Runs in ICM Exchange

# 



## Contents

[Introduction 3](#_Toc16239999)

[Creating runs 3](#_Toc16240000)

[Running simulations 5](#_Toc16240001)

[Exporting results 6](#_Toc16240002)

[Exporting to GIS 6](#_Toc16240003)

[CSV exports 7](#_Toc16240004)

[Binary export 7](#_Toc16240005)

# Introduction

Refer to the main Exchange documentation for the objects, their methods and the parameters of those methods.

In the examples it should, I hope, be obvious where the lines have overflowed onto the next line.

Some of the examples are snippets and are not designed to run on their own.

## Creating runs

This is an example of setting up runs, setting up one run for each combination of 4 levels, 4 ground infiltrations and two rainfall events.

db=WSApplication.open

group\_name = '>MODG~Sensitivity'

variables\_name = '>MODG~Variables'

runs\_name = '>MODG~RUNS'

levels=['none','38.5','39.5','40.5']

gis=['none','25','50','75']

rainfall\_names=['Dry','Wet']

run\_group=db.model\_object group\_name+runs\_name

rainfalls=Array.new

rainfall\_names.each do |rn|

rainfalls << group\_name+'>MODG~Rainfall>RAIN~RAIN 5YR 2hr '+rn

end

levels.each do |l|

level=nil

ic2d=nil

if l!='none'

level=db.model\_object group\_name+variables\_name+'>LEV~LEV\_'+l

ic2d=db.model\_object group\_name+variables\_name+'>IC2D~IC2D\_'+l

end

gis.each do |gi\_name|

gi=nil

if gi\_name!='none'

gi=db.model\_object group\_name+variables\_name+'>IFN~GI\_'+gi\_name

end

runParamsHash=Hash.new

runParamsHash['ExitOnFailedInit']=true

runParamsHash['Duration']=24\*60

runParamsHash['DurationUnit']="Hours"

runParamsHash['ResultsMultiplier']=5

runParamsHash['TimeStep']=60

runParamsHash['StorePRN']=true

runParamsHash['Level']=level

runParamsHash['Ground Infiltration']=gi

runParamsHash['Initial Conditions 2D']=ic2d

run\_group.new\_run "level #{l},gi #{gi\_name}",group\_name+'>NNET~Network',nil,rainfalls,nil,runParamsHash

end

end

Notice that one run is created for each pair of ground infiltration and levels (amongst the many parameters put into the parameter hash) – then the rainfall events are put into an array which is passed into the new\_run method as the 4th parameter.

This code takes a run, changes one parameter (the results multiplier) and creates an otherwise unchanged run. As you can see the complexity here is the need to take all the sims which are each for one rainfall event and one scenario and recreate those lists as parameters for new\_run.   
In this case I have used the IDs of the run being copied and the model group into which the new run is being created – there are of course other ways of doing this – see the main Exchange documentation for details

db=WSApplication.open

moParent=db.model\_object\_from\_type\_and\_id('Model Group',59)

moCopyThis=db.model\_object\_from\_type\_and\_id('Run',181)

eventsHash=Hash.new

scenariosHash=Hash.new

moCopyThis.children.each do |c|

thisEvent=c['Rainfall Event']

if !thisEvent.nil?

eventsHash[c['Rainfall Event']]=0

end

scenariosHash[c['NetworkScenarioUID']]=0

end

events=nil

if !eventsHash.empty?

events=Array.new

eventsHash.keys.each do |k|

events << k

end

end

scenarios=nil

if !scenariosHash.empty?

scenarios=Array.new

scenariosHash.keys.each do |k|

if k.nil?

scenarios << 'Base'

else

scenarios << k

end

end

end

params=Hash.new

db.list\_read\_write\_run\_fields.each do |p|

params[p]=moCopyThis[p]

end

network=moCopyThis['Model Network']

commit\_id=moCopyThis['Model Network Commit ID']

params['ResultsMultiplier']=678

newRun=moParent.new\_run("Example2",network,commit\_id,events,scenarios,params)

Here are examples of two of the water quality parameters being set – see the appendix of the main documentation for details.

params['QM Pollutant Enabled']=['DO\_','PH\_']

params['Sediment Fraction Enabled']=[false,false]

## Running simulations

There are essentially two ways of doing this, one of them is synchronous, one is asynchronous.

To run a run synchronously (notice that the run\_ex method is a method of the simulation, therefore if the run has multiple simulations you need to loop through them)

newRun.children.each do |c|

c.run\_ex('.',1)

end

To run two sims asynchronously – again I am using the sim IDs

puts WSApplication.version

db=WSApplication.open

WSApplication.connect\_local\_agent(1)

sims=Array.new

(0..2).each do |i|

mo=db.model\_object\_from\_type\_and\_id 'Sim',97+i

sims << mo

end

handles=WSApplication.launch\_sims(sims,'.',false,0,0)

puts WSApplication.wait\_for\_jobs(handles,true,2147483647)

puts 'done'

The above code waits indefinitely, to wait for 60 seconds and then cancel any that haven’t finished you can do something like this

puts WSApplication.version

db=WSApplication.open

sim1=db.model\_object\_from\_type\_and\_id 'Sim',163

sim2=db.model\_object\_from\_type\_and\_id 'Sim',164

sim3=db.model\_object\_from\_type\_and\_id 'Sim',165

sim4=db.model\_object\_from\_type\_and\_id 'Sim',166

sims = Array.new

sims << sim1

sims << sim2

sims << sim3

sims << sim4

WSApplication.connect\_local\_agent(1)

handles=WSApplication.launch\_sims sims,'.',false,0,0

handles.each do |h|

puts "handle #{h}"

end

index=WSApplication.wait\_for\_jobs handles,true,60000

handles.reverse.each do |h|

puts "cancelling #{h}"

WSApplication.cancel\_job h

end

## Exporting results

### Exporting to GIS

The export of results to GIS is done using the results\_GIS\_export method of the sim.

Here is an example of exporting the maximum 2D elements for all the runs in a model group

exportHash=Hash.new

exportHash['Tables']=['\_2DElements']

@group.children.each do |c|

if c.type=='Run'

c.children.each do |sim|

sim.results\_GIS\_export 'shp','Max',exportHash,@resultspath

end

end

end

This rather contrived example demonstrate the parameters for this method and some supporting functionality by taking a simulation and

1. Listing the timesteps
2. Listing the results attributes
3. Listing the tables available to export
4. Performing a shape file export for the listed tables, creating some expression fields for the 2D zones, using the alternative naming system (see the main documentation), and outputting the results for timesteps 0, 2, 4 and 6 (counting from 0) as well as the maximum results

puts WSApplication.version

db=WSApplication.open nil,true

mo=db.model\_object\_from\_type\_and\_id 'Sim',163

arr=mo.list\_timesteps

arr.each do |ts|

puts ts

end

arr=mo.list\_results\_attributes

arr.each do |e|

puts '----'

puts e[0]

puts '\*\*\*\*'

puts e[1]

end

arr=mo.list\_results\_GIS\_export\_tables

arr.each do |t|

puts t

end

params=Hash.new

params['Tables']=['\_2DElements','\_links','hw\_bridge\_opening']

params['2DZoneSQL']=[['one','1'],['two','2'],['three','678',3],['params','sim.depth2d + 234',5]]

params['ExportMaxima']=true

params['AlternativeNaming']=true

mo.results\_GIS\_export 'SHP',[0,2,4,6],params,'c:\\temp\\s'

### CSV exports

This example uses the results\_csv\_export\_ex method to export only the ds flow and ds velocity results for links for a simulation

puts WSApplication.version

db=WSApplication.open nil,false

mo=db.model\_object\_from\_type\_and\_id 'Sim',163

arr=mo.list\_results\_attributes

puts arr.to\_s

mo.results\_csv\_export\_ex nil,[["Link",["ds\_flow","ds\_vel"]]], 'c:\\temp'

### Binary export

It is possible to export files in a documented binary file format. The documentation and an example Ruby script are available on request

This example exports the flood depth result for all nodes to this format.