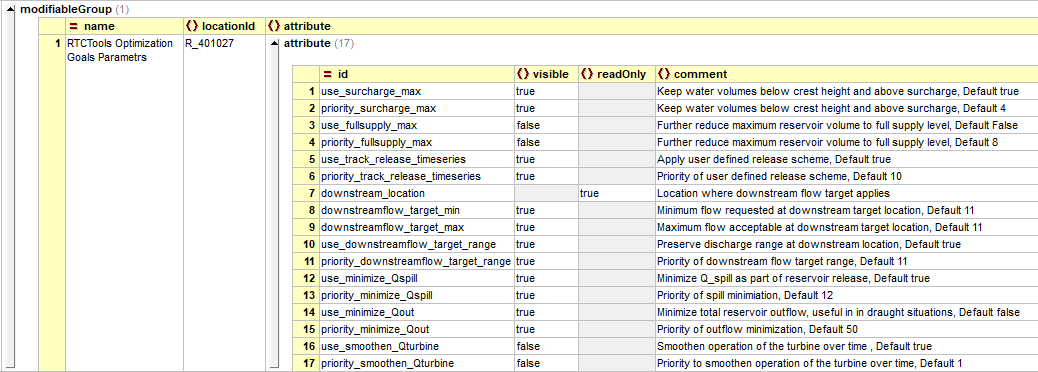


Figure 7.6Location attribute modifier for the Hume Dam optimization model

Note: when adding another Dam model to the configuration, it is recommended to introduce ModifierGroups by Dam (i.e. RTCTools\_HumeDam\_S, RTCTools\_HumeDam\_O).

## Add RTC-Tools section to topology

After the workflows, modifiertypes and displays are defined, the Workflows tree in the GUI can be adjusted by adding a section to the Topology.xml under the relevant URBS model.

|  |  |
| --- | --- |
| Following the same setup as URBS, a distinction is made between the local run (using URBS official), its persistent and shared version and the ensemble run. |  |

As can be noted in Figure 7.7 only the modifiergroup for RTCTools\_S (or RTCTools\_O) is shown, with the default modifier being the release time series.

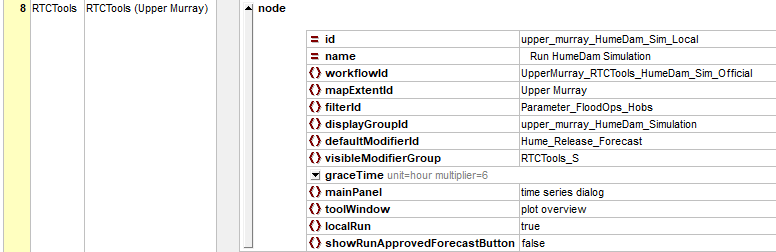


Figure 7.7 Specification of topology node in the workflow tree

# Testing RTC-Tools in Delft-FEWS

## Testing - URBS Forecast

With this test we want to verify that if no user input is present the simulation and optimization models of RTC-Tools use the forecasts from the hydrological model URBS.

|  |  |  |
| --- | --- | --- |
| Action | Pass | Remarks |
| Prepare Delft-FEWS |  |  |
| * Unzip test package, containing Delft-FEWS software, Config and LDS * Ensure T0 is set to Wed,31-08-2016 09:00:00 in the global.properties) |  |  |
| Startup ROWS 2018.02 |  |  |
| * Start the test application with: $REGION\_HOME$\ROWS.lnk |  |  |
| * When asked, upgrade the datastore to 2017.02 format. The LDS contains: rating curves and processed observations |  |  |
| * Confirm system time of ROWS is Wed 31-08-2016 09:00 AEST |  |  |
| Prepare the URBS module |  |  |
| * Open the Workflows display, run Flood Ops > Prepare system > Prepare URBS modules folder |  |  |
| Run the URBS model and save |  |  |
| * In the Workflows display, run Run the workflow “Run Upper Murray – ToHumeDam:      * Run the workflow “Save Upper Murray Forecast” |  |  |

|  |  |  |
| --- | --- | --- |
| Expected results | Ok | Remark |
| * All above workflows run without errors |  |  |
| * The model produces a forecast for water level (H) and discharge (Q) over time. |  |  |

## Test – Run RTC-Tools in simulation mode

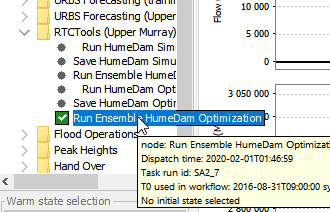
|  |  |  |
| --- | --- | --- |
| Action | Pass | Remarks |
| Starting point: URBS forecast is ready (section 8.1) |  |  |
| Run RTC-Tools Simulation |  |  |
| * In the Workflows display, run  Flood Ops > RTC-Tools (Upper Murray) > Run HumeDam Simulation |  |  |
| * Select Plots. Expected result is a display view as follows.      * There is no user-defined release scheme yet. Q\_release is extended with the last available value. Volume and the water level in the reservoir increase over time, because inflow is larger than outflow. |  |  |
| Apply a modifier for Q\_release |  |  |
| * Select the Modifier tab * Press the Hume Release Forecast button * Fill in a Q\_release of 80 000 ML/day for the entire (default) period * Apply the Modifier.      * Re-run * Switch to Plots.   Expected result is a plot like the following:     * Q\_release is as specified in the modifier, Q\_out follows the modifier time series. * The reservoir outflow Q\_out is lower than the reservoir inflow in the beginning and becomes later higher than the reservoir inflow. Consequently, volume and water level in the reservoir increase first and then decrease slowly. * If the water level is higher than the crest of the spillway (183.4 m), so both spillway and bottom outlet are used. Below the crest, only the bottom outlet is used.   Zoom into the plot   * Zoom into the graph and analyze the behavior of the spillway |  |  |

## Test – Run RTC-Tools in optimization mode

|  |  |  |
| --- | --- | --- |
| Action | Pass | Remarks |
| Starting point: Previous test completed (section 8.2) |  |  |
| Run RTC-Tools in optimization mode   * In the Workflows display, run  Flood Ops > RTC-Tools (Upper Murray) >  Run ToHumeDam – Optimization * Go to plots and check the results:      * The user-defined Q\_release does not cover the whole simulation period. As soon as the user-defined Q\_release is no longer active, the outflow drops due to the minimize release goal. * At this point in time, the outflow is lower than the inflow. Consequently, the reservoir starts to fill again. |  |  |
| De-activate goals   * Go to the Modifier tab. * Under RTCTools Goals parameters and switch off the following goals:   + Use Surcharge Target Waterlevel      * Apply and re-run. * Go to Plots and check the results:      * The reservoir water level exceeds the surcharge level of 192.25 m in the end of the simulation period. * The release is smaller because of the Minimize Release Goal, which is still active. |  |  |
| RTC-Tools find its own release scheme   * Delete all modifiers and re-run. * Go to Plots and check the results:      * The reservoir release Q\_out is chosen by RTC-Tools such that all goals are met as good as possible:   + The maximum reservoir water level of 192.25 is not exceeded   + Spill is minimized and smoothed   + The maximum flow at Doctor’s point (289.352 m³/s, 25000ML/d) is exceeded, because this goal has a lower priority than the surcharge water level in the reservoir. |  |  |
| Normal flow   * Set the system time to Thu,20-10-2016 09:00:00 * Run the URBS Forecast (see Section 8.1) * Run the workflow HumeDam Optimization * Go to the Modifiers tab, select RTCTools Goals parameters and de-select “Use Minimize Release”. Apply and re-run. * Go to Plots and check the results from the two forecasts:      * The Minimize Release Goal aims to minimize the reservoir release, and the effect is that the water level is higher in the end of the simulation period. |  |  |

## Ensemble mode

An ensemble run can be carried out with the nodes “Run Ensemble HumeDam Simulation” and “Run Ensemble HumeDam Optimization” for simulation mode and optimization mode, respectively:



For ensembles, an ensemble forecast from the hydrological model (URBS) is required. RTC-Tools carries out one individual run for each ensemble member.

This feature has not been tested in detail because of lack of ensemble forecast data.