Raven seamless integration

Contents

Raven is a robust and flexible hydrological modelling framework, designed for application to challenging hydrological problems in academia and practice. This fully object-oriented code provides complete flexibility in spatial discretization, interpolation, process representation, and forcing function generation. Models built with Raven can be as simple as a single watershed lumped model with only a handful of state variables to a full semi-distributed system model with physically-based infiltration, snowmelt, and routing.

This document describs different aspects of integrating Raven models. More information about Raven can be found on http://raven.uwaterloo.ca/Main.html

Introduction

The Raven Modelling Framework is integrated with Delft-FEWS for a variety of inflow forecasting systems. This connection is currently done with a Python based model adaptor developed by Alan Barton of the National Research Council of Canada. See Figure 1.1 for a high level schematic of the current Raven-FEWS Connections.

The adaptor has been iteratively improved to allow advanced functionality between Delft-FEWS and Raven. However there are limitations identified with the current adaptor approach, namely:

- Dependencies of the model package, including use of Python 2.7 which is no longer actively supported.
- Limitation in flexibility for new applications, which vary from the initial applications for which it was developed.
- Additional processing time in the conversion of model files.

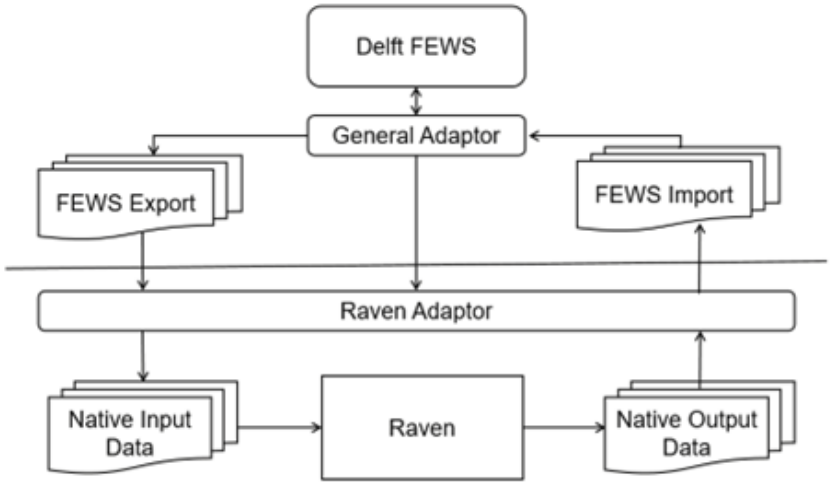

Figure 1.1: Current Raven-FEWS Connections - High Level Schematic

Recent implementations and iterations of the Raven have included the capability of netCDF reading and writing, removing the need for an intermediary adaptor for some functionality.

In collaboration with James Craig, the developer of the Raven model software, a design was made for seamless integration of Raven. In this “adaptor-less” approach the output of the FEWS general adaptor can be directly read by the Raven model. See Figure 1.2 for the desired approach.

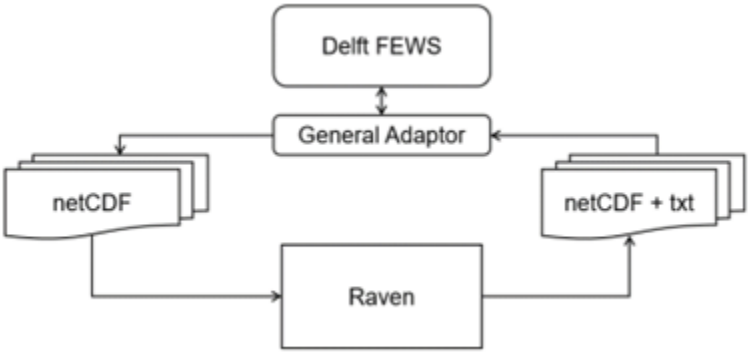

Figure 1.2: Desired Raven-FEWS Connections - High Level Schematic

This design required changes to the Raven software as well as to Delft-FEWS (only for model settings, see par 5.6). The result of this development leads to several important benefits:

- Increased performance (faster workflows),
- Simplified configuration
- More robust integration due to id mapping
- Increase of the number of parameters/variables that are allowed
- Reduced adaptor logging information, allowing for cleared debugging.

Supported by the Delft-FEWS users employing Raven models in their forecasting systems, James Craig and Deltares have implemented the new approach. This report describes the configuration needed to integrate Raven via the new approach. Because of the versatility of Delft-FEWS, deviations from the described configuration approach are possible. Most of the described Delft-FEWS functionality is available in version NA 2019.02 (and probably also 2018.02): where NA 2020.02 is required this is mentioned. The Raven version containing all the relevant functionality is: 3.66.

Module folder structure

Although different set-ups are possible, the following folder structure is used in the following examples:

- Modules\Raven\bin\Raven.exe
- Modules\Raven\[Basin]
- Modules\Raven\[Basin]\work: contains model files
- Modules\Raven\[Basin]\input: contains input files

- Modules\Raven\[Basin]\output: contains output files

This is not prescribed, but a uniform approach is advised. With the above approach, it is possible to run individual Raven simulations in the %TEMP_DIR% folder, which allows FEWS to kickoff multiple simulations in parallel by placing them in unique folders in the FEWS temp directory. This greatly speeds-up ensemble forecasts.

Timezone

In case Raven is run with daily timesteps, input timeseries should have a timezone setting without daylight saving. If timesteps for a timezone with daylight saving are defined, Raven will crash when the simulation period contains the moment when daylight saving is started or ended.

Raven executable

The Raven executable is preferably included as a ModuleDataSet in the configuration. By including this moduleDataSet in the clientconfig used by OC and FSS, the Raven executable will be deployed automatically:

```
<autoExportModuleDataSet name="Raven" exportDir="Modules/Raven/bin"/>
```

General Settings

Raven

The following general settings are needed in the rvi file for integrating a model in Delft-FEWS (see also C.1 of the Raven user manual):

```
:DeltaresFEWSMode
:WriteNetCDFFormat
:OutputDirectory ../output/
:FEWSRunInfoFile ../run_info.nc
```

The DeltaresFEWSMode has several functions:

- The model will not crash in case the first timestep of a forcing timeserie has missing values which Raven would typically complain about
- Corrects for the missing data in the first time index when checking whether the input timespan overlaps the simulation duration
- Relaxes the constraint that memory chunks are complete days, i.e., start and end at midnight
- Raven will only use location IDs for reading input data and writing output data. Without this setting, location names are used for output data.

Model RunName

There are several ways the model can be provided with a RunName:

- -the -r runname flag from the command line, or
- -the RunName attribute in the properties variable of the RunInfo.nc file
- -the :RunName command in the .rvi file

The RunName is used as prefix for the output files: this is relevant in the configuration of the ImportActivity. If no RunName is given (or has zero length), no prefix will be used. If the Runname appears in more than one of these, the precedence takes over, with command line dominating the others.

GA - General section

The general section mentioned below follows the folder structure as mentioned above:

```
<general>
            <description>Raven Model for $MODEL$</description>
            <piVersion>1.5</piVersion>
            <rootDir>%REGION_HOME%/Modules/Raven/</rootDir>
            <workDir>%ROOT_DIR%/$MODEL$/work</workDir>
            <exportDir>%ROOT_DIR%/$MODEL$/input</exportDir>
            <exportDataSetDir>%ROOT_DIR%/$MODEL$/work</exportDataSetDir>
            <exportIdMap>IdExportRaven</exportIdMap>
            <exportUnitConversionsId>ExportRaven</exportUnitConversionsId>
            <importDir>%ROOT_DIR%/$MODEL$/output</importDir>
            <importIdMap>IdImportRaven</importIdMap>
            <dumpFileDir>$GA_DUMPFILEDIR$</dumpFileDir>
            <dumpDir>%ROOT_DIR%/$MODEL$</dumpDir>
            <diagnosticFile>%ROOT_DIR%/$MODEL$/output/diag.xml</diagnosticFile> <!--ignored since Raven logfile
will be read-->
            <timeZone>
                        <timeZoneName>EST</timeZoneName> <!--Needs to be non daylight-saving-->
            </timeZone>
            <startDateTimeFormat>yyyy-MM-dd hh:mm:ss</startDateTimeFormat>
            <modelTimeStep id="day_EST"/> <!—ensures end time is full timestep-->
</general>
```

GA - startUpActivities

With the following settings, the relevant folders are cleared first.

```
<startUpActivities>
            <purgeActivity>
                        <filter>%ROOT_DIR%/$MODEL$/*.*</filter>
            </purgeActivity>
            <purgeActivity>
                        <filter>%ROOT_DIR%/$MODEL$/input/*.*</filter>
            </purgeActivity>
            <purgeActivity>
                        <filter>%ROOT_DIR%/$MODEL$/output/*.*</filter>
            </purgeActivity>
            <purgeActivity>
                        <filter>%ROOT_DIR%/$MODEL$/work/*.*</filter>
            </purgeActivity>
</startUpActivities>
```

GA – exportStateActivity

In the exportStateActivity, the current (warm or cold) state is exported. The following example is in line with the above folder structure.

```
<exportStateActivity>
            <moduleInstanceId>Raven_$MODEL$_Historical</moduleInstanceId>
            <stateExportDir>%ROOT_DIR%/$MODEL$/work</stateExportDir>
    <loopTimeStep id="day_EST"/><!-- only when daily warm state should be generated-->
            <stateSelection>
                        <warmState>
                                    <stateSearchPeriod start=”-10” end="-1" unit="day"/>
                        </warmState>
            </stateSelection>
</exportStateActivity>
```

The loopTimeStep can be used to have FEWS run the model with daily timesteps: after each daily simulation a warm state is stored in the database. So in case a simulation of 10 days is run, the user would be able to select a warm state for any of these daily timesteps as starting point for a new run. Note that this does not work well when the model is provided with state update timeseries.

The warm state selection period can be left out of the configuration if it is defined in the topology.xml or the workflowdescriptors.

GA - exportDataSetActivity

This activity exports the model itself. The model is included in a zip file in the ModuleDataSetFiles in the configuration. Generally it would include an rvi, rvp, rvt and rvh files. These files should not be in a folder in the zip file: upon extraction, the contents of the moduleDataSet will be extracted to the exportDataSetDir described in the general section.

```
<exportDataSetActivity>
            <moduleInstanceId>Raven_$MODEL$_Historical</moduleInstanceId>
</exportDataSetActivity>
```

GA - exportNetCDFRunFileActivity

Raven uses a netCDF Run Info File to determine information about the simulation such as start and end time. Other commands can be passed to Raven via the Run Info file. Commands that should be adjustable by the user via the FEWS interface should be defined as locationAttributes. The locationAttributeModifier allows for changing these settings, which are then passed to FEWS instead. This functionality was added in Delft-FEWS release 2020.02 and is therefore not available in earlier releases.

The commands that can be passed to Raven are (see also C2.1. in the Raven user manual):

- blockRavenWarnings
- BlockRavenCustomOutput (not advised because it can lead to the workflow failing if the expected output is not present)
- NoisyMode
- SilentMode
- Mode
- AssimilateStreamFlow

```
<exportNetcdfRunFileActivity>
            <exportFile>%ROOT_DIR%/$MODEL$/run_info.nc</exportFile>
            <properties>
                        <string key="RunName" value="$MODEL$"/>
<locationAttribute key="BlockRavenWarnings" locationId="$MODEL$" attributeId="BlockRavenWarnings"/>
            </properties>
</exportNetcdfRunFileActivity>
```

Required model inputs

Meteo input

Raven allows for different types of meteo input; per gauge, per spatial entity or gridded. FEWS exports timeseries in NetCDF format, being scalar or grid. The NetCDF files are exported to the *exportDir* as defined in the *general* section above. In Raven, the reference to this file is made in the rvt file. These different approaches are discussed below.

Gauges

Gauges are defined in the rvt and have this format:

```
:Gauge [gaugename]
:Latitude [latitude]
:Longitude [longitude]
```

Data for the *forcing_types* MAX_TEMPERATURE, MIN_TEMPERATURE and PRECIP need to be provided in NetCDF format (when integrated in FEWS). The definition in the rvt file is as follows (see also A.4.1 of the Raven User Manual):

```
  :Data [forcing_type]
        :ReadFromNetCDF
            :FileNameNC ../input/meteo_inputs.nc            :VarNameNC [forcing_type]
            :DimNamesNC stations time
            :PeriodEndingNC
            :StationIdx 1
        :EndReadFromNetCDF
  :EndData
```

In the above example, the data is read from the first column of the matrix with data (stations x time) in the NetCDF file. An inspection of the NetCDF file is needed in order to determine the column number. A change in the FEWS configuration might lead to a reordering of columns, which might break this mapping. To have Raven look for the correct column, use the FROM_STATION_VAR command instead:

```
:StationIdx FROM_STATION_VAR
```

With this setting, Raven will select the station in the NetCDF file that corresponds with [gaugename]. This requires settings the correct name either in the idMap in FEWS, or by changing the [gaugename] to match the station Id in the NetCDF file. To map multiple locations at once, use the

:MapStationsTo SUBBASINS or

:MapStationsTo HRUS

A typical configuration of the general Adapter would look like;

```
<exportNetcdfActivity>
            <exportFile>meteo_inputs.nc</exportFile>
            <timeSeriesSets>
                        <timeSeriesSet>
                                    <moduleInstanceId>$HIST_PRECIP_MODULE$</moduleInstanceId>
                                    <valueType>scalar</valueType>
                                    <parameterId>P.obs</parameterId>
                                    <locationSetId>precip_stations_$CATCHMENT$</locationSetId>
                                    <timeSeriesType>external historical</timeSeriesType>
                                    <timeStep id="day_MST"/>
                                    <relativeViewPeriod unit="day" end="$ENDTIME$"/>
                                    <readWriteMode>read only</readWriteMode>
                        </timeSeriesSet>
…
            </timeSeriesSets>
</exportNetcdfActivity>
```

The element in the export IdMap would be:

```
            <function internalLocationSet="precip_stations" internalParameter="P.obs" externalLocationFunction="
@Raven_ID@" externalParameterFunction="PRECIP"/>
```

Grid

Some models are set-up to directly read gridded data. In the model the HRUs are mapped to grid cells. The rvt file looks like this (from A.4.7 of the Raven User Manual):

```
:GriddedForcing    [forcing name]
:ForcingType     [type]
:FileNameNC    [path/filename of .nc file]
:VarNameNC     [name of variable in .nc file]
:DimNamesNC    x y time
:GridWeights
:NumberHRUs [total number of HRUs]
:NumberGridCells [total number of grid cells]
 [HRU ID] [Cell ID] [weight]
...
:EndGridWeights
:EndGriddedForcing
```

A typical exportActivity in the general Adapter would be:

```
<exportNetcdfActivity>
            <exportFile>meteo_input.nc</exportFile>
            <timeSeriesSets>
                        <timeSeriesSet>
                                    <moduleInstanceId>Modify_Grids_historic_$catchment$</moduleInstanceId>
                                    <valueType>grid</valueType>
                                    <parameterId>P</parameterId>
                                    <locationId>HRDPA</locationId>
                                    <timeSeriesType>external historical</timeSeriesType>
                                    <timeStep id="EST_day"/>
                                    <relativeViewPeriod unit="day" end="0"/>
                                    <readWriteMode>read only</readWriteMode>
                        </timeSeriesSet>
…
            </timeSeriesSets>
</exportNetcdfActivity>
```

Polygons

In case the meteo input per HRU is calculated in FEWS, a NetCDF with multiple timeseries can be provided as input for Raven. In that case, the rvt file would have the following setting:

```
:StationForcing PRECIP
  :ForcingType PRECIP
  :FileNameNC  ../input/precip_inputs.nc  :VarNameNC   PRECIP
  :DimNamesNC  stations time
  :PeriodEndingNC
  :RedirectToFile GriddedForcings.txt
:EndStationForcing
```

The redirected file (GriddedForcing.txt in above example) could look like this:

```
:GridWeights
   :NumberHRUs       14
   :NumberGridCells  1
   # [HRU ID] [station ID] [weight]
22020000    0     1
22020100    1     1
….
:EndGridWeights
```

The above approach requires knowledge of the order of the stations in the NetCDF file. If a configuration change leads to a change in the order of the exported stations, the GridWeights settings would have to change accordingly. Therefore a direct IdMapping is possible. In the example above, the

```
:RedirectToFile GriddedForcings.txt
```

command (which points to the :GridWeights command from within a :GriddedForcing) within the :GriddedForcing block can be replaced with the following:

```
:MapStationsTo SUBBASINS
```

Or

```
:MapStationsTo HRUS
```

In the first case, Raven expects that the number of stations is equal to the number of subbbasins in the model, with the station attribute vector filled with SubBasin IDs. In the second, the number of stations must be equal to the number of HRUs in the model, with the station attribute filled with HRU IDs. This overrides the need for custom :GridWeights commands.

This method can be applied to the:StationForcing command as well.

Optional inputs

Control inputs

Raven allows for many additional timeseries that can be used to define how reservoirs are operated (ReservoirTargetStage, OverrideReservoirFlow, etc), determine irrigation (IrrigationDemand, ReservoirDownStreamDemand, etc) and external basin inflows (BasinInflowHydrograph). These inputs are defined in the rvt file. An example for a BasinInflowHydrograph looks like this:

```
:BasinInflowHydrograph   [subbasin name]
  :ReadFromNetCDF
    :FileNameNC ../input/control_inputs.nc    :VarNameNC BASIN_INFLOW
    :DimNamesNC stations time
    :StationIdx 1
    :PeriodEndingNC
  :EndReadFromNetCDF
:EndBasinInflowHydrograph
```

In this example, Raven needs to look in the first column in the NetCDF file for the timeseries for this location/parameter (stationIdx 1). By changing this to

```
:StationIdx FROM_STATION_VAR
```

Raven will look up [subbasin name] in the NetCDF file. This requires that the mapping is correctly configured in FEWS. An example of the configuration in the general Adapter:

```
<exportNetcdfActivity>
            <exportFile>control_inputs.nc</exportFile>
            <timeSeriesSets>
                                <timeSeriesSet>
                                        <moduleInstanceId>Raven_$MODEL$_Historical_pre</moduleInstanceId>
                            <valueType>scalar</valueType>
                            <parameterId>Q.inflow</parameterId>
                            <locationSetId>RAVEN_IF_$MODEL$</locationSetId>
                            <timeSeriesType>simulated historical</timeSeriesType>
                            <timeStep id="day_EST"/>
                            <relativeViewPeriod unit="day" startOverrulable="true" start="$START$" end="0"/>
                            <readWriteMode>add originals</readWriteMode>
                                </timeSeriesSet>
…
                        </timeSeriesSets>
</exportNetcdfActivity>
```

The configuration of the exportIdmap would look like this:

```
<map internalLocation="basin" internalParameter="Q.inflow" externalLocation="[subbasin name]"
externalParameter="BASIN_INFLOW"/>
```

State Updating

The seamless Raven integration allows for updating of all state variables. In order to inform Raven that it needs to look for state variables in the NetCDF provides by Delft-FEWS, the following commands needs to be included in the rvi file:

```
:FEWSStateInfoFile ../input/state_update.nc
```

No other changes are needed in the Raven model. The export from FEWS needs to be aligned with the model set-up, that is that HRU IDs and state variables names should be exactly the same in the exported NetCDF file. IdMapping is key here. The configuration of the exportNetCDF activity could look like:

```
<exportNetcdfActivity>
            <exportFile>state_updates.nc</exportFile>
            <timeSeriesSets>
                        <timeSeriesSet>
                                    <moduleInstanceId>ExportMODS</moduleInstanceId>
                                    <valueType>scalar</valueType>
                                    <parameterId>SWE_HRU</parameterId>
                                    <locationSetId>RAVEN_HRU_$MODEL$</locationSetId>
                                    <timeSeriesType>external historical</timeSeriesType>
                                    <timeStep id="day_EST"/>
                                    <relativeViewPeriod unit="hour" end="0"/>
                                    <readWriteMode>add originals</readWriteMode>
                                    <ensembleId>main</ensembleId>
                        </timeSeriesSet>
            <omitEmptyTimeSeries>false</omitEmptyTimeSeries>
</exportNetcdfActivity>
```

The omitEmptyTimeSeries setting is required to ensure there is always a NetCDF file exported: if the file is configured in the rvi file, Raven will give a warning if no NetCDF file is present.

The exportIdmap used looks like this:

```
<function internalLocationSet="RAVEN_HRU" internalParameter="SWE_HRU" externalLocationFunction="@HRU_ID@"
externalParameterFunction="SNOW"/>
```

The external parameter “SNOW” is the exact name of the state variable used by Raven. The external locations referred to (@HRU_ID@) are exactly the same as the HRU IDs in the Raven model. As such, this export will result in a NetCDF file that contains a variable “SNOW” and locations with the same name as the HRUs. Section C.2.2. of the Raven User manual describes in detail how the provided variables are used to update the model state.

Subbasin state updating

With the new Raven version, it is also possible to update subbasin states, such as stream discharge and reservoir stage. The approach is similar to the state updating. The timeseries used for subbasin state updating need to be included in a separate NetCDF file and referenced to in the rvi file:

```
:FEWSBasinStateInfoFile  ../input/basin_state_updates.nc
```

The only variables allowed here are BASIN_OUTFLOW and RESERVOIR_STAGE, which obviously needs to be configured in the exportIdMap:

```
<function internalLocationSet="RAVEN_HW" externalLocationFunction="@RAVEN_ID@" externalParameterFunction="
RESERVOIR_STAGE" internalParameter="HW.obs"/>
```

`Parameter updating`

Similar to state updating, Raven allows for parameter updating. Parameters are expected to be provided as timeseries in NetCDF format. From the User Manual:

All names not in the model will be ignored, as will all other variables in the update file. If the parameter is in the model, the parameter value corresponding to the model start time will override the value as specified in the .rvp or .rvt file. If the value is blank, parameter updating will not be performed.

The only setting needed in the Raven model is the following line in the rvi file:

:FEWSParamInfoFile ../input/param_updates.nc

Because with model parameters there is no clear combination of location/parameter, a slightly different approach is needed to map timeseries from Delft-FEWS to model parameters. In general, in FEWS only parameters that are relevant for operational modification are configured. One example is given below. The exported timeseries contains the Percolation coefficient for INT_SOIL layer:

```
<exportNetcdfActivity>
            <exportFile>param_updates.nc</exportFile>
            <timeSeriesSets>
                        <timeSeriesSet>
                                    <moduleInstanceId>$MODIFIER_MODULE$</moduleInstanceId>
                                    <valueType>scalar</valueType>
                                    <parameterId>INT_SOIL_PERC_RVP</parameterId>
                                    <locationId>$CATCHMENT$</locationId>
                                    <timeSeriesType>external historical</timeSeriesType>
                                    <timeStep id="day_MST"/>
                                    <relativeViewPeriod unit="day" end="$ENDTIME$"/>
                                    <readWriteMode>add originals</readWriteMode>
                        </timeSeriesSet>
                        ,,,
            </timeSeriesSets>
</exportNetcdfActivity>
```

Also here IdMapping has a key role, especially since the variablename in the netCDF file needs to meet the requirements specified in section C.2.4 of the Raven user Manual.

```
<function internalLocationSet="catchments" internalParameter="INT_SOIL_PERC_RVP" externalLocationFunction="
INT_SOIL" externalParameterFunction="PERC_COEFF_in_INT_SOIL"/>
```

The value for the externalLocation is irrelevant: Raven only searches for the variable.

Running Raven in ensemble mode

Parallel simulation

Because Raven is not a heavy model, it is very suitable for running in parallel when ensembles are run. When the model is run in ensemble mode, the *root Dir* in the *general* section of the *general Adapter* (see par 5.3) can be directed to the temporary folder:

```
<rootDir>%TEMP_DIR%/Raven/</rootDir>
```

This is the *temp* folder in the application root folder. The model for each ensemble member will be placed in a unique folder. In combination with the runInLoopParallelProcessorCount setting in the global.properties, this setting allows multiple simulations to take place in parallel.

Realization dimension in NetCDF

Until 2019.02, NetCDF files generated as part of an ensemble Loop would contain a realization dimension, even if it would contain only one ensemble member. This required a separate rvt file for models that were run in ensemble mode. In Delft-FEWS 2020.02 it is possible to instruct FEWS to not include this realization dimension:

```
<exportNetcdfActivity>
        <exportFile>precip_inputs.nc</exportFile>
        <writeRealizationDimension>false</writeRealizationDimension>
        <timeSeriesSets>
                <timeSeriesSet>
                        ….
                </timeSeriesSet>
        </timeSeriesSets>
</exportNetcdfActivity>
```

Because of this setting, in many cases the same moduleDataSet can be used for deterministic and probabilistic simulations.

Including multiple modes in one model

With the recent Raven version, it is possible to include multiple settings that can be used in different runs. The new functionality is the *Mode* command (section A.1.8 of the User Manual):

```
:IfModeEquals A # use target stage for reservoir control
   :ReservoirTargetStage 220300
    :ReadFromNetCDF
     :FileNameNC ../input/control_inputs.nc     :VarNameNC TARGET_STAGE
     :DimNamesNC stations time
     :StationIdx FROM_STATION_VAR
     :PeriodEndingNC
    :EndReadFromNetCDF
  :EndReservoirTargetStage
:EndIfModeEquals

:IfModeEquals B # use outflow for reservoir control
  :OverrideReservoirFlow 220300
    :ReadFromNetCDF
     :FileNameNC ../input/control_inputs.nc     :VarNameNC RESERVOIR_OUTFLOW
     :DimNamesNC stations time
     :StationIdx FROM_STATION_VAR
     :PeriodEndingNC
    :EndReadFromNetCDF
:EndIfModeEquals
```

The *Mode* is a setting that can be provided in the runinfo.nc file (par 5.7) as a property. Equally to the other settings, this can be modified in FEWS as a location Attribute.

The *IfModeEquals* can be used in every Raven file, except the RVI file.

Executing the model

Because of the seamless integration, no model adapter is needed anymore. With the following configuration, the model will be executed. The Raven logfile is read directly by FEWS:

```
<executeActivity>
            <command>
                        <executable>%REGION_HOME%/Modules/Raven/bin/Raven.exe</executable>
            </command>
            <arguments>
                        <argument>$MODEL$</argument>
            </arguments>
            <logFile>
                        <file>%ROOT_DIR%/$MODEL$/output\Raven_errors.txt</file>
                        <errorLinePattern>ERROR*</errorLinePattern>
                        <warningLinePattern>WARNING*</warningLinePattern>
                        <infoLinePattern>ADVISORY*</infoLinePattern>
                        <debugLinePattern>DEBUG*</debugLinePattern>
            </logFile>
            <timeOut>3000000</timeOut>
            <ignoreDiagnostics>true</ignoreDiagnostics>
</executeActivity>
```

The argument needs to be the name of the model. Logfile settings can be changed according to specific needs, more information can be found here: https://publicwiki.deltares.nl/display/FEWSDOC/05+General+Adapter+Module#id-05GeneralAdapterModule-logFile

Importing results

Model state

In case the simulation is used to create an updated model state, the importActivities include an importStateActivity. The Raven model state is stored as a [model]_solution.rvc in the output folder. The following configuration reads in the file and stores it as a model state:

```
<importStateActivity>
            <stateFile>
                        <importFile>%ROOT_DIR%\$MODEL$/output/$MODEL$_solution.rvc</importFile>
                        <relativeExportFile>$MODEL$.rvc</relativeExportFile>
            </stateFile>
</importStateActivity>
```

Simulation results

Raven exports all relevant timeseries in NetCDF format. These can be imported in Delft-FEWS in the importActivities as normal NetCDF import. The export files are placed in the *Output* folder and have as prefix the RunName of the simulation.