ICM Exchange / InfoAsset Exchange / WSPro Exchange

Version 2021.8

Innovyze®

Empowering water experts

October 2021

Table of Contents

Disclaimer

We do not guarantee the information in this document. The presence of a class or method in this document does not guarantee it will exist and/or work as described.

Introduction

This document describes the Ruby interface for InfoWorks ICM, InfoAsset Desktop and WS Pro. It assumes a degree of familiarity with Ruby and with the terminology of object-oriented programming e.g. 'class','method'.

UI and Exchange

Scripts may be run both from

a) the user interface of InfoWorks ICM, InfoAsset Desktop and WS Pro
b) the separately licensed InfoWorks ICM Exchange, InfoAsset Exchange and WS Pro Exchange products, which run from the command line without displaying a graphical user interface.

Whilst there is a great degree of overlap between the functionality available when running scripts from the user interface and that available within the Exchange products, some functionality is only available with one or the other. The distinction between the two is broadly speaking as follows:

1. Within the user interface, scripts are run when a GeoPlan is open displaying a network, possibly with a guest network loaded. The scripts may manipulate the data in the network, perform imports and exports using the Open Data Import and Export centres, commit and revert changes, and display some simple user interface features. The scripts may not manipulate tree objects except selection lists, and may not open and close databases, set up and run simulations etc.
2. From the Exchange applications, users may in addition manipulate tree objects, create databases, set up and run simulations etc., however the user interface items available from the ICM / InfoAsset Desktop / WS Pro Exchange user interfaces will not be available.

Important note – users must note that the flexible nature of the Ruby scripting language means that the language is almost infinitely flexible, the embedding of the language within the InfoWorks ICM / InfoAsset Desktop / WS Pro applications is intended primarily for the manipulation of data via the product's APIs, and it may not be possible to get other Ruby libraries etc. working within InfoWorks without problems and it will not be possible to provide support for such attempts. If you aspire to use Ruby within the Innovyze products for tasks not centred on the products' APIs you should consult with Innovyze to get advice as to the feasibility of what you are intending to attempt.

Notational Conventions

Example code will be shown in a non-proportional font e.g.

```
puts 'Hello Word'
```

Running Scripts

Running Scripts from the IExchange programs

The Exchange applications are run by running the IExchange program from the command line with suitable arguments. The two required arguments are the script name and the application code.

Relative paths are permitted for the script names but if you are running a script from the current working directory then this follows the convention (inherited from the 'normal' Ruby program) of requiring the script name to be preceded by ./

The application code are:

ICM for ICM Exchange

IA for InfoAsset Exchange

WS for WS Pro Exchange

Note that the applications are separate i.e. if you have installed the Innovyze Workgroup Client for ICM and/or InfoAsset Manager you can use the first 2 command line arguments (if you have the appropriate licences), and if you have installed WS pro you can use the 3rd.

e.g.

IExchange c:\temp\script.rb ICM
IExchange ./script.rb IA

IExchange \\server\dir\script.rb WS

It is possible to provide more arguments to the script with the extra arguments following the application code e.g.

IExchange c:\temp\1.rb WS one two three four

The normal ruby mechism is used here of passing the arguments in an array of strings called ARGV. Note that the application code IS included in that array so with the example above the following script

```
puts ARGV.size
(0...ARGV.size).each do |i|
      puts "#{i} '#{ARGV[i]}'"
end
```

will give the following output

```
5
0 'WS'
1 'one'
2 'two'
3 'three'
4 'four'
```

Running scripts from the UI

Ruby scripts may be run from the user interface when a network of any type is open.

The menu items to run them can be found on the 'Network' menu.

"Run Ruby script..." allows the user to select a script file, the default suffix being ".rb".

Once scripts are run the output directed to 'standard output' e.g. by using the command puts (e.g. '`puts 'hello world'`) is displayed. This output is not currently displayed synchronously as the script is run, though this may change in a future release of the software.

The last ten scripts run are saved in a list and accessed via the 'Recent scripts' menu item.

In all 3 UI products it is possible to configure user actions to run Ruby scripts.

The actions appear in the 'Actions' menu and toolbar and are configured using the 'User custom actions' and 'Shared custom actions' menu items on the 'Master database settings' menu item of the 'File' menu. see the product's help for further details.

In ICM and InfoAsset Manager it is also possible to configure a list of 'add ons' that can be run via an extra menu – see Appendix 5.

The Exchange Object Model

The classes provided are as follows:

WSApplication

The WSApplication class represents the top-level of the application. It consists purely of class methods. These methods fall into three categories:

a) Ones that get and set settings global to the application
b) Ones that create and open databases represented by the WSDatabase class
c) Running simulations – these methods allow access to advanced features of running simulations via the simulation agent.

WSDatabase

The WSDatabase class represents master and transportable databases.

WSModelObjectCollection

The WSModelObjectCollection class represents collections of objects of class WSModelObject and the classes derived from it, WSNumbatNetworkObject (ICM and InfoAsset), WSNetworkObject (WS Pro) and WSSimObject.

WSModelObject

The WSModelObject class represents individual tree objects e.g. selection lists, stored queries, ICM runs etc.

When one of the methods of WSDatabase / WSModelObject returns a model object, the actual class of the object returned will be determined at run time and an object of the appropriate type will be returned e.g. WSNumbatNetworkObject for network types, WSSimObject for simulations.

WSNetworkObject *(WS Pro only)*

The WSNetworkObject class is derived from the WSModelObject class. It represents the lock version-controlled objects in WS Pro.

WSNumbatNetworkObject

The WSNumbatNetworkObject class is derived from the WSModelObject class. It represents networks in ICM and InfoAsset, specifically both types of asset network and the modelling network. It also represents the merge version-controlled objects in WS Pro.

WSSimObject

The WSSimObject class is derived from the WSModelObject class. It represents an InfoWorks ICM simulation object. In ICM, it can also represent the Risk Analysis Results and Risk Analysis Sim objects.

WSRiskAnalysisRunObject *(ICM Only)*

The WSRiskAnalysisRunObject class is derived from the WSModelObject class. It represents an InfoWorks ICM risk analysis run object.

WSRunScheduler *(WS Pro only)*

WS Pro Exchange works slightly differently from ICM in its mechanism for setting up runs, making use of a specific class to do so rather than having a method of WSModelObject.

WSRun *(WS Pro only)*

In WS Pro the runs are of class WSRun, derived from WSModelObject. They have one additional method, 'run'.

WSOpenNetwork

The WSOpenNetwork class also represents networks. The distinction between the two classes is described below.

WSRowObjectCollection

The WSRowObjectCollection class represents a collection of objects which is designed to be iterated through. Obtaining a WSRowObjectCollection and iterating through it is more efficient than obtaining a vector of WSRowObject objects.

WSRowObject

The WSRowObject class represents individual objects in the network e.g. a node, a CCTV survey etc.

When one of the methods of WSOpenNetwork, WSRowObjectCollection, WSRowObject etc. returns an object in the network, the actual class of the object will be determined at run time and an object of the appropriate type will be returned e.g. WSNode for a node, WSLink for a link or WSRowObject otherwise.

WSNode

The WSNode class is derived from the WSRowObject class. It represents individual nodes in the network.

WSLink

The WSLink class is derived from the WSRowObject class. It represents individual links in the nework.

WSStructure

The WSStructure class represents the data stored in a 'structure blob' field in a WSRowObject (or derived class). It is largely a collection class, containing a collection of WSStructureRow objects, each of which represents a single row in the 'structure blob'.

WSStructureRow

The WSStructureRow class represents a row in a 'structure blob' field in a WSRowObject (or derived class).

WSTableInfo

The WSTableInfo class represents information about the table i.e. about the table itself rather than individual objects contained in that table in a particular network.

WSFieldInfo

The WSFieldInfo class represents information about a field i.e. about the field for the table rather than the field for an individual object contained in that table.

WSSWMMRunBuilder

SWMM runs works slightly differently from InfoWorks runs in ICM. The mechanism for setting up runs makes use of a specific class to do so rather than having a method of WSModelObject. It is also slightly different in detail from the WSRunScheduler for WS Pro runs.

WSCommits

The WSCommits class is a collection class represents the information about the collection of commits for a network. This applies to all version-controlled objects in ICM and InfoAsset and merge version-controlled objects in WS Pro.

WSCommit

The WSCommit class represents the information about one of the commits for a network.

This applies to all version-controlled objects in ICM and InfoAsset and merge version-controlled objects in WS Pro.

WSValidations *(ICM / WS Pro only)*

The WSValidations represents the results generated by the validation of a network. It is essentially a collection class, containing a collection of WSValidation objects, each of which represents a single validation message which would appear in a separate row if the validation were performed within the user interface of the software.

WSValidation *(ICM / WS Pro only)*

The WSValidation class represents the information about one of the individual validation messages found when performing the validation.

WSRunScheduler *(WS Pro only)*

This class is used for setting up runs in WS Pro Exchange. The mechanism used is different from that followed in ICM Exchange.

WSRun *(WS Pro only)*

This class is used to do the actually running of runs in WS Pro Exchange.

WSNumbatNetworkObject / WSNetworkObject vs WSOpenNetwork

Networks (model and asset) are represented by three object types: WSNumbatNetworkObject, WSNetworkObject and WSOpenNetwork. These may be thought of as corresponding to the difference between a file on a disk, which has attributes such as its name, size etc., and an open file handle which can be read from and/or written to. Alternatively, it may be viewed as the difference between a closed network viewed in the explorer tree and a network which is opened. Most operations looking at or altering the network data must be performed on a WSOpenNetwork, although there is some overlap as some methods that affect the network data, particularly those used in InfoAsset Exchange, are also available as methods of the WSNumbatNetworkObject class.

The process of getting a WSOpenNetwork from a WSNumbatNetworkObject / WSNetworkObject may be thought of as being akin to opening a network from the tree in the software by dragging it to the background, or right-clicking on it and then selecting the 'Open' menu item.

Methods

Unless noted, methods return nil.

In the examples below example paths are given, these will of course be different in real-life cases.

Dates and Times

The DateTime class provided by the Ruby core library is used to represent dates and times. Accordingly, if you use any methods getting, setting or otherwise returning dates you will need to include the date header using

```
require 'date'
```

The default to_s implementation for the DateTime class can be overridden to provide a more normal date and time format e.g. as follows

```
class DateTime
      def to_s
            return strftime('%d/%m/%Y %H:%M:%S')
      end
end
```

Ruby's behaviour means that you can add this method in your code and have this method called as you would hope.

You will typically want to create DateTimes using the new method which is of the form

DateTime.new(year,month,day,hours,minute,seconds) e.g.

```
myDate=DateTime.new(2012,7,23,12,34,56.789)
```

As ICM Exchange does not have a Ruby data time class specifically to represent the use of times in ICM simulations, in which both relative times and absolute times are used, the following convention is used:

Absolute times are represented as a DateTime object, relative times as a negative double – a time in seconds.

This does not apply to WS Pro, which does not have relative times.

Handling objects at the master database level

By 'at the master database level' we mean at the level of handling objects that appear in the database tree rather than within an individual network.

By and large this functionality is only available in ICM Exchange, InfoAsset Exchange and WS Pro Exchange.

The classes that are of most relevance here are WSDatabase, WSModelObjectCollection and WSModelObject.

Objects in the tree are typically represented two ways – by type of object (e.g. is is a Run, an Asset Group, a Selection List etc.) and ID, a number which appears in the property sheet when 'properties' is selected for an object in the tree, or by a scripting path. The scripting path can be thought of as being roughly analogous to the path of a file e.g. 'C:\program files (x86)\innovyze\innovyzewc.exe. They are described in more detail below.

If I have the scripting path for an object, either obtained using a Ruby method, or worked out 'by hand', you can get access to the object and then call whatever methods you desire on it.

For example (in InfoWorks ICM, though the principle is applicable to the other products) if I know the scripting path of a simulation is

```
>MODG~Basic Initial Loss Runs>MODG~Initial Loss Type>RUN~Abs>SIM~M2-
60
```

and I wish to export a full binary results file, I can simply write the following script:

```
db=WSApplication.open
 mo=db.model_object '>MODG~Basic Initial Loss Runs>MODG~Initial Loss
Type>RUN~Abs>SIM~M2-60'
mo.results_binary_export nil,nil,'d:\\temp\\sim.dat'
```

When developing scripts, or when running an ad-hoc script, a 'quick and dirty' way of identifying an object can often be by using its type and ID.

For example, if I want to export a rainfall event to a CSV file, I can simply look up its ID in the tree in the user interface and write the following script:

```
db=WSApplication.open
mo=db.model_object_from_type_and_id 'Rainfall Event',18
mo.export 'd:\\temp\\myfile.csv','csv'
```

Clearly this can be done in the user interface, but simple scripts like this can form the building blocks for more complex scripts in which you process the files in Ruby, or use Ruby to call other programs via COM interfaces or similar.

As well as obtaining objects by path and type and ID it is also possible to obtain them by type and name using the find_child_model_object method of a group, or find_root_model_object of a database.

It is possible to find all the objects in the root of the database using the root_model_objects method of the database e.g.

```
db=WSApplication.open
db.root_model_objects.each do |o|
      puts o.path
end
```

Similarly, it is possible to find all the objects which are children of a given object using the children method of the parent object.

This code, therefore, finds all the objects which are children of objects in the root of the database.

```
db=WSApplication.open
db.root_model_objects.each do |o|
      o.children.each do |c|
            puts c.path
      end
end
```

These methods can be used recursively to find all the objects in the database. The technique used in the example below is a 'breadth first search' i.e. we start by finding the objects in the root of the database and putting them in an array. Thereafter we take the first object in the array, find its children, add them onto the end of the array and remove the first object.

```
db=WSApplication.open
toProcess=Array.new
```

```
db.root_model_objects.each do |o|
      toProcess << o
end
while toProcess.size>0
      working=toProcess.delete_at(0)
      puts working.path
      working.children.each do |c|
            toProcess << c
      end
end
```

Note use of delete_at(0) which returns the first item in the array whilst at the same time removing it from the array – the old second item becomes the first etc.

In both the examples above the snippets of code output the paths of the found objects but in real life you will want to do something else with each object found.

Where WSModelObject objects are specified in parameters of other methods, they can be passed as

- A ruby object
- The object's ID (if the parameter can only be of one object type)
- The path of the object

Scripting paths

The purpose of a scripting path is to uniquely identify an object in a database by giving its name and type, the name and type of the group in which it is contained if any, the group in which that group is contained if any and so on. This is very like the way the path of a file gives the name of the file, the name of the directory in which it is contained, the name of the directory in which that directory is contained and so on.

Since, however, it is possible to have objects of the same name of different types in the same group (e.g. you can have a stored query and a selection list both called 'My Nodes' in the same group), the names need to be made non-ambiguous by adding the object types to them.

The paths always begin with >, then each object in the tree is formed by taking the object type's 'short code' (as given in the table below), following this with a ~ then adding the name so, for example, a rainfall event 'Winter 5 20' in a model group 'West' in a master group 'General' has the path:

```
>MASG~General>MODG~North>RAIN~Winter 5 20
```

If the name of any object in the scripting path for an object contains the characters ~ or >, then those characters are 'escaped' with a backslash. The backslash character is also escaped with another backslash. e.g. a master group with the unlikely name

```
My Master >>>~~~\\ Group
```

Will have the path

```
>MASG~My Master \>\>\>\~\~\~\\\\ Group
```

See appendix 9 for the short codes

Handling objects within a network

By 'within a network' we refer to the navigating between, adding, deleting and altering objects within a network, e.g. adding a node, changing the ground level of a node etc.

The most relevant classes here are WSOpenNetwork, WSRowObject collection and WSRowObject and the classes derived from it.

Before operating on individual objects within a network it is necessary to object a WSOpenNetwork object. The mechanism for doing this is different between use of the Ruby scripting functionality within the software's user interface and via the Exchange products:

Within the user interface, obtain the WSOpenNetwork object for the current network by using the current method of the WSApplication class i.e.

```
on=WSApplication.current_network
```

From the Exchange products you should obtain the WSOpenNetwork by use of the open method of the WSNetworkObject or WSNumbatNetworkObject class i.e.

```
db=WSApplication.open
net=db.model_object_from_type_and_id 'Model Network',2
on=net.open
```

Having obtained the WSOpenNetwork object it is possible to obtain WSRowObjectCollection and WSRowObject (both individually and in arrays) from it.

To do this it is generally necessary to know the table name or category.

The table names used are the internal table names. These are not generally used in the software's user interface but can be seen in the exported CSV files. They are typically of the form prefix_name or prefix_parts_of_name (i.e. lower-case strings, beginning with a prefix and with words separated by underscores). The prefixes are as follows:

model networks (ICM Exchange) – hw

collection networks (InfoAsset Exchange) – cams

distribution networks (InfoAsset Exchange) – wams

WS Pro model networks (WS Pro Exchange) – wn

WS Pro controls (WS Pro Exchange) – wn_ctl

e.g. hw_node, cams_manhole, wams_manhole, wn_node, wn_ctl_node

Categories are used to obtain the objects in more than one table; the most common use of a category is to obtain all of the nodes or links in a network regardless of the types of the individual nodes or links

The categories are as follows:

'_nodes' – all nodes

'_links' – all links

'_subcatchments' – all subcatchments (model networks only)

'_other' – other objects

The lists of tables for the types of network are as follows:

Model Network Tables

2D boundary	hw_2d_boundary_line
2D point source	hw_2d_point_source
2D zone	hw_2d_zone
2D zone defaults	hw_2d_zone_defaults
Bank line	hw_bank_survey
Base linear structure (2D)	hw_2d_linear_structure
Bridge	hw_bridge
Bridge inlet	hw_bridge_inlet
Bridge opening	hw_bridge_opening
Bridge outlet	hw_bridge_outlet
Channel	hw_channel
Channel defaults	hw_channel_defaults
Channel shape	hw_channel_shape
Conduit	hw_conduit
Conduit defaults	hw_conduit_defaults
Cross section line	hw_cross_section_survey
Culvert inlet	hw_culvert_inlet
Culvert outlet	hw_culvert_outlet
Flap valve	hw_flap_valve
Flow efficiency	hw_flow_efficiency
Flume	hw_flume

General line	hw_general_line
General point	hw_general_point
Ground infiltration	hw_ground_infiltration
Head discharge	hw_head_discharge
Headloss curve	hw_headloss
IC zone - hydraulics (2D)	hw_2d_ic_polygon
IC zone - infiltration (2D)	hw_2d_inf_ic_polygon
IC zone - water quality (2D)	hw_2d_wq_ic_polygon
Infiltration surface (2D)	hw_2d_infil_surface
Infiltration zone (2D)	hw_2d_infiltration_zone
Inline bank	hw_inline_bank
Irregular weir	hw_irregular_weir
Land use	hw_land_use
Large catchment parameters	hw_large_catchment_parameters
Mesh zone	hw_mesh_zone
Network results line (2D)	hw_2d_results_line
Network results point (1D)	hw_1d_results_point
Network results point (2D)	hw_2d_results_point
Network results polygon (2D)	hw_2d_results_polygon
Node	hw_node
Node defaults	hw_manhole_defaults
Orifice	hw_orifice
Polygon	hw_polygon
Porous polygon	hw_porous_polygon
Porous wall	hw_porous_wall
Pump	hw_pump
RTC data	hw_rtc
RTK hydrograph	hw_unit_hydrograph
River defaults	hw_river_reach_defaults

River reach	hw_river_reach
Roughness zone	hw_roughness_zone
Runoff surface	hw_runoff_surface
Screen	hw_screen
Shape	hw_shape
Sim parameters	hw_sim_parameters
Siphon	hw_siphon
Sluice	hw_sluice
Sluice linear structure (2D)	hw_2d_sluice
Snow pack	hw_snow_pack
Snow parameters	hw_snow_parameters
Storage area	hw_storage_area
Subcatchment	hw_subcatchment
Subcatchment defaults	hw_subcatchment_defaults
User control	hw_user_control
Water quality parameters	hw_wq_params
Weir	hw_weir

Collection Network Tables

Approval level	cams_approval_level
Blockage incident	cams_incident_blockage
CCTV survey	cams_cctv_survey
Channel	cams_channel
Collapse incident	cams_incident_collapse
Connection node	cams_connection_node
Connection pipe	cams_connection_pipe
Connection pipe name group	cams_name_group_connection_pipe
Cross section survey	cams_cross_section_survey
Customer complaint	cams_incident_complaint

Data logger	cams_data_logger
Defence area	cams_defence_area
Defence structure	cams_defence_structure
Drain test	cams_drain_test
Dye test	cams_dye_test
FOG inspection	cams_fog_inspection
Flooding incident	cams_incident_flooding
Flume	cams_flume
GPS survey	cams_gps_survey
General asset	cams_general_asset
General incident	cams_incident_general
General line	cams_general_line
General maintenance	cams_general_maintenance
General survey	cams_general_survey
General survey line	cams_general_survey_line
Generator	cams_generator
Manhole repair	cams_manhole_repair
Manhole survey	cams_manhole_survey
Material	cams_material
Monitoring survey	cams_mon_survey
Node	cams_manhole
Node name group	cams_name_group_node
Odor incident	cams_incident_odor
Order	cams_order
Orifice	cams_orifice
Outlet	cams_outlet
Pipe	cams_pipe
Pipe clean	cams_pipe_clean
Pipe name group	cams_name_group_pipe

Pipe repair	cams_pipe_repair
Pollution incident	cams_incident_pollution
Property	cams_property
Pump	cams_pump
Pump station	cams_pump_station
Pump station electrical maintenance	cams_pump_station_em
Pump station mechanical maintenance	cams_pump_station_mm
Pump station survey	cams_pump_station_survey
Resource	cams_resource
Screen	cams_screen
Siphon	cams_siphon
Sluice	cams_sluice
Smoke defect observation	cams_smoke_defect
Smoke test	cams_smoke_test
Storage area	cams_storage
Treatment works	cams_wtw
User ancillary	cams_ancillary
Valve	cams_valve
Vortex	cams_vortex
Weir	cams_weir
Zone	cams_zone

Distribution Network Tables

Approval level	wams_approval_level
Borehole	wams_borehole
Burst incident	wams_incident_burst
Customer complaint	wams_incident_complaint
Data logger	wams_data_logger
Fitting	wams_fitting

GPS survey	wams_gps_survey
General asset	wams_general_asset
General incident	wams_incident_general
General line	wams_general_line
General maintenance	wams_general_maintenance
General survey	wams_general_survey
General survey line	wams_general_survey_line
Generator	wams_generator
Hydrant	wams_hydrant
Hydrant maintenance	wams_hydrant_maintenance
Hydrant test	wams_hydrant_test
Leak detection	wams_leak_detection
Manhole	wams_manhole
Manhole repair	wams_manhole_repair
Manhole survey	wams_manhole_survey
Material	wams_material
Meter	wams_meter
Meter maintenance	wams_meter_maintenance
Meter test	wams_meter_test
Monitoring survey	wams_mon_survey
Node name group	wams_name_group_node
Order	wams_order
Pipe	wams_pipe
Pipe name group	wams_name_group_pipe
Pipe repair	wams_pipe_repair
Pipe sample	wams_pipe_sample
Property	wams_property
Pump	wams_pump
Pump station	wams_pump_station

Pump station electrical maintenance	wams_pump_station_em
Pump station mechanical maintenance	wams_pump_station_mm
Pump station survey	wams_pump_station_survey
Resource	wams_resource
Surface source	wams_surface_source
Tank	wams_tank
Treatment works	wams_wtw
Valve	wams_valve
Valve maintenance	wams_valve_maintenance
Water quality incident	wams_incident_wq
Zone	wams_zone

WS Pro Network Tables

Customer Point	wn_address_point
Customer Point Allocation	wn_address_point_allocation
Polygon Category	wn_category
Channel Shape	wn_channel_shape
Area Group	wn_componentset_area
Node Group	wn_componentset_node
Link Group	wn_componentset_pipe
Pump Station Group	wn_componentset_pump
Reservoir Group	wn_componentset_res
Demand Area	wn_demand_area
DigitiseValues	wn_DigitiseValues
Fixed Head	wn_fixed_head
Float Valve/Inlet	wn_float_valve
Hydrant	wn_hydrant
Spatial Data	wn_incident_report
Land Use	wn_landuse
Meter	wn_meter
Meter Construction	wn_meter_construction
Node	wn_node
Non Return Valve	wn_non_return_valve
Open Channel	wn_open_channel
Pipe	wn_pipe
Pipe Material	wn_pipe_material

Polygons	wn_polygon
Node Polygons	wn_polygon_node
PRD Curve	wn_prd_curve
Pruned Link	wn_prunes
Pump Station	wn_pst
Pump	wn_pump
Reservoir	wn_reservoir
Transfer Node	wn_transfer_node
Valve	wn_valve
Valve Curve	wn_valve_curve
Well	wn_well

e.g. to obtain all the nodes as a WSRowObjectCollection object

```
roc=on.row_object_collection('_nodes')
```

To obtain them as an array of WSRowObject objects

```
ro_vec=on.row_objects('_nodes')
```

In both cases the resultant WSRowObjectCollection or array can be iterated through e.g. this example, which finds the smallest x coordinate of all the manholes in the network.

```
net=WSApplication.current_network
minx=nil
net.row_object_collection('cams_manhole').each do |ro|
      if minx.nil? || ro.x < minx
            minx=ro.x
      end
end
puts minx
```

or the more or less identical

```
net=WSApplication.current_network
minx=nil
net.row_objects('cams_manhole').each do |ro|
      if minx.nil? || ro.x < minx
            minx=ro.x
      end
end
puts minx
```

In this instance, the fact that one method returns a Ruby array of WSRowObjects and the other returns a WSRowObjectCollection object is disguised by the

WSRowObjectCollection object following the normal Ruby convention for enumerable objects.

To obtain an individual WSRowObject from a WSOpenNetwork if you know its name use the row_object method e.g.

```
ro=on.row_object('cams_manhole','MH359457')
```

This uses the ID of an object, so for links in model networks use the upstream node ID followed by a dot followed by the links suffix e.g. the following example which clears the selection, and selects one link:

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_object('hw_conduit','44633101.1')
ro.selected=true
```

This example uses the links category describe above:

```
net=WSApplication.current_network
net.clear_selection
net.row_objects('_links').each do |ro|
      ro.selected=true
end
```

Because names are unique within the node, link and subcatchment categories, you can also use the category to find individual objects, thus the example which selects one link above can be rewritten as:

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_object('_links','44633101.1')
ro.selected=true
```

Nodes and links are always automatically returned as objects of classes WSNode and WSLink respectively, with the parent class WSRowObject. These are largely identical to the WSRowObject class but have a few extra methods for use when navigating between objects (see below).

Getting and setting values in 'row objects'.

A fundamental part of the purpose of the Ruby scripting within the products is to get and set values of fields for the objects within the networks.

There are two fundamentally different sorts of values that can be got or set and two fundamentally different ways of doing this.

The two sorts of values are:

1. Values of object fields stored in the network. These are the values with which users of the software will be familiar; they are the values that are displayed in the grids and property sheets, imported via the Open Data Import Centre, exported via the Open Data Export Centre etc.
   The names of the object fields are fixed for each type of object.
2. Tags. These are temporary values added to the objects for the duration of the running of the script. They are typically used for storing 'working' values which will later be aggregated or stored into the object fields which will persist after the running of the script– they can, of course, also be written to files using Ruby's file access mechanisms.
   The names of tags are not fixed but must begin with _ and can only contain digits and the letters A-Z and a-z (without accents) and the _ character.

The two ways of accessing values are:

1. By using the [] and []= notation e.g. ro['field']=value, value=ro['field'] (for object fields) and ro['_tag_name']=value, value=ro['_tag_name'] (for tags)
2. By using the field name or tag name as though it were the name of a method e.g. ro.field=value, value=ro.field (for object fields) and ro._tag_name = value, value=ro._tag_name (for tags).

The key differences in behaviour between object fields and values, beyond that of the object field values having a life beyond the duration of the running of the script, are

1. Object field values must be explicitly written back to the local database for the network using the write method – since tags are not stored anywhere other than in working memory, the write method does not need to be called for them.
2. Object field values can only be stored within an active 'transaction' (see below).
3. Object field values are stored in the 'InfoWorks / InfoAsset' world. Any given field has a particular data type and, for string fields, a length. Any attempt to store values incompatible with the object's data type will fail. Tags, on the other hand, exist in the Ruby world and may therefore contain anything that can be stored in a Ruby variable. There is no requirement for all the values for different objects of the same tag to be of the same data type.
4. Object field values may be cached in the database, allowing more objects and more data to be manipulated within a network than with tags, which always exist in memory. Using too many tags and storing too much data in them may cause the program's memory limit to be exceeded.

Flags are treated as being separate fields.

Fields can, in general, be set to nil which is the equivalent of causing them to be blank in the user interface or setting them to NULL in SQL. NULL in SQL and nil in Ruby are essentially the same.

Arrays e.g. of coordinates are returned as a Ruby array.

This example finds and selects pipes with width less than 200 or length less than 60 or, of course, both.

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_objects('cams_pipe').each do |ro|
      if (!ro.width.nil? && ro.width<200) || (!ro.length.nil? &&
ro.length<60)
            ro.selected=true
      end
end
```

This demonstrates a key difference between Ruby and SQL; in SQL it is safe to say

width<200, the expression will ignore values which are NULL. In Ruby however, it is necessary to explicitly check for nil values, nil being the Ruby counterpart to NULL.

If you fail to do this check a runtime error will be raised.

An equivalent way of writing the same script would be to use the [] notation as follows:

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_objects('cams_pipe').each do |ro|
      if (!ro['width'].nil? && ro['width']<200) ||
(!ro['length'].nil? && ro['length']<60)
            ro.selected=true
      end
end
```

In the rare cases where the field name begins with a digit or the _ character it is necessary to use the ro['fieldname'] form to access the value.

To set values it is necessary to

a) Set them within a transaction. Transactions are treated as a single unit for purposes of undo / redo. When run from the user interface, each transaction is treated as a single undo / redo step and appears in the menu as 'Scripted transaction'.
b) Call the write method on the row object to explicitly put the values into the database. This is the equivalent in the user interface of finishing to edit an object, of which you might have changed a number of values.

This example sets a couple of users fields for CCTV surveys based on simple calculations performed on other fields:

```
net=WSApplication.current_network
net.clear_selection
net.transaction_begin
ro=net.row_objects('cams_cctv_survey').each do |ro|
      ro.user_number_1 = ro.surveyed_length / ro.total_length
      ro.user_number_2 = ro.total_length / ro.pipe_length
      ro.write
end
net.transaction_commit
```

The equivalent way of writing the script using the [] notation is as follows:

```
net=WSApplication.current_network
net.clear_selection
net.transaction_begin
ro=net.row_objects('cams_cctv_survey').each do |ro|
      ro['user_number_1'] = ro['surveyed_length'] /
ro['total_length']
      ro['user_number_2'] = ro['total_length'] / ro['pipe_length']
      ro.write
end
net.transaction_commit
```

The use of the form which looks as though it is a method e.g. ro.user_number_1 is potentially clearer to those writing and maintaining scripts, but the [] form can be more flexible since the parameter of the [] method is a Ruby string and therefore can be an expression. The following demonstrates this by storing the two values used on the right-hand side of the above expressions as string parameters, and building up the user field name as a string expression:

```
net=WSApplication.current_network
net.clear_selection
net.transaction_begin
expressions=[['surveyed_length','total_length'],['total_length','pip
e_length']]
ro=net.row_objects('cams_cctv_survey').each do |ro|
      (0...expressions.size).each do |i|
ro['user_number_'+(i+1).to_s] = ro[expressions[i][0]] /
ro[expressions[i][1]]
            ro.write
      end
end
net.transaction_commit
```

Once the user has run a script such as the above, the changes will have been made to the local network as though the change had been made manually in the user interface, or via SQL or similar, the changes have NOT been committed to the master database. It IS possible to commit the network to the master database by adding a call to the commit method with a suitable comment as a parameter e.g.

```
net.commit 'set user fields'
```

Two users of tags, one simple and one more complex, are demonstrated below in the 'navigating between objects' section.

Various data fields in InfoWorks and InfoAsset are represented as 'structure blobs' - the field contains a number of 'rows' of values for each object which in some respects behave as though they are a sub-table – they have a number of named fields with values.

The structure blobs that are most common are the following:

- hyperlinks
- attachments
- material_details
- resource_details

Many tables contain a hyperlinks field. The following tables in asset networks contain one or more of the other three fields named above:

Collection Network

Table	has attachments field	has material_details field	has resource_details field
Blockage incident	Y	Y	Y
CCTV survey	Y	Y	Y
Channel	Y	N	N
Collapse incident	Y	Y	Y
Connection node	Y	N	N
Connection pipe	Y	N	N
Cross section survey	Y	Y	Y
Customer complaint	Y	Y	Y
Data logger	Y	N	N
Defence area	Y	N	N

Defence structure	Y	N	N
Drain test	Y	Y	Y
Dye test	Y	Y	Y
FOG inspection	Y	Y	Y
Flooding incident	Y	Y	Y
Flume	Y	N	N
GPS survey	Y	Y	Y
General asset	Y	N	N
General incident	Y	Y	Y
General line	Y	N	N
General maintenance	Y	Y	Y
General survey	Y	Y	Y
General survey line	Y	Y	Y
Generator	Y	N	N
Manhole repair	Y	Y	Y
Manhole survey	Y	Y	Y
Monitoring survey	Y	Y	Y
Node	Y	N	N
Odor incident	Y	Y	Y
Orifice	Y	N	N
Outlet	Y	N	N
Pipe	Y	N	N
Pipe clean	Y	Y	Y
Pipe repair	Y	Y	Y
Pollution incident	Y	Y	Y
Property	Y	N	N
Pump	Y	N	N
Pump station	Y	N	N

Pump station electrical maintenance	Y	Y	Y
Pump station mechanical maintenance	Y	Y	Y
Pump station survey	Y	Y	Y
Screen	Y	N	N
Siphon	Y	N	N
Sluice	Y	N	N
Smoke defect observation	Y	N	N
Smoke test	Y	Y	Y
Storage area	Y	N	N
Treatment works	Y	N	N
User ancillary	Y	N	N
Valve	Y	N	N
Vortex	Y	N	N
Weir	Y	N	N
Zone	Y	N	N

Distribution Network

Table	has attachments field	has material_details field	has resource_details field
Borehole	Y	N	N
Burst incident	Y	Y	Y
Customer complaint	Y	Y	Y
Data logger	Y	N	N
Fitting	Y	N	N
GPS survey	Y	Y	Y
General asset	Y	N	N

General incident	Y	Y	Y
General line	Y	N	N
General maintenance	Y	Y	Y
General survey	Y	Y	Y
General survey line	Y	Y	Y
Generator	Y	N	N
Hydrant	Y	N	N
Hydrant maintenance	Y	Y	Y
Hydrant test	Y	Y	Y
Leak detection	Y	Y	Y
Manhole	Y	N	N
Manhole repair	Y	Y	Y
Manhole survey	Y	Y	Y
Meter	Y	N	N
Meter maintenance	Y	Y	Y
Meter test	Y	Y	Y
Monitoring survey	Y	Y	Y
Pipe	Y	N	N
Pipe repair	Y	Y	Y
Pipe sample	Y	Y	Y
Property	Y	N	N
Pump	Y	N	N
Pump station	Y	N	N
Pump station electrical maintenance	Y	Y	Y
Pump station mechanical maintenance	Y	Y	Y
Pump station survey	Y	Y	Y
Surface source	Y	N	N

Tank	Y	N	N
Treatment works	Y	N	N
Valve	Y	N	N
Valve maintenance	Y	Y	Y
Valve shut off	Y	N	N
Water quality incident	Y	Y	Y
Zone	Y	N	N

In addition to these four fields, the following fields containing structure blobs occur in the tables as follows:

Model Network

Bank line

Bank line	Bank data	bank_array

Base linear structure (2D)

Base linear structure (2D)	Section data	sections

Bridge

Bridge	Bridge deck data	bridge_deck
Bridge	DS bridge section data	ds_bridge_section
Bridge	DS link section data	ds_link_section
Bridge	US bridge section data	us_bridge_section
Bridge	US link section data	us_link_section

Bridge linear structure (2D)

Bridge linear structure (2D)	Section data	sections

Bridge opening

Bridge opening	Piers	piers

Channel shape

Channel shape	Channel profile	profile

Cross section line

Cross section line	Section data	section_array

Flow efficiency

Flow efficiency	Flow efficiency table	FE_table

Head discharge

Head discharge	Head discharge power table	HDP_table

Inline bank

Inline bank	Section data	bank

Irregular weir

Irregular weir	Chainage elevation	chainage_elevation

Node

Node	Storage array	storage_array

River reach

River reach	Left river bank	left_bank
River reach	Right river bank	right_bank
River reach	River sections	sections
River reach	Section spacing	section_spacing

Shape

Shape	Geometry	geometry

Subcatchment

Subcatchment	ReFH descriptors	refh_descriptors

Collection Network

CCTV survey

CCTV survey	Details	details

Cross section survey

Cross section survey	Section data	section_data

General survey line

General survey line	Points	point_array

Manhole survey

Manhole survey	Details	details
Manhole survey	Incoming pipes	pipes_in
Manhole survey	Outgoing pipes	pipes_out

Order

Order	Order details	order_details

Pipe clean

Pipe clean	Pipes	pipes

Pump station

Pump station	Available telemetry	available_telemetry
Pump station	Levels	levels
Pump station	Pump groups	pump_groups

Pump station mechanical maintenance

Pump station mechanical maintenance	Pumps	pumps

Pump station survey

Pump station survey	Drop tests	drop_tests

Storage area

Storage area	Level Data	level_data

Distribution Network

General survey line

General survey line	Points	point_array

Hydrant test

Hydrant test	Flow hydrants	flow_hydrants

Order

Order	Order details	order_details

Pump station

Pump station	Available telemetry	available_telemetry
Pump station	Levels	levels
Pump station	Pump groups	pump_groups

Pump station mechanical maintenance

Pump station mechanical maintenance	Pumps	pumps

Pump station survey

Pump station survey	Drop tests	drop_tests

Valve shut off

Valve shut off	Points	point_array
Valve shut off	Valves	valve_details

WS Pro Network

Node

Node	Demand By Category	demand_by_category
Node	Land Use Areas	landuse_areas

Reservoir

Reservoir	Demand By Category	demand_by_category
Reservoir	Depth Volume	depth_volume
Reservoir	Land Use Areas	landuse_areas

Hydrant

Hydrant	Demand By Category	demand_by_category
Hydrant	Land Use Areas	landuse_areas

Pump

Pump	Triplets	triplets
Pump Station		
Pump Station	Pumps	pumps
Valve Curve		
Valve Curve	Loss Coefficients	curve
PRD Curve		
PRD Curve	Curve Values	curve
Link Group		
Link Group	Link ID	pipe_links
Pump Station Group		
Pump Station Group	Link ID	pump_links
Node Group		
Node Group	Node ID	nodes
Reservoir Group		
Reservoir Group	Node ID	res_nodes
Area Group		
Area Group	Boundary links	boundary_links
Area Group	Node ID	area_nodes

A simple example which loops through all the CCTV details to build up a list of videos used is as follows:

```
net=WSApplication.current_network
videos=Hash.new
ro=net.row_objects('cams_cctv_survey').each do |ro|
      ro.details.each do |d|
            video=d.video_no
            if !video.nil?
                  if !videos.has_key?(video)
                        videos[video]=0
                  end
            end
      end
end
videos.keys.sort.each do |k|
```

```
      puts k
end
```

ro.details may be written as ro['details'] as with all other fields. ro.details in this case is an object of type WSStructure. Each row of the structure is accessed as an object of type WSStructureRow, a class which has only two methods, [] and []=.

An alternative way of writing the code is to get the rows by index rather than use the 'each' method:

```
net=WSApplication.current_network
net.transaction_begin
net.row_objects('cams_cctv_survey').each do |ro|
      ro_details=ro.details
      (0...ro_details.size).each do |i|
            detail_row=ro_details[i]
            if detail_row.code=='OJS'
                  detail_row.code='SJO'
            end
      end
      ro_details.write
      ro.write
end
net.transaction_commit
```

This version makes it more explicit that ro.details and the individual rows are Ruby objects.

When setting values in structure blobs it is necessary to call the write method on the WSStructure to save the data back to the WSRowObject, the write method must then be called on the WSRowObject to save it back to the local database e.g.

```
net=WSApplication.current_network
net.transaction_begin
net.row_objects('cams_cctv_survey').each do |ro|
      ro.details.each do |d|
            if d.code=='OJS'
                  d.code='SJO'
            end
      end
      ro.details.write
      ro.write
end
net.transaction_commit
```

This example changes the OJS code to SJO in all defects in all CCTV surveys.

Navigating between objects

The term 'navigate' is used here for the process of finding objects that are either physically connected to a given object (e.g. the upstream node, the downstream links) or conceptually linked (e.g. the surveys for an asset, the assets for a survey).

There are two methods for navigating between objects.

Specific to nodes and links.

Node and links are presented to the user as instances of classes WSNode and WSLink respectively. The nodes have the methods us_links and ds_links and the links have methods us_node and us_link.

This code clears the selection, then selects a node, then iteratively selects its upstream links, then their upstream nodes, then their upstream links etc.

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_object('cams_manhole','MH354671')
ro.selected=true
ro._seen=true
unprocessedLinks=Array.new
ro.us_links.each do |l|
      if !l._seen
            unprocessedLinks << l
            l._seen=true
      end
end
while unprocessedLinks.size>0

      working=unprocessedLinks.shift
      working.selected=true
      workingUSNode=working.us_node
      if !workingUSNode.nil? && !workingUSNode._seen
            workingUSNode.selected=true
            workingUSNode.us_links.each do |l|
                  if !l._seen
                        unprocessedLinks << l
                        l.selected=true
                        l._seen=true
                  end
            end
      end
end
```

As well as demonstrating use of the us_links method of WSNode and the us_node method of WSLink, this demonstrate some other useful techniques:

1 – as with the example above listing the WSModelObject objects in a database, this demonstrates the use of a breadth first search – we add the upstream links of the node to an array, then work through the array from the front, taking the links from it, selecting them,

then if they have an upstream node, getting the upstream links of that node and adding them to the back of the array. In this case we are using the shift method of the Ruby array, which returns the first item in the array, removing it from the array.

2-unlike the navigation of the database, where the objects are in a simple tree structure, networks can contain loops, therefore you will typically need to make sure that you only process any given node or link once, otherwise your script may well keep revisiting the same objects over and over again. We do this by use of a tag which we have named '_seen'. Whenever we process a node or link we set the value of the _seen tag to true, and we ensure that we don't process nodes or links if they have got the tag set to true, signifying that they have already been processed.

General

The more general way of navigating between objects is to use the navigate and navigate1 methods of the WSRowObject. The difference between the 2 methods is that navigate1 may only be used for one to one links and returns a WSRowObject or nil, whereas navigate may also be used for one-to-many links and returns an array, possibly containing zero elements.

The code above may be rewritten using these methods as follows:

```
net=WSApplication.current_network
net.clear_selection
ro=net.row_object('cams_manhole','MH354671')
ro.selected=true
ro._seen=true
unprocessedLinks=Array.new
ro.navigate('us_links').each do |l|
      if !l._seen

            unprocessedLinks << l
            l._seen=true
      end
end
while unprocessedLinks.size>0
      working=unprocessedLinks.shift
      working.selected=true
      workingUSNode=working.navigate1('us_node')
      if !workingUSNode.nil? && !workingUSNode._seen
            workingUSNode.selected=true
            workingUSNode.navigate('us_links').each do |l|
                  if !l._seen
                        unprocessedLinks << l
                        l.selected=true
                        l._seen=true
                  end
            end
      end
end
```

As you can see, the only changes here are that calls to us_links are replaced by calls to nagivate('us_links') and the call to us_link is replaced by a call to navigate1('us_link').

The navigate method however is much more versatile – this example navigates from CCTV surveys to pipes

```
net=WSApplication.current_network
interesting_codes=['ABC','DEF','GHI','JKL','MNO']
net.transaction_begin
net.row_objects('cams_pipe').each do |ro|
      (0...interesting_codes.size).each do |i|
            ro['user_number_'+(i+1).to_s]=nil
      end
      ro.write
end
codes=Hash.new
net.row_objects('cams_cctv_survey').each do |ro|
      ro.details.each do |d|
            code=d.code
            code_index=interesting_codes.index(code)
            if !code_index.nil?
                  pipe=ro.navigate1('pipe')
                  if pipe
                        if pipe._defects.nil?

      pipe._defects=Array.new(interesting_codes.size,0)
                        end
                        pipe._defects[code_index]+=1
                  end
            end
      end
end
net.row_objects('cams_pipe').each do |ro|
      if !ro._defects.nil?
            (0...interesting_codes.size).each do |i|
                  ro['user_number_'+(i+1).to_s]=ro._defects[i]
            end
            ro.write
      end
end
net.transaction_commit
```

Essentially, it clears user numbers 1 to 5 for all pipes, then iterates through all defects, counting the number of defects of 5 particular codes for each pipe, then stores those in user numbers 1 to 5.

Note the use of arrays stored in tags for temporary storage of counts.

Method Reference

WSApplication

add_ons_folder (ICM / InfoAsset Desktop UI only)

s=WSApplication.add_ons_folder

Returns the full path of the 'add ons' folder described in Appendix 5 e.g. C:\Users\badgerb\AppData\Roaming\Innovyze\WorkgroupClient\scripts

Note that the folder will not exist unless manually created. Its parent folder will almost certainly exist.

background_network (ICM / InfoAsset Desktop UI only)

bn=WSApplication.background_network

Returns the background network for the GeoPlan that currently has focus. Scripts may only work on current and background networks in the UI mode. The background network may, of course, be nil if no background network is loaded.

cancel_job (ICM Exchange only)

WSApplication.cancel_job(job)

Cancels a job being run by the agent. The parameter is a job ID from the array returned by launch_sims – see below.

choose_selection (UI only)

sel=WSApplication.choose_selection('prompt text')

Displays a dialog allowing the choice of a selection list object in the current master database, returns a WSModelObject representing that selection list if the user chooses one and hits OK, returns nil otherwise.

colour (UI only)

col=WSApplication.colour(r,g,b)

This method exists for convenience in the context of the graph method described below. Given 3 parameters, r, g and b from 0 to 255 returns a colour with those red, green and blue values e.g. colour(0,0,0) returns black, colour(255,0,0) returns red, colour(255,255,255) returns white.

connect_local_agent (ICM Exchange only)

bSuccess=WSApplication.connect_local_agent(wait_time)

Connects to the local agent, returning true if it succeeds. The call waits for a number of milliseconds specified in the parameter.

This method must be called before the launch_sims method is called.

current_database (UI only)

db=WSApplication.current_database

Returns the current database. Only very limited database functionality is available from Ruby scripting from within the UI. This method is used to return the database object for the currently open master database.

current_network (UI only)

on=WSApplication.current_network

Returns the network for the GeoPlan that currently has focus. Scripts may only work on the current network (and background networks) in the UI mode.

If a Ruby script is run in the UI when the current network has results then the results will be available to the script.

create (Exchange only)

WSApplication.create(path)

e.g.

```
WSApplication.create 'd:\\temp\\1.icmm'
```

Creates a standalone database at the location given by the path parameter.

Likely exceptions for this method to throw are the following RuntimeExceptions:

```
Error 43 : Can't overwrite an existing database
z:\test.icmm contains an incorrect path
```

create_transportable (Exchange only)

WSApplication.create_transportable(path)

Creates a transportable database at the location given by the path parameter.

file_dialog (UI only)

file_or_files=WSApplication.file_dialog(open, extension, description, default_name,allow_multiple_files,hard_wire_cancel)

Displays a file dialog (open or save), and if OK is selected returns the file path, or if allow_multiple_files was set to true, an array of selected files.

The parameters are as follows:

open – true if the dialog is to be an 'open' dialog (i.e. to select an existing file to be read subsequently in the Ruby script), false if it is to be a 'save' dialog (i.e. to select the name of a file to be written subsequently in the Ruby script)

extension – the extension of the file e.g. 'csv', 'dat', 'xml'.

Description – a description of the type of file to save which will appear in the file dialog e.g. 'Comma Separated Value file'.

Default name – a default name to save the file (if open is false), or a default name to search for (if open is true).

Allow multiple files – true if open is false and you wish to allow more than one file to be selected. The parameter is ignored if open is true.

The method returns the path of the file chosen as a string, unless open is true and allow_multiple_files is true, in which case an array of strings is returned.

Hard-wire cancel – if this parameter is true or nil, if the user cancels from the dialog, the execution of the Ruby script is stopped.

folder_dialog (UI only)

folder=WSApplication.folder_dialog(title,hard_wire_cancel)

Displays a dialog allowing the selection of a folder, returned as a string.

The parameters are as follows:

Title – a title for the dialog

Hard-wire cancel – if the parameter is true or nil, if the user cancels from the dialog, the execution of the Ruby script is stopped.

If the user selects OK, the path of the folder is returned as a string. If the user selects cancel and the hard_wire_cancel parameter is set to false, nil will be returned.

graph (ICM / InfoAsset Desktop UI only)

WSApplication.graph(params)

Displays a graph according to the parameters passed in.

The graph method contains 1 parameter, a hash.

It has the following keys, which are all strings:

WindowTitle – a string containing the title of the graph window

GraphTitle – a string containing the title of the graph (i.e. this appears IN the window rather than being the title)

XAxisLabel – a string containing the label of the X-axis

YAxisLabel – a string containing the label of the Y-axis

IsTime – a value which evaluates as true or false, which should be set to true if the x axis is made up of time values and is labelled as dates / times.

Traces – an array of traces defined as follows:

Each trace in the array of traces is in turn also a hash.

The trace hash has the following keys, which are all strings:

Title – a string giving the trace's name

TraceColour – an integer containing an RGB value of the trace's colour. A convenient way of getting this is to use the WSApplicatioon.colour method

SymbolColour - – an integer containing an RGB value of the colour used for the symbol used at the points along the trace. A convenient way of getting this is to use the WSApplicatioon.colour method

Marker – a string containing the symbol to be used for the points along the trace – possible values are:

None, Cross, XCross, Star, Circle, Triangle, Diamond, Square, FCircle, FTriangle, FDiamond, FSquare

The F in the above names means 'filled'.

LineType – a string containing the style to be used for the trace's line – possible values are:

None, Solid, Dash, Dot, DashDot, DashDotDot

XArray – an array containing the values used in the trace for the x coordinates of the points. They must be floating point values (or values that can be converted to a floating point values) if IsTime is false or time values if IsTime is true.

YArray - – an array containing the values used in the trace for the x coordinates of the points. They must be floating point values (or values that can be converted to a floating point values).

There must be an equal number of values in the XArray and YArray in each trace, though they can vary between traces.

input_box(UI only)

s=WSApplication.input_box(prompt,title,default)

Displays a box allowing the input of a text value. The prompt appears on the dialog, the title appears as the dialog title, unless it is nil or the empty string in which case a default title appears. The text field is initially set to the 'default' parameter. If the user hits OK, the value in the text field is returned, otherwise the empty string is returned.

launch_sims (ICM Exchange only)

arr=WSApplication.launch_sims(sims, server, results_on_server, max_threads, after)

Launches one or more simulations in a flexible way. This method requires connect_local_agent to have been called prior to its being called.

The parameters are as follows:

Sims – an array of WSModelObject objects for the simulations

Server – the name of the server to run the simulation on, or '.' for the local machine or '*' for any computer.

Results_on_server – Boolean

Max_threads – the maximum number of threads to use for this simulation (or 0 to allow the simulation agent to choose)

After – the time (as a time_t time) after which the simulation should run, or 0 for 'now'.

The method returns an array of 'job IDs', one for each simulation in the sims array, the ID of a given simulation will be nil if the simulation failed to launch. The job IDs are strings intended for use as parameters to the wait_for_jobs method and the cancel_job method. Any nil values in the array will be safely ignored by the wait_for_jobs method so the results array may be passed into it.

map_component

s=WSApplication.map_component

Gets the map component used by IExchange

map_component= (UI only)

WSApplication.map_component=component_name

Sets the map component used by IExchange – valid values are MapXTreme, ArcObjects, ArcEngine.

message_box (UI only)

text=WSApplication.message_box(text,options,icon,hard_wire_cancel)

Displays a message box.

The parameters are as follows:

Text – the text displayed

Options – must be nil or one of the following strings: 'OK', 'OKCancel', 'YesNo', 'YesNoCancel'. If the parameter is nil, then the OK and Cancel buttons are displayed.

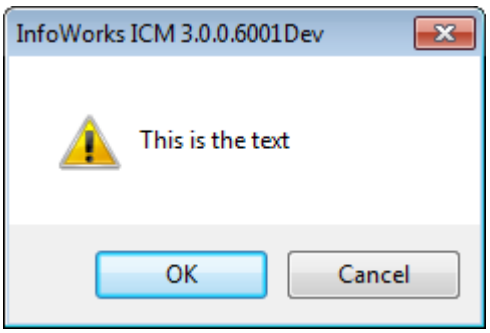
Icon – must be nil or one of the following strings: '!', '?', 'Information'. If the parameter is nil then the '!' icon is used.

Hard Wire Cancel – if this is set to true or nil then hitting the cancel button (if there is one) will result in the Ruby script processing stopping as though the 'cancel' button had been hit.

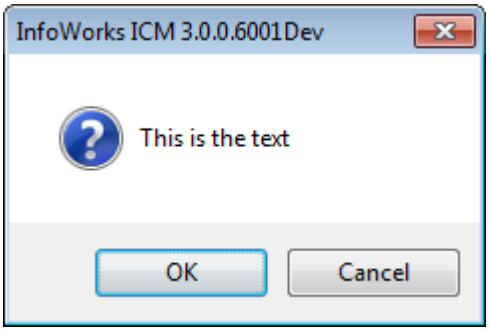
The method returns 'Yes','No','OK' or 'Cancel' as a string, unless cancel is hit and the 'hard wire cancel' parameter is set to true or nil in which case execution of the script is stopped.

The icons appear as follows:

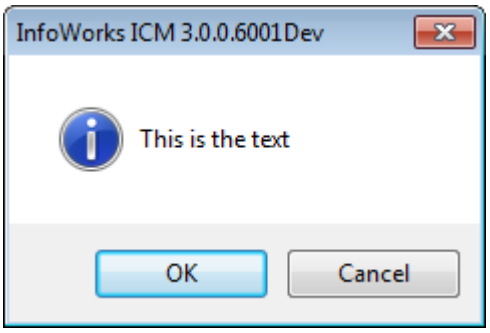
'!' (and nil)



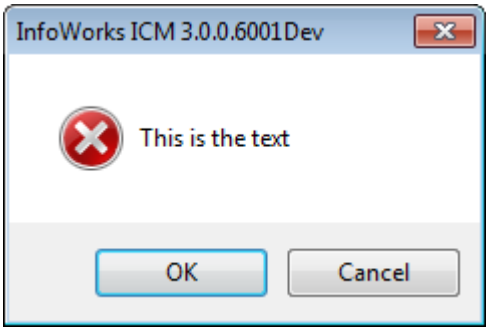
'?'



'Information'

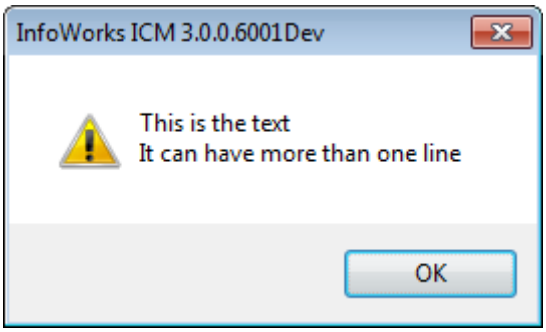


'Stop'

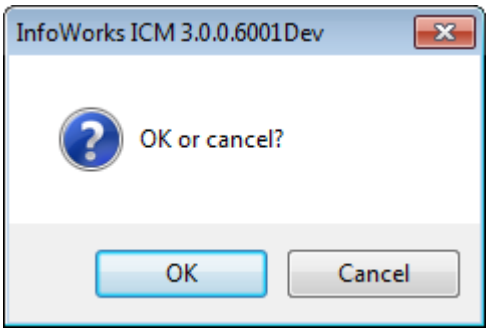


The buttons appear as follows:

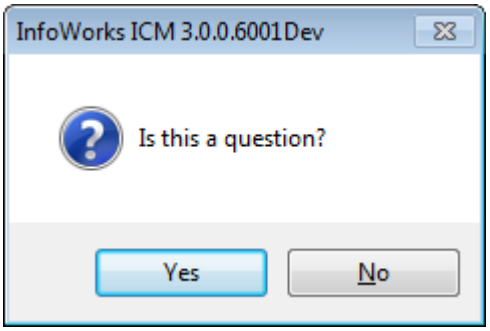
'OK'



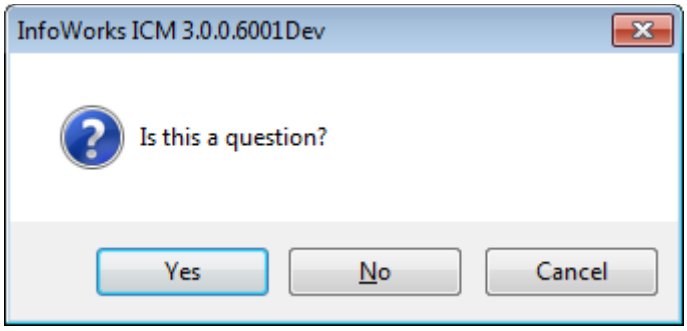
'OKCancel' (and nil)



'YesNo'



'YesNoCancel'



open_text_view (UI only)

WSApplication.open_text_view(title,filename,delete_on_exit)

This method opens a text file in a text view within the application. The parameters are a window title, the filename and a Boolean, which if set to true, will cause the file to be deleted when the view is closed.

It is important to realise that this method does not cause the script to wait until the file is closed – it opens the view and then continues to the next item in the script.

The purpose of the delete_on_exit parameter is allow the user to create a temporary file which will be displayed by this method and then deleted when the view is closed by the user.

prompt (UI only)

arr=WSApplication.prompt(title,layout,hard_wire_cancel)

This method displays a prompt grid, similar to that used in the SQL in the software. It displays values and allows users to edit them.

Title – the title of the displayed dialog.

Layout – an array of arrays as described below.

Hard-wire cancel – if this is set to true or nil, if the user hits cancel, the running of the Ruby script is interrupted.

The method returns an array of values, one for each line in the dialog.

The layout parameter must be an array consisting of one array for each line to be displayed.

The array for each line must contain between 2 and 9 values as follows:

Index 0 – a string to display as a description of the value for the row

Index 1 – a type of the value – one of the following strings:

NUMBER – a number

STRING- a string

DATE – a date (as a Ruby DateTime object)

BOOLEAN – a Boolean value, appearing in the edit column as a check-box

READONLY – a read only value, appearing in the edit column as a string with a grey background. The value is converted to a string except for Ruby types float and double.

Index 2 (optional, except for rows of read only type) – a default value i.e. the initial value when the dialog appears. For read only rows, this is the value displayed which cannot, of course, be changed

Index 3 (optional) a number of decimal places – used for numbers and read only values of Ruby types float and double.

Index 4 (optional) a subtype – one of the following strings

RANGE – valid for NUMBER only, the value will be chosen from a combo box with values specified in array elements index 5 and 6 inclusive.

LIST – valid for NUMBER, STRING and DATE only. The value will be chosen from a combo box of values supplied as an array in array element 5.

MONTH – valid for NUMBER only, the value will be chosen from a combobox containing the names of the months.

FILE – valid for STRING only, the value will be chosen by pressing a button and selecting one or more filenames via a file dialog, the precise details of which will be determined by array elements with indices 5, 6, 7 and 8 as follows:

Index 5 – Boolean – true for an 'open' dialog, false for a 'save' dialog

Index 6 – String – the file extension

Index 7 – String – a description of the file type

Index 8 – Boolean – true to allow the selection of multiple files if index 5 is set to true giving an 'open' dialog, ignored otherwise

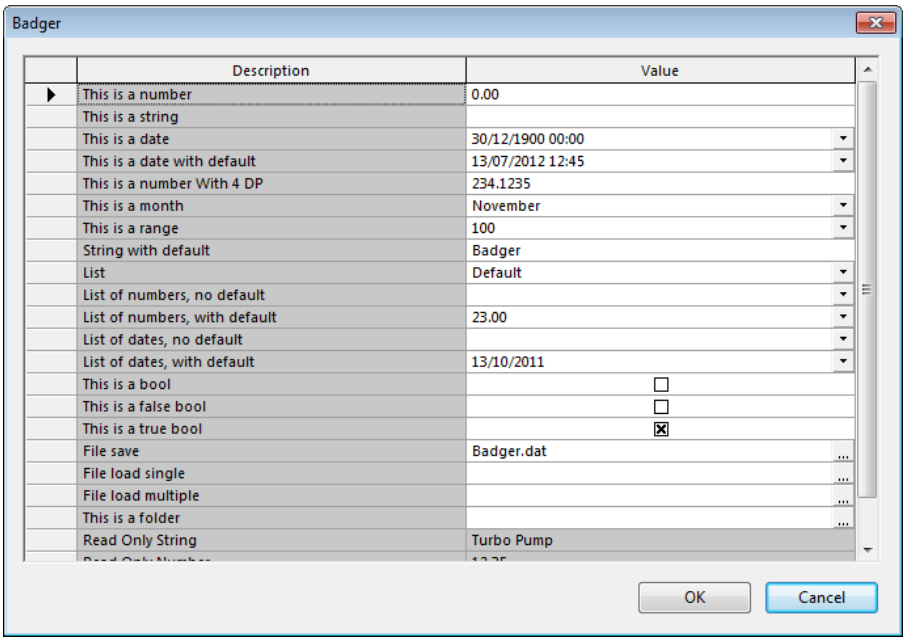
FOLDER – valid for STRING only, the value will be chosen by pressing a button and selecting a folder. Array element 6 must be a folder description as a string or nil, in which case a default title will be used.

Here is an example of a prompt exercising most of these options:

```
require 'Date'
val=WSApplication.prompt "Badger",
[
['This is a number','NUMBER'],
['This is a string','String'],
['This is a date','Date'],
['This is a date with
default','Date',DateTime.new(2012,7,13,12,45,00)],
['This is a number With 4 DP','Number',234.123456,4],
['This is a month','Number',11,nil,'MONTH'],
['This is a range','Number',13,2,'RANGE',100,200],
['String with default','String','Badger'],
['List','String','Default',nil,'LIST',['Alpha','Beta','Gamma']],
['List of numbers, no default','Number',nil,nil,'LIST',[3,5,7,11]],
['List of numbers, with
default','Number',23,nil,'LIST',[13,17,19,23]],
['List of dates, no
default','Date',nil,nil,'LIST',[DateTime.new(2012,7,13,12,45,00),Dat
eTime.new(2012,7,17,13,15,00),DateTime.new(2012,7,10,17,15,00)]],
['List of dates, with
default','Date',DateTime.new(2011,10,13,14,12,30),nil,'LIST',[DateTi
me.new(2012,7,13,12,45,00),DateTime.new(2012,7,17,13,15,00),DateTime
.new(2012,7,10,17,15,00)]],
['This is a bool','Boolean'],
['This is a false bool','Boolean',false],
['This is a true bool','Boolean',true],
['File save','String','Badger.dat',nil,'FILE',false,'dat','Data
file',false],
['File load single','String',nil,nil,'FILE',true,'dat','MySystem
data file',false],
['File load multiple','String',nil,nil,'FILE',true,'dat','More than
one MySystem data file',true],
['This is a folder','String',nil,nil,'FOLDER','Name of MyFolder'],
['Read Only String','Readonly','Turbo Pump'],
['Read Only Number','Readonly',12.3456678],
['Read only number 6 dp','ReadOnly',87.65456789,6]
```

```
],false
puts val.to_s
```

When run, it looks like this:



As you can see, there are a number of lists (with the button for the pull down to the right), and a number of filenames (with a button showing ellipses to press to invoke the file or folder dialog).

The read only values are showed greyed out, in this example the rows are ordered so they appear at the bottom of the grid.

Notice that the date time values in the lists and for defaults are set using DateTime.new as described earlier in this document.

If OK is hit without changing any values, it returns an array like this:

```
[nil, nil, nil, #<DateTime: 2012-07-13T12:45:00+00:00
(78595905/32,0,2299161)>, 234.123456, 11.0, 100.0, "Badger",
"Default", nil, 23.0, nil, #<DateTime: 2011-10-13T14:12:30+00:00
(1414568501/576,0,2299161)>, false, false, true, "Badger.dat", nil,
nil, nil, "Turbo Pump", 12.3456678, 87.65456789]
```

scalars (UI only)

WSApplication.scalars(title,layout,hard_wire_cancel)

This method displays a grid of values, similar to that used in the SQL

The parameters are as follows:

Title – the title of the displayed dialog.

Layout – an array of arrays as described below.

Hard-wire cancel – if this is set to true or nil, if the user hits cancel, the running of the Ruby script is interrupted.

The layout parameter must be an array consisting of one array for each line to be displayed.

The array for each line must contain 2 or 3 values as follows:

Index 0 – a string to display as the description of the value for the row

Index 1 – the value to be displayed as a Ruby value. If the value is a float or a double it will be displayed by using the Ruby to_f method, otherwise the to_s method will be used.

Index 2 (optional) – a number of decimal places to be used between 0 and 8 inclusive, this will be used for float and double values and ignored for any others.

wait_for_jobs (ICM Exchange only)

id_or_nil=WSApplication.wait_for_jobs(array_of_job_ids,wait_for_all,timeout)

Waits for one or all of the jobs in the array to complete, or for the timeout time to be reached.

The parameters are as follows:

Array_of_job_ids – an array of job IDs, i.e. values in the array returned by launch_sims. This array may contain nil values which are safely ignored.

Wait_for_all – true to wait until all the jobs in the array complete, false to wait for one.

Timeout – a timeout time in milliseconds.

Returns Qnil if waittime is exceeded, otherwise returns the array index of the job that caused the wait to end (if waitall is true, this is the last job to complete)

wds_query_databases

hash=WSApplication.wds_query_databases(server,port)

Returns a hash with 3 values: response, databases and allowDatabaseCreation

'response' is the repsonse from the server as a string

'databases' is an array of hashes, with one member per database, see below.

'allowDatabaseCreation' is a Boolean

The hash for each database has 3 keys:

'databaseName' – the database name as a string

'version' – the database version as a string

'versionIsCurrent' – a Booolean.

working_folder (Exchange only)

s=WSApplication.working_folder

e.g.

```
puts WSApplication.working_folder
```

Returns the current working folder as a string

NB – if you have not specified a working folder, either in a script or in the UI, the default working folder which is used will be returned i.e. this returns the working folder that is actually being used regardless of whether you have explicitly set it or not

results_folder (Exchange only)

s=WSApplication.results_folder

e.g.

```
puts WSApplication.results_folder
```

Returns the current results folder as a string

NB – if you have not specified a results folder, either in a script or in the UI, the default results folder which is used will be returned i.e. this returns the results folder that is actually being used regardless of whether you have explicitly set it or not

open (Exchange only)

iwdb=WSApplication.open(path,bUpdate)

iwdb=WSApplication.open(path)

iwdb=WSApplication.open

e.g.

```
db=WSApplication.open 'd:\\temp\\1.icmm',true
db.open 'd:\\temp\\1.icmm'
db.open
```

Opens the database with path 'path' and returns an object of type WSDatabase. If the database requires updating and the flag bUpdate evaluates to true then the database will be updated (if possible) otherwise the call will fail

If the path is blank or the method is called with no parameters then uses the current master database, if any. If bUpdate is omitted it is treated as false.

Likely exceptions for this method to throw are:

```
Error 13 : File Not Found : d:\temp\misc.icmm (error=2: "The system
cannot find the file specified.")
```

if the database is not present.

```
Error 13 : File Not Found : z:\misc.icmm (error=3: "The system
cannot find the path specified.")
```

if the database path is invalid

```
no database path specified
```

if the path is nil and there is no currently selected master database via the UI

```
major update failed
minor update failed
```
if there is a problem with a database update

```
database requires major update but allow update flag is not set
database requires minor update but allow update flag is not set
```
if the database requires an update but the second parameter is false

override_user_unit (Exchange only – see note in method text)

OK=WSApplication.override_user_unit(code,value)

e.g.

```
OK=WSApplication.override_user_unit 'X','ft'
```

Overrides a current user unit for the duration of the script

This can be useful where ICM Exchange is running as a service, where the "user" has no settings of it own and you don't want the default user units that are selected for the locale

Returns true if successful, false for an unknown unit or value.

Note – when scripts are run from the UI, the user units will always be the ones set up in the UI (or the default).

override_user_units (Exchange only)

errs=WSApplication.override_user_units(filename)

e.g.

```
errs=WSApplication.override_user_units 'd:\\temp\\uu.csv'
if errs.length>0
      puts “error reading CSV file”
end
```

Bulk overrides the current user units for the duration of the script.

Takes a filename of a file containing comma separated pairs of unit code, unit value pairs, one pair per line. E.g.

```
XY,US Survey ft
```
Useful where ICM Exchange is running as a service, where the "user" has no settings of it own.

Returns an error string – an empty string indicates success. Note, however, that any valid units will be applied, regardless of whether there are any errors with other lines of the file.

A likely exception for this method to throw is:

```
Error 13 : File Not Found : z:\uu.csv (error=3: "The system cannot
find the path specified.")
```
if the file does not exist.

script_file

s=WSApplication.script_file

Returns the full path of the script file. This will either by the file specified on the command line of the Exchange product or the file selected in the user interface.

This method is supplied primarily to obtain the script's path in order to process the path to get the paths of files in the same directory e.g. config files for the Open Data Import and Export Centres.

set_exit_code (Exchange only)

WSApplication.set_exit_code(exit_code)

e.g.

```
WSApplication.set_exit_code 12
```

Sets the InfoLite application's exit code. The exit code must evaluate to a Fixnum, if it does not, an exception will be thrown:

```
exit code is not a number
```

Note that calling this method *does not* cause the script to terminate, it merely sets the exit code returned to the operating system when the script *does* terminate.

set_working_folder (Exchange only)

WSApplication.set_working_folder(path)

e.g.

```
WSApplication.set_working_folder 'd:\\temp\\wf'
```

Sets the working folder.

NB – this working folder will be used for the duration of this ICM Exchange run. it will not be stored in the registry and will not be used by any other instances of ICM Exchange or the ICM UI.

set_results_folder (Exchange only)

WSApplication.set_results_folder(path)

e.g.

```
WSApplication.set_results_folder 'd:\\temp\\rf'
```

Sets the results folder.

NB – this results folder will be used for the duration of this ICM Exchange run. it will not be stored in the registry and will not be used by any other instances of ICM Exchange or the ICM UI.

ui?

b=WSApplication.ui?

Returns true in interactive mode, false otherwise. The purpose of this method is to allow Ruby code to be written which does different things when run from within the user interface and when run from InfoWorks ICM / InfoAsset Exchange.

use_arcgis_desktop_licence (Exchange only – see note in method text)

WSApplication.use_arcgis_desktop_licence

When using the open data import and export centre methods with ArcGIS the software uses an ArcGIS server licence type by default. Use this method to use an ArcGIS desktop licence instead. It is the responsibility of the user to choose an appropriate ArcGIS licence based on their use of the software.

Note – when scripts are run from the UI, the desktop licence is, of course, used.

use_user_units?

b=WSApplications.use_user_units

e.g.

```
puts WSApplication.use_user_units?
```

Returns a Boolean indicating whether the application is using user units (default is false for the Exchange products and true within the user interface).

use_user_units=

WSApplication.use_user_units = flag

e.g.

```
WSApplication.use_user_units=true
```

Sets the flag indicating whether the application is using user units.

use_utf8?

b=WSApplications.use_utf8?

e.g.

```
puts WSApplication.use_utf8?
```

Returns a Boolean indicating whether the application is using UTF8 in string handling (default is false). See appendix 6 for more details.

use_utf8=

WSApplication.use_utf8 = flag

e.g.

```
WSApplication.use_utf8=true
```

Sets the flag indicating whether the application is using UTF8 in string handling (default is false). See appendix 6 for more details.

version

s=WSApplication.version

e.g.

```
puts WSApplication.version
```

Returns the InfoWorks ICM version number as a string e.g.

```
"2.5.0.5001"
```

WSDatabase

The majority of these methods are only available in the Exchange products. The primary purpose of the methods that ARE permitted from the product UIs is to allow navigation of the database in order to find and create selection list objects.

copy_into_root (Exchange only)

mo=iwdb.copy_into_root object,bCopySims,bCopyGroundModels

Given a model object (typically from another database) copies it into the database in the root, returning the new model object. The bCopySims and bCopyGroundModels parameters determine whether or not simulation results and ground models respectively are copied, corresponding to the equivalent user interface options.

file_root (Exchange only)

s=iwdb.file_root

Returns the root used for GIS files for the database – i.e. the path within which GIS file names will be treated as having relative paths. This will be the path shown in the UI as 'Remote files root' when the user selects the 'Select remote roots...' option.

If the database is a standalone database and the check-box is checked to force the roots to be below the master database, then the path returned will be the one used by the software in this situation i.e. the folder containing the master database.

find_model_object

mo=iwdb.find_model_object(type,name)

e.g.

```
puts iwdb.find_model_object('Model Network','MyNetwork')
```

Given the scripting type of an object and a name of a version controlled object (not its scripting path), will return the model object in the database of that type with that name. The name should not be the scripting full path, just the name of the object.

This method only works for version controlled objects and takes advantage of the fact that version controlled objects have unique names so the name, rather than scripting full path is enough to identify the object in the database uniquely.

find_root_model_object

mo=iwdb.find_root_model_object(type,name)

e.g.

```
mo=iwdb.find_root_model_object('Master Group','My Master Group')
```

Given the scripting type of an object and the name of an object, will return the model object with that name in the root of the database. Currently the only object types found in the root of the database are 'Master Group', 'Model Group' and 'Asset Group'

guid

s=iwdb.guid

Returns the GUID for the database, otherwise known as the 'database identifier'.

list_read_write_run_fields (ICM only)

arr=iwdb.list_read_write_run_fields

e.g.

```
iwdb.list_read_write_run_fields {|fn| puts fn}
```

Returns an array of strings containing all the fields in the Run object that are read-write fields i.e. may be set from ICM Exchange scripts.

model_object

mo=iwdb.model_object(scripting_path)

e.g.

```
mo=iwdb.model_object('>MODG~My Root Model Group')
```

Given a scripting path of an object returns the model object with that path, otherwise returns nil.

If you know the scripting path of the object in the database, this is the 'official' way of finding the model object.

model_object_collection

moc=iwdb.model_object_collection(type)

Given a scripting type, returns a WSModelObjectCollection of all the objects of that type – not just the objects of that type in the root.

model_object_from_type_and_guid (Exchange only)

mo=iwdb.model_object_from_type_and_guid(type,guid)

e.g.

```
mo=iwdb.model_object_from_type_and_guid 'Model Network,'{CEB7E8B9-
D383-485C-B085-19F6E3E3C8CD}'
```

Returns the model object of the given scripting type with the 'CreationGUID' (an internal database field) given in the second parameter.

model_object_from_type_and_id

mo=iwdb.model_object_from_type_and_id(type,id)

e.g.

```
mo=iwdb.model_object_from_type_and_id('Rainfall Event',1)
```

Returns the model object in this database with the type and ID given, the type is from the list of scripting types (see above), and the ID is the Id in the database which will be found by selecting the 'properties' option on the object.

This can be an easy way of getting a WSModelObject from something you have found in the InfoWorks ICM UI by selecting the object in the tree and obtaining its properties.

new_network_name (Exchange only)

name=iwdb.new_network_name(type,name,branch,add)

e.g.

```
newname=iwdb.new_network_name('Model Network',oldname,false,false)
```

The purpose of this method is to generate a name for a new network based on a given network name, with the intention that the old name is the name of an existing network and the new name will be used as the name of a subsequently created network.

The 3rd and 4th parameters are Boolean parameters which control the precise naming convention used. If branch is false the names will be got by appending #1 on the end if the name does not end in # followed by a number, if it does end in a # followed by a number the number will be incremented. If branch is true the same applies except that _ is used as the special character instead of #. If add is true then #1 or _1 will be appended to the name regardless of whether or not the name ends in # or _ followed by a number.

new_model_object (Exchange only)

mo=iwdb.new_model_object(type,name)

```
e.g.

root_master_group=iwdb.new_model_object('Master
Group','MyRootMasterGroup')
```

Given a scripting type and a name, creates a new object with that name in the root of the database. For this to work it is necessary that the type be a legal object type that can go into the root of the database i.e. this type must be "Asset Group","Model Group" or "Master Group". More may be added in the future.

Likely exceptions thrown by this method are:

unrecognised type – if the type is not a valid scripting type name

invalid object type for the root of a database – if objects of this type cannot be placed in the root of the database

an object of this type and name already exists in root of database – if an object with this type and name exists in the root of the database

licence and/or permissions do not permit creation of a child of this type in the root of the database – if the object cannot be created in the root of the database for licence and/or permissions reasons e.g. an attempt to create an Asset Group when the user doe not have an InfoAsset licence of some sort

unable to create object – if the creation fails for some other reason

path
s=iwdb.path

e.g.

```
puts iwdb.path
```

returns the pathname of the master database as a string

root_model_objects
moc=iwdb.root_model_objects

Returns a WSModelObjectCollection of all the objects in the root of the database.

use_merge_version_control? (WS Pro only)
b=iwdb.use_merge_version_control?

Returns true if the master database is using merge version control for new objects, false otherwise (i.e. the same behaviour as the checked menu item in the UI)

use_merge_version_control=
iwdb.use_merge_version_control=b

Sets the value of the flag which determines whether new objects in the master database will use merge version control (i.e. the same behaviour as the checked menu item in the UI)

WSModelObjectCollection

[]
mo=moc[n]

Returns the nth WSModelObject in the collection. The index is 0 based i.e. valid values are from 0 to length-1.

count
n=moc.count

Returns the number of WSModelObjects in the collection

each
WSModelObjectCollection.each do |mo|

e.g.

```
moc.each { |x| puts x.name }
```

The each method can be used to iterate through all the objects in the collection.

WSModelObject

The majority of these methods are only available in the Exchange products.

[] (ICM / InfoAsset only)
[]=(ICM / InfoAsset Exchange only)

mo['field']=value
v=mo['field']

The array operator is used to get and set values from fields in the object.

This is most useful for the InfoWorks Run object in InfoWorks ICM – the parameters for the InfoWorks run can be found in Appendix 2 (SWMM Runs and runs in WS Pro are handled differently – see WSRunScheduler and WSSWMMRunBuilder).

The parameter is invariably a field name, the value is a Ruby value of an appropriate type to be stored in the InfoWorks master database.

A special case is the case where the value being stored is the reference to another object in the database. In this case the value stored may be

a) The object as a WSModelObject (or class derived from it)
b) The ID of the object
c) The scripting path of the object

bulk_delete (Exchange only)

mo.bulk_delete

This deletes the object and all its children. The deleted objects are NOT put into the recycle bin, they are completely deleted.

children (Exchange only)

moc=mo.children

e.g.

```
master_group.children.each {|c| puts "child #{c.path}"}
```

Returns the children of the object as a WSModelObjectCollection

comment (Exchange only)

s=mo.comment

Returns the description (i.e. the text which appears in the description tab of the properties for an object) as a string.

comment= (Exchange only)

mo.comment=s

Sets the description (as described above) for an object from a string.

compare (ICM / InfoAsset Exchange only)

b=mo.compare(mo2)

e.g.

```
if mo.compare(mo2)
      puts "networks are identical"
end
```

Given another model object, compare the 2 objects returning true if they are identical, false otherwise. The objects must both be of the same type and must be version controlled objects or simulation results. There is currently a limitation that the simulation results may only be compared if the database from which they came is the current master database.

copy_here (Exchange only)

mo2=mo.copy_here(object,bCopySims,bCopyGroundModels)

Given a model object (typically from another database) copies it into the database as a child of this object, returning the new model object. The bCopySims and bCopyGroundModels parameters determine whether or not simulation results and ground models respectively are copied, corresponding to the equivalent user interface options.

csv_import_tvd (InfoAsset Exchange only)

arr=mo.csv_import_tvd(file_path, name, config_file_path)

Performs an import of time varying data into an asset group, creating one or more 'time varying data' objects in the same manner as the user interface when the import time varying data from generic CSV option is used. The parameters are the path of the file, the root name of the new object and the path of the config file saved from the user interface.

deletable? (Exchange only)

b=mo.deletable?

Returns true if the object can be deleted by normal means i.e. without using bulk_delete which can, of course, delete anything. This corresponds to whether the object can be deleted from the normal user interface and reflects the same rules e.g. does it have children, is it used in a simulation etc.

delete (Exchange only)

mo.delete

Deletes the object, providing it can be deleted in the normal user interface i.e. without using bulk_delete which can, of course, delete anything. This reflects, therefore, the rules used in the user interface.

delete_results (ICM Exchange only)

mo.delete_results

Deletes the results for the object, if the object is a simulation.

export (ICM & InfoAsset Exchange only)

mo.export(path,format)

Exports the model object in the appropriate format.

The formats permitted depend on the object type. The format string may affect the actual data exported as well as the format in which the data is exported e.g. for rainfall events the parameter 'CRD' means that the Catchment Runoff Data is exported.

When the format is the empty string the data is exported in the InfoWorks text file format. This format may be used for:

Inflow

Level

Infiltration

Waste Water

Trade Waste

Rainfall Event (non-synthetic)

Pipe Sediment Data

Observed Flow Event

Observed Depth Event

Observed Velocity Event

Layer List (this is a different file format but still termed the 'InfoWorks file' in the user interface).

Regulator *(from 6.0)*

For rainfall events the following parameters cause the export of other data in a text file format:

CRD – Catchment Runoff Data

CSD – Catchment Sediment Data

EVP – Evaporation

ISD – Initial Snow Data

TEM – Temperature Data

WND – Wind Data

For pollutant graphs the parameters listed in Appendix 1 list cause the export of the appropriate pollutant's data in the text file format.

If the format is 'CSV' the file will be exported in 'InfoWorks CSV' format for the following object types:

Level

Infiltration

Inflow

Observed Flow

Observed Depth

Observed Velocity

Rainfall Event (synthetic – main rainfall data)

Regulator

Damage Function *(from version 6.5)*

Waste Water *(from a version 7.5 patch)*

Trade Waste *(from a version 7.5 patch)*

(from version 6.5)

The results obtained by risk analysis runs may be exported as follows:

For Risk Analysis Results objects (known in ICM Exchange as Risk Calculation Results) the files may be exported by using the following in the format field:

"Receptor Damages"

"Component Damages"

"Code Damages"

"Impact Zone Damages"

"Category Damages"

"Inundation Depth Results"

For Risk Analysis Sim objects (known in ICM Exchange as Damage Calculation Results) the files may be exported by using the following in the format field:

"Receptor vs Code"

"Receptor vs Component"

"Code vs Component"

"Impact Zone vs Code"

"Impact Zone Code vs Component"

"Category Code vs Component"

For dashboards in InfoAsset Manager the format must be 'html', and the filename the name of the HTML file. Note in this case that other files are exported alongside the html file in the same folder. The names of these files are fixed for each individual dashboard object in the database so exporting the same dashboard object multiple times to different HTML files in the same folder will not give the intended results, you should instead export them to different folders.

find_child_model_object

mo_found=mo.find_child_model_object(type,name)

Given the scripting type of an object and a name will return the child of the object with that name and type.

id

n=mo.id

Returns the ID of the model object as a number

import_all_sw_model_objects(ICM Exchange only)

arr=mo.import_all_sw_model_objects(filepath,format,scenarioId,logFilepath)

Imports all swmm model objects that is present from a supported file.

Object types imported:
SWMM Network
Inflow
IWSW Run
IWSW Time Patterns
Selection List
Level
Rainfall Event
IWSW pollutograph
IWSW Climatology
Regulator

The format parameter may be "INP" (for SWMM5 files), "XPX"(XPSWMM/XPStorm files) or "MXD"(for InfoSWMM files).

ScenarioId is only used when importing from a MXD file. You may leave it blank for other formats.

The logfilepath is an optional parameter.

The object on which the method is called must be of a suitable type to contain the object imported.

The method returns an array of created objects

e.g.

```
arr=model_group.import_all_sw_model_objects('d:\\temp\\1.inp','INP',
'','d:\\temp\\log.txt')
```

import_data (ICM Exchange only)

import_data(format,filepath)

Imports data into an existing object.

This is only relevant for rainfall events and pollutographs because they contain multiple pages of data which must be imported and exported separately. You have the choice of either

a) Creating an empty object and importing all the data items using this method
b) Importing the first data item into a new object using import_new_model_object and importing subsequent items into the object using this method – in the case of rainfall events the first item must be the rainfall, in the case of pollutographs is can be any item.

Both InfoWorks CSV and InfoWorks file formats are supported. The parameter is 'CSV' for CSV files or some other string for the InfoWorks files.

For rainfall events the formats are SMD, EVP, SOL, WND, TEM, CSD, ISD, CRD and RED – for RED you can also put nil or ''. These are as documented in the main product documentation.

For pollutographs the format name is the name of the pollutant field, as detailed in Appendix 1.

The CSV files import the data into the blob based on data in the CSV file.

For rainfall events the CSV files may only be used for the time varying data i.e Rainfall, Temperature, Wind, Evaporation, Solar Radiation and Soil Moisture Deficit.

e.g.

```
moRain.import_data 'CSV','c:\\temp\\MyRain_EVP.csv'
moRain.import_data 'SMD','c:\\temp\\MyRain.smd'
moPG.import_data 'CSV'.'c:\\temp\\P.csv'
```

import_grid_ground_model (ICM & InfoAsset Exchange only)

mo.import_grid_ground_model(polygon,files,optionsHash)

Imports a gridded ground model into a model or asset group.

The first parameter must be nil or a WSRowObject (or object derived from it) from a currently open WSOpenNetwork object with polygon geometry.

The second parameter must be an array of one or more filenames.

The third parameter must be a hash, as follows:

Key	Type	Default	Notes
ground_model_name	String		Must be non-empty and unique in group
data_type	String		Displayed in the UI
cell_size	Float	1.0	Must be non-zero
unit_multiplier	Float	0.001	Must be non-zero
xy_unit_multiplier	Float	1	Must be non-zero
systematic_error	Float	0	

use_polygon	Boolean	false	Polygon is only used if this is true, if it is true the polygon must be non-nil
integer_format	Boolean	true	

e.g.

```
mg=db.model_object_from_type_and_id 'Model Group',2
files=Array.new
files << 'c:\\temp\\small_grid.asc'
fred=Hash.new
fred['ground_model_name']='fredi'
fred['data_type']='badger'
fred['cell_size']=5.0
fred['unit_multipler']=1.0
fred['xy_multiplier']=1.0
fred['integer_format']=false
fred['use_polygon']=false
mg.import_grid_ground_model nil,files,fred
```

import_infodrainage_object (ICM Exchange only)

mo=mo.import_infodrainage_object(filepath, objectType, logFile)

This method imports object/s of type objectType in a model group

The parameters are as follows:

filepath - full path for the InfoDrainage iddx file.

objectType - type of object to import. Only Inflow is currently imported.

logFile - full filepath where the import log is written. The log is in rich text format.

import_new_model_object (ICM Exchange only)

mo=mo.import_new_model_object(type,name,format,filepath)

This method Imports a new model object from a file.

Your code may have versinons of this with a fifth 'event' parameter. The event parameter is optional (from version 11.5), you wil almost certainly not need it, if you use it you should set it to 0.

Permitted types are:

- Inflow
- Level
- Infiltration
- Waste Water
- Trade Waste
- Rainfall Event (non-synthetic)

Pipe Sediment Data

Observed Flow Event

Observed Depth Event

Observed Velocity Event

Layer List (this is a different file format but still termed the 'InfoWorks file' in the user interface)

Regulator

Damage Function

Pollutograph

Except for pollutographs, the format parameter may be "" (for 'InfoWorks format' files) or "CSV" for InfoWorks format CSV files (not available for layer lists or damage functions).

The event parameter is for compatibility with old files and should in general be set to 0

The object on which the method is called must be of a suitable type to contain the object imported.

For pollutographs you can use CSV (the data imported will depend on the CSV file), or the 3 letter code of a pollutant type from Appendix 1. You can only import one pollutant, if you wish to import more into the same InfoWorks object you can use the import_data method. You can also use that method to import additional data into Rainfall Events.

e.g.

```
rainfall=model_group.import_new_model_object 'Rainfall Event','The
Rainfall','','d:\\temp\\1.red'
```

import_new_model_object_from_generic_CSV_files

arr=model_group.import_new_model_object_from_generic_CSV_files(type,name, file_or_files,config_file)

This model imports a new model object from a file using the generic CSV importer. It requires a config file previously set up in the UI.

The parameters are as follows:

Type - permitted types are as for the UI, i.e.:

Permitted types are:

Inflow

Level

Infiltration

Rainfall Event (non-synthetic)

Pipe Sediment Data

Observed Flow Event

Observed Depth Event

Observed Velocity Event

Regulator

Name. This is used as a prefix for the event name (and as with the UI ignored for multiple rainfall events).

File or files. This can be a filename or, for rainfall events only, a number of filenames in which case the behavior will be as for the 'import multiple files into an event' menu item.

Config file. The filename of the config file.

The return value is an array with 2 elements. The first element is the WSModelObject created. The second element is either nil or the warning message that would appear in the UI as a string.

import_new_sw_model_object (ICM Exchange only)

myobj=mo.import_new_sw_model_object(type,format,filepath,scenarioId,logFilepath)

Imports a new swmm model object from a file.

Permitted types are:

Inflow
IWSW Run
IWSW Time Patterns
Selection List
Level
Rainfall Event
IWSW pollutograph
IWSW Climatology
Regulator

The format parameter may be "INP" (for SWMM5 files), "XPX" (XPSWMM/XPStorm files) or "MXD" (for InfoSWMM files).

ScenarioId is only used when importing from a MXD file. You may leave it blank for other formats.

The logfilepath is an optional parameter.

The object on which the method is called must be of a suitable type to contain the object imported.

The method returns the object importer.

e.g.

```
model_group.import_new_sw_model_object('Rainfall
Event','INP','d:\\temp\\1.inp','','d:\\temp\\log.txt')
```

import_tvd(ICM & InfoAsset Exchange only)

mo.import_tvd(filename,format,event)

Imports event data into an existing object.

The parameters are as follows:

Filename – file to be imported

Format – 'CSV' or 'RED'

Event – integer – must be present but is ignored

If the format is 'CSV' this will import a CSV file in the 'InfoWorks CSV file' format into an existing event, overwriting the data already there if there is any.

If the format is 'RED' and the type of the object is a rainfall event this will import the data in event file format into an existing event, overwriting the data already there if there is any.

modified_by

s=mo.modified_by

Returns the name of the user who last modified the object as a string.

name

s=mo.name

Returns the name of an object as a string.

name= (Exchange only)

mo.name=s

Sets the name of an object from a string

new_model_object – limited functionality in UI, full in Exchange – see below

mo2=mo.new_model_object(type,name)

e.g.

```
child=mo.new_model_object('Model Network','MyNetwork')
```

Given a scripting type and a name will create a child object with that type and name. The type must be a type that can be contained in the object.

Runs must be created using the new_run method and sims cannot be created directly – they are created along with the runs based on the nature of the network and rainfall events / flow surveys used to create the run.

Risk analysis runs must be created using the new_risk_analysis_run method.

Likely exceptions to be thrown by this method are:

unrecognised type – if the type is not a valid scripting type

runs must be created using the new_run method – if an attempt is made to create a run

sims cannot be created directly – if an attempt is made to create a sim

invalid child type for this object – if the type in the first parameter may not be a child of this object type

name already in use – if the name is in use (globally for a version controlled object, as a child of this object for other types)

licence and/or permissions do not permit creation of a child of this type for this object – if this type of object cannot be created for licensing and/or permissions reasons e.g. creation of an Asset Group when the software is begin run without an InfoAsset licence of some sort.

unable to create object – if the call fails for some other reason.

new_risk_analysis_run (ICM Exchange only)

raro=mo.new_risk_analysis_run(name,damage_function,runs,param)

Creates a new risk analysis run object. The parameters are as follows:

Damage_function - The parameter must be the ID of the damage function object, its scripting path, or the WSModelObject (i.e. the damage function object returned from a suitable call).

Runs – This parameter must be one of

1. The ID of the 'normal' run object
2. The scripting path of the 'normal' run object
3. The WSModelObject representing the run object
4. An array of items which must all be of the types described in 1 to 3 above

Param – the numerical parameter

new_run (ICM Exchange Only – WS Pro Exchange uses a different mechanism)

run=mo.new_run(name,network,commit_id,rainfalls_and_flow_surveys,scenarios,parameters)

Creates a new run. The method must be called on a model group otherwise an exception will be thrown – 'new_run: runs may only be created in model groups'.

The method can take arrays as parameters for both the rainfalls and flow surveys and for the scenarios. In the same way that dropping multiple rainfall events and flow surveys into the drop target on the schedule run dialog and selecting multiple scenarios on it yield multiple simulations for a run, so calling this method with arrays of values and with synthetic rainfall events which have multiple parameters (singly or in an array) will yield multiple sims for the run.

The 'run' method which actually runs simulations is a method of the individual sim objects below the run, which may easily be found by using the 'children' method of the WSModelObject created using this method.

The parameters are as follows:

Name – the name of the new run. This must not already be in use in the model group otherwise an exception will be thrown – 'new_run: name already in use'.

Network – the network used for the run. The parameter must be the ID of the network, its scripting path, or the WSNumbatNetworkObject (i.e. the network object returned from a suitable call).

Commit ID – the commit ID to be used for the run. This may either be the integer commit ID or nil in which case the latest commit at time the run is created will be used.

Rainfalls and flow surveys – this may be:

1. nil – in this case the run will be a dry weather flow run
2. a WSModelObject which is a rainfall event or a flow survey
3. the scripting path of a rainfall event or a flow survey as a string
4. the ID of a rainfall event
5. a negative number equal to -1 times the ID of a flow survey e.g. -7 means the Flow Survey with ID 7.
6. An array. If the parameter is an array, then if the length of the array is 0 then the event will be a dry weather flow run, otherwise all the array elements must be one of 2 – 5 above. The array may not contain duplicates otherwise an exception will be thrown.

Scenarios – this may be:

1. nil – in this case the run will use the base scenario of the network
2. the name of a scenario
3. an array in which each element is the name of a scenario as a string. It must not contain scenarios that do not exist or duplicates.

Parameters – this must be a hash containing parameters (see Appendix 2).

Parameters for the run may either be set in this call.

update_to_latest (Exchange only)

mo.update_to_latest

This method may only be used on runs. The following conditions must apply:

a) The 'Working' field must have been set to true
b) There must be no uncommented changes for the network for the run.
c) all scenarios which were included in the list of scenarios for which the run was set up must be present and validates.

This method has the same effect as pressing the 'update to latest version of network' button on the Run view in the user interface.

open (Exchange only)

opennet=mo.open

Returns a WSOpenNetwork object corresponding to the model object, providing the model object is of a network type or a sim – see above for the description of the difference between a WSModelObject and a WSOpenNetwork.

When this method is called on a sim, the network will be opened with the results of the simulation loaded into it. An exception will be thrown if the simulation did not succeed, the results are inaccessible or are not open.

When you open the results of a simulation:

- The network is opened as read only
- The current scenario is set to the scenario used for the simulation (for ICM Exchange)
- The current scenario cannot be changed (for ICM Exchange)
- As with the behaviour in the UI of the software, the network with the results loaded has a current timestep. The results start by being opened at the first timestep (timestep 0) unless there are only maximum results in which case they are opened as the maximum results timestep.

path

s=mo.path

e.g.

```
puts mo.path
```

Returns the scripting path of the object as a string.

parent_type

s=mo.parent_type

Returns the scripting type of the parent of the object (or 'Master Database' if the object is in the root of the database).

parent_id

n=mo.parent_id

Returns the ID of the parent object (or 0 if the object is in the root of the database).

status (set / get) (WS Pro Exchange only)

l=mo.status

mo.status=7

Sets and gets a status field stored in the database for the model object – the field is a long integer and can be used for any purpose. Unless set by this method the value is null in the database which is treated as nil by Ruby. The value is not visible in the UI

type

s=mo.type

Returns the scripting type of the object.

Network Classes:

WSNumbatNetworkObject / WSNetworkObject

The WSNumbatNetworkObject class is used in ICM Exchange , InfoAsset Exchange and WS Pro Exchange, the WSNetworkObject class is used WS Pro Exchange. There are some methods in common which are described first (although some of the parameters are different in some cases).

Common

csv_import

nno.csv_import(filename,options)

Updates the network from a CSV file with options similar to those available in the user interface. In WSPro Exchange the version controlled object must be checked out. The second parameter may either be nil or a hash containing the options. If the parameter is nil then the default options will be used. The hash values are as follows:

Key	Type	Default	Notes
Force Link Rename	Boolean	TRUE	
Flag Genuine Only	Boolean	FALSE	
Load Null Fields	Boolean	TRUE	
Update With Any Flag	Boolean	TRUE	True to update all values, false to only update fields with the 'update flag' flag
Use Asset ID	Boolean	FALSE	
User Units	Boolean	TRUE	Set to true for User Units, false for Native Units - used for fields without an explicit unit set in a 'units' record
UK Dates	Boolean	FALSE	If set to true, the import is done with the UK date format for dates regardless of the PC's settings
Action	String	Mixed	One of Mixed,'Update And Add','Update Only',Delete'
Header	String	ID	One of ID,'ID Description','ID Description Units','ID Units'
New Flag	String		Flag used for new and updated data
Update Flag	String		If the 'update with any flag' option is set to false, only update fields with this flag value

csv_export

nno.csv_export(filename,options)

Exports the network to a CSV file with options similar to those available in the user interface.

The second parameter may either be nil or a hash containing the options. If the parameter is nil then the default options will be used. The hash values are as follows:

Key	Type	Default	Notes
Use Display Precision	Boolean	TRUE	
Field Descriptions	Boolean	FALSE	
Field Names	Boolean	TRUE	
Flag Fields	Boolean	TRUE	
Multiple Files	Boolean	FALSE	Set to true to export to different files, false to export to the same file
Native System Types	Boolean	FALSE	ICM / InfoAsset only
User Units	Boolean	FALSE	
Object Types	Boolean	FALSE	
Selection Only	Boolean	FALSE	
Units Text	Boolean	FALSE	
Triangles	Boolean	FALSE	ICM Model Networks only
Coordinate Arrays Format	String	Packed	One of Packed, None or Separate
Other Arrays Format	String	Packed	One of Packed, None or Unpacked

e.g.

```
myHash=Hash.new
myHash['Multiple Files']=true
myHash['Coordinate Arrays Format']='None'
no.csv_export 'd:\\temp\\network.csv',myHash
```

odec_export_ex

nno.odec_export_ex(format, config_file, options, ...)

This method exports network data using a previously set up Open Data Export Centre configuration.

The number of parameters for this method is variable and depends on the type of data imported (determined by the first parameter) and the number of imports set up in one call to the method, which can be more then one.

The first parameter can be one of the table below. Depending on that value, the number of parameters per table is as shown in the table:

Format	Meaning	Parameters per table
CSV	CSV	2
TSV	Tab separated text data (not to be confused with TAB files)	2
XML	XML	4
MDB	Access database	3
SHP	Shape file	2
MIF	MapInfo Interchange Format	2
TAB	TAB file (MapInfo format binary file) *(not WS Pro)*	2
GDB	GeoDatabase	6
ORACLE	Oracle	7
SQLSERVER	SQL Server	9

The GDB option is only available with a suitably licenced and available ESRI desktop or server product.

The TAB option (but not the MIF option) is only available if your copy of InfoWorks ICM was installed with the installer with the MapXTreme component.

The ORACLE and SQLSERVER options require the relevant client software to be installed.

The second parameter is the path of the CFG file, as saved from the Open Data Import Centre user interface. It can contain the mappings for more than one table.

The third parameter must be nil or a hash containing the options, which broadly correspond to those in the Open Data Export Centre user interface as follows:

Key	Type	Default	Notes

Callback Class	Ruby Class	nil	ICM / InfoAsset Only
Script File	String	nil	WS Pro Only
Error File	String	nil	
Image Folder	String	""	Asset Networks Only
Units Behaviour	String	Native	Native or User
Report Mode	Boolean	FALSE	True to export in 'report mode'
Append	Boolean	FALSE	True to enable 'Append to existing data'
Export Selection	Boolean	FALSE	True to export the selected objects only
Previous Version	Integer	0	Previous version, if not zero differences are exported
Create Primary Key	Boolean	FALSE	ICM / InfoAsset Only
Previous Version	Integer	0	ICM / InfoAsset Only
Append	Boolean	FALSE	ICM / InfoAsset Only
WGS84	Boolean	FALSE	ICM / InfoAsset Only
Don't Update Geometry	Boolean	FALSE	ICM / InfoAsset Only

Following these three parameters, there then follow a multiple of the 'parameters per table' depending on data type, as show in the table above. It is possible to export data into more than one table by supplying a multiple of that number of parameters e.g. to export data from 3 tables into CSV files, six parameters are supplied for these bringing the total to 9 parameters. Alternatively, of course, it is possible to export the data by making three calls to the method.

The parameters are as follows:

In all cases the first parameter is the name of the table to export. In this context the name is as displayed in the software's user interface in the Open Data Export Centre with any spaces removed. E.g. Node, CCTVSurvey. This differs from the convention used elsewhere e.g. for SQL.

For CSV, TSV, MIF, TAB and Shape files the second parameter is the path of the file to export.

For Access databases (MDB) the parameters are as described below:

1- Table to export
2- Name of table in Access database
3- File name of Access database

For XML the parameters are as follows:

1- Table to export
2- 'Feature class' (as it is called in the UI – i.e. the name of the root element)
3- 'Feature dataset' (as it is called in the UI- i.e. the name used for each data element)
4- XML file name.

For GeoDatabase the parameters are as follows:

1- Table to export
2- Feature class
3- Feature dataset
4- Update – true to update, false otherwise. If true the feature class must exist.
5- ArcSDE configuration keyword – nil for Personal / File GeoDatabases, and ignored for updates
6- Filename (for personal and file GeoDatabases, connection name for SDE)

For Oracle the parameters are as follows:

1- Table to export
2- Table name in Oracle
3- Table owner
4- Update – true to update, false otherwise. If true the table must exist.
5- User name
6- Password
7- Connection string

For SQL Server the parameters are as follows:

1- Table to export
2- Table name in SQL Server
3- Server name
4- Instance name
5- Database name
6- Update – true to update, false otherwise
7- Integrated security – true if integrated, false if not
8- User name
9- Password

odic_import_ex

nno.odic_import_ex(format, config_file, options, ...)

This method imports and updates network data using a previously set up Open Data Import Centre configuration.

Unlike the WSOpenNetwork version, it returns nil.

The number of parameters for this method is variable and depends on the type of data imported (determined by the first parameter) and the number of imports set up in one call to the method, which can be more then one.

The first parameter can be one of the table below. Depending on that value, the number of parameters per table is as shown in the table:

Format	Meaning	Parameters per table
CSV	CSV	2
TSV	Tab separated text data (not to be confused with TAB files)	2
XML	XML	3
MDB	Access database	3
SHP	Shape file	2
TAB	MapInfo TAB file (ICM & InfoAsset only)	2
GDB	GeoDatabase	3
ORACLE	Oracle	6
SQLSERVER	SQL Server	SEE BELOW

The SQL Server export has 10 parameters per table for WS Pro and 8 for ICM and InfoNet

The GDB option only available with a suitably installed and licenced copy of an ESRI server or desktop product.

The ORACLE and SQLSERVER options require the relevant client software to be installed.

The second parameter is the path of the CFG file, as saved from the Open Data Import Centre user interface. It can contain the mappings for more than one table.

The third parameter must be nil or a hash containing the options, which broadly correspond to those in the Open Data Import Centre user interface as follows:

Key	Type	Default	Notes
Error File	String	Blank	Path of error file
Callback Class	Ruby Class	nil	Class used for Ruby callback methods (ICM & InfoAsset only)
Script File	String	blank	Path of VBScript file
Set Value Flag	String	blank	Flag used for fields set from data

Default Value Flag	String	blank	Flag used for fields set from the default value column
Image Folder	String	blank	folder to import images from (asset networks only)
Duplication Behaviour	String	Merge	One of 'Duplication Behaviour','Overwrite','Merge','Ignore'
Units Behaviour	String	Native	One of 'Native','User','Custom'
Update Based On Asset ID	Boolean	FALSE	
Update Only	Boolean	FALSE	
Delete Missing Objects	Boolean	FALSE	
Allow Multiple Asset IDs	Boolean	FALSE	
Update Links From Points	Boolean	FALSE	
Blob Merge	Boolean	FALSE	
Use Network Naming Conventions	Boolean	FALSE	
Import Images	Boolean	FALSE	Asset networks only
Group Type	String	blank	Asset networks only
Group Name	String	blank	Asset networks only
Network	WSOpenObject	Nil	WS Pro Only (see below)
Connect Pipes To Dummy Nodes	Boolean	FALSE	WS Pro Only (see below)
Dummy Node User Field	String	Nil	WS Pro Only (see below)

Following these three parameters, there then follow a multiple of the 'parameters per table' depending on data type, as show in the table above. It is possible to import data into more than one table, or data into one table from more than one source by supplying a multiple of that number of parameters e.g. to import data into 3 tables from CSV files, six parameters are supplied for these bringing the total to 9 parameters. Alternatively, of course, it is possible to import the data by making three calls to the method.

All these parameters are strings with the exception of one parameter for SQL Server, which is the sixth out of eight parameters and is a Boolean which is described below.

The parameters are as follows:

In all cases the first parameter is the name of the table to import. In this context the name is as displayed in the software's user interface in the Open Data Import Centre with any

spaces removed. E.g. Node, CCTVSurvey. This differs from the convention used elsewhere e.g. for SQL.

For subtables the name is the name of the table, followed by the name of the subtable without any spaces e.g. to import the Details from the CCTV Survey, the string should be CCTVSurveyDetails.

For CSV, TSV and Shape files the second parameter is the path of the file to import.

For Access databases the second parameter is the path of the Access database and the third parameter is the name of the table or stored query to import. Data can be imported on the basis of SQL queries by adding an SQL query to the database, however an SQL expression cannot be used in this parameter.

For GeoDatabases the second parameter is the name of the feature class to import from and the third parameter is the path of the GeoDatabase.

For Oracle the parameters are as follows:

1 – the table name to import into as described above

2 – the table name in the Oracle database

3 – the connection string e.g. '//power:/orcl'

4 – the owner of the Oracle table being imported from

5 – the oracle user name

6 – the oracle password

For SQL server the parameters are as follows for WS Pro:

1– the table name to import into as described above

2 – the table name in SQL Server

3 – A 'where' clause to filter the import at the database level (this is intended to allow more efficient imports than fetching all the data and doing the filtering in VBScript in cases where you are doing simple filtering – as such it allows field values, constant numbers and strings, arithmetic and logical operators, null tests and some simple string functions)

4 – An 'order by' clause to order the input (likely to mainly be useful for importing blob data)

5 – the server name

6 – the SQL Server instance name

7 – the database name

8 – integrated security (Boolean – true or false, true meaning integrated security)

9 – user name (for non-integrated security, nil otherwise)

10 - password (for non-integrated security, nil otherwise)

For ICM and InfoAsset Manager they are

1– the table name to import into as described above

2 – the table name in SQL Server

3 – the server name

4 – the SQL Server instance name

5 – the database name

6 – integrated security (Boolean – true or false, true meaning integrated security)

7 – user name (for non-integrated security, nil otherwise)

8- password (for non-integrated security, nil otherwise)

(i.e. these products do not have the 3rd and 4th parameters that WS Pro has)

The 'Network' parameter may only be used in WS Pro when this method is used to import a control. The parameter must be a WSOpenNetwork object of a network and this can only be used in conjection with 'Update Based on Asset ID'. The purpose of this functionality is to allow controls to be imported and updated in tandem with the network – i.e. it provides a way of updating the control based purely on the asset ID without requiring the data source to know the primary key of the object being updated, which may not be readily available in the data source being imported.

The 'Connect Pipes To Dummy Nodes' parameter works as follows: if a pipe goes from 12345 to 45678 say, and we have imported 12345 or 45678 as point objects that we represent as links we will have created dummy nodes 12345X, 12345Y, 45678X and 45678Y. Therefore if at this point we find that 12345 doesn't exist but 12345Y does, and similarly 45678 does but 45678X does then we change the value in the US node ID / DS node ID field for this pipe to the X or Y version as appropriate (it is X for downstream and Y for upstream since we expect them to connect to the downstream and upstream ends of the link respectively)

The 'Dummy Node User Field' parameter if set to the name of a string field will set that string field to 'Y' when a dummy node is added at the end of a link.

remove_local

nno.remove_local

Removes the local copy of the network, any uncommitted changes will be lost.

WSNetworkObject Only

This class is used for lock version control in WS Pro only

branched?

b=no.branched?

Returns true of the object is branched, false otherwise.

check_in
no.check_in

Checks in the version controlled object

check_out
new_object=no.check_out(new_name)

Checks out the version controlled object, giving the checked out version the name newname and returns the new object.

checked_out_by
s=no.checked_out_by

Returns the user-name of the user who has the object currently checked out. An empty string is returned if the object is not checked out.

checked_out?
b=no.checked_out?

Returns true if the object is checked out, false otherwise.

check_out_and_branch
new_object=no.check_out_and_branch(new_name)

As check_out but the checked out object is branched

undo_check_out
no.undo_check_out

For a checked out version controlled object, undoes the check-out i.e. effectively deletes it. The method delete will, in fact, perform the same function, but undo_check_out will only work for checked out objects so is slightly safer.

WSNumbatNetworkObject Only Version Control Methods

This class is used for Infoworks ICM and InfoAsset, and well as for WS Pro when merge version control is used.

The following methods are directly related to version control and are available in all 3 products.

branch
new_nno=nno.branch(commitid,newname)

Branches a version controlled object creating a new version controlled object, branches based on the commit ID (these can be found from the commits method below or from the user interface). The commit ID must be in the list of commits for that object.

commit
id=nno.commit(comment)

e.g.

```
nno.commit 'this is the commit comment for this commit'
```

or

```
id=nno.commit('this is the commit comment for this commit')
```

Commits changes to a network to the database. Returns the commit ID or returns nil if there were not changes made since the last commit.

commits

c=nno.commits

Returns a WSCommits object containing the commits for the network i.e. the details of what is committed at each stage in the history of the network.

e.g.

```
mo.commits.each do |c|
      puts
"#{c.branch_id},#{c.comment},#{c.commit_id},#{c.deleted_count},#{c.i
nserted_count},#{c.modified_count},#{c.setting_changed_count},|#{c.u
ser}|"
end
```

or

```
(0...mo.commits.length).each do |i|
      c=mo.commits[i]
      puts
"#{c.branch_id},#{c.comment},#{c.commit_id},#{c.deleted_count},#{c.i
nserted_count},#{c.modified_count},#{c.setting_changed_count},|#{c.u
ser}|"
end
```

commit_reserve

id=nno.commit_reserve(comment)

This is similar to commit in that it commits changes to the network to the database and returns the commit ID if changes have been made, nil otherwise. The difference is that commit_reserve keeps the network reserved.

current_commit_id

id=nno.current_commit_id

Returns the commit ID of the local copy of the network – this may not be the latest commit ID i.e. the most recent commit ID on the server, which is returned by latest_commit_id (see below).

latest_commit_id

id=nno.latest_commit_id

Returns the latest commit ID for the network i.e. the ID of the most recent commit on the server. This may not be the same as the commit ID of the local copy, which is returned by current_commit_id (see above).

open_version

on=nno.open_version(commit id)

This opens a previous commit of a version controlled object as a WSOpenNetwork.

reserve

nno.reserve

Reserves the network so no-one else can edit it and updates the local copy to the latest version.

revert

nno.revert

Reverts any uncommitted changes in the local copy of the network.

uncommitted_changes?

b=nno.uncomitted_changes?

Returns true if there are uncomitted changes in the local copy of the network, false otherwise.

unreserve

nno.unreserve

Cancels the reservation of the network.

update

b=nno.update

Updates the local copy of the network from the server. Returns true if there are no conflicts, false if there are.

WSNumbatNetworkObject Only Other Methods

This class is used for Infoworks ICM and InfoAsset, and well as for WS Pro when merge version control is used.

The following methods are not directly related to version control and are only available in ICM Exchange and InfoAsset Exchange

csv_changes

nno.csv_changes(commit_id1, commit_id2, filename)

Outputs the differences between commit id 1 and commit id 2 to the specified filename. The CSV file output is in the same format as generated by the "Compare network" function and can be used to apply the changes to another network via "Import/Update from CSV files..." functionality.

GIS_export

nno.GIS_export(format,params,location)

Exports the network data to GIS.

The parameters are as follows:

Format :

This parameter must be one of the following:

- SHP – Shape file
- TAB – Tab file
- MIF – MIF file
- GDB – GeoDatabase

Params:

This parameter can either be a hash, described below, or nil. If the parameter is nil then the default values for its various options will be used.

Folder:

This parameter contains the base folder for the files to be exported, except for the GDB format where it is the name of the GeoDatabase.

The params hash can contain a number of keys. If the hash is nil or the value is not specified the default behaviour applies as described below.

ExportFlags – Boolean – if this is true then the flags are exported along with the data values, if false they aren't. The default is true.

Feature Dataset – String – for GeoDatabases, the name of the feature dataset. The default is an empty string.

SkipEmptyTables – Boolean – if true skips empty tables (even if they are listed in the value for the Tables key). The default is false.

Tables – Array as described below – the default is to export results for all tables.

UseArcGISCompatibility – Boolean – this is the equivalent of selecting the check-box in the UI. The default is false i.e. the equivalent of not checking the check-box in the UI.

The tables element of the hash must be an array of strings which must be names of tables as returned by the list_GIS_export_tables. Duplicates and unrecognised values are not permitted.

list_GIS_export_tables

arr=nno.list_GIS_export_tables

This method lists the tables that will be exported to GIS as an array of strings. This method is designed to return information useful when customising the parameters to GIS_export.

select_changes

nno.select_changes

Select all objects added or changed between the commit id specified and the current network.

The network must have no outstanding changes for this to work (otherwise an exception will be thrown).

Obviously, deleted objects will not be selected.

select_clear

nno.select_clear

Clears the selection in the network

select_count

n=nno.select_count

Returns the number of selected items in the network

select_sql

n=nno.select_sql(table,query)

e.g.

```
n=nno.select_sql('hw_node','x>0')
```

This method runs the SQL query with the given table being the 'current table' as it appears in the SQL dialog i.e. the default table if the SQL query does not further qualify the table name.

The names '_nodes' or '_links' can be used to run the SQL over all node or link tables in the same way as can be achieved by selecting 'All nodes' or 'All links' in the dropdowns in the SQL dialog.

The SQL query can include multiple clauses in the same was as SQL run from the user interface, and can use any of the options that do not open results or prompt grids e.g.

```
nno.select_sql 'hw_node', "SELECT distinct x into  file
'd:\\temp\\distinctx.dat'"
```

The method returns the number of objects selected in the last clause in the SQL block selecting a non-zero number of object (or zero if there is no such clause).

user_field_names

nno.user_field_names(filename,arbitrarystring)

Produces a CSV file consisting of the names of all the user fields for all the object types in the network.

The CSV file has no header and the arbitrary string is output as the first column, the second column being the internal table name and the third column being the user field name.

WSSimObject

An object of this type represents an ICM Sim object, a WS Sim object, an ICM Risk Analysis Results object or an ICM Risk Analysis Sim object. The Risk Analysis Results objects contain the results for a number of return periods and summary results. The Risk Analysis Sim objects contain only summary results.

The different return periods for the Risk Analysis Results objects correspond to the timesteps for 'normal' simulations. The names of the methods reflect the usage for 'normal' simulations i.e. to list the return periods for a risk analysis results object you should use the list_timesteps method.

Various methods cannot be called for the Risk Analysis Sim objects if those methods are not relevant for sims with only summary results.

This chart shows which methods are available for which object types:

	ICM Sim	WS Pro Sim	Risk Analysis Results	Risk Analysis Sim
list_max_results_attributes	*		*	*
list_results_attributes	*		*	
list_results_GIS_export_tables	*	*	*	*
list_timesteps	*		*	*
max_flood_contours_export	*			
max_results_binary_export	*		*	*
max_results_csv_export	*			
results_binary_export	*		*	
results_csv_export	*	*		
results_csv_export_ex	*	*		
results_GIS_export	*	*	*	*
results_path	*			
run	*	*		
run_ex	*	*		
status	*	*		
success_substatus	*	*		
get	*			
timestep_count	*	*	*	*

list_max_results_attributes

arr=sim.list_max_results_attributes

Returns an array of arrays. The arrays returned correspond to the tabs on the binary results export dialog e.g. 'Node', 'Link', 'Subcatchment' for the time varying results. Each array contains 2 elements, the first of which is the name of the Tab, whilst the 2nd is an array of attribute names e.g.

```
[['Scalar',['totfl','totout','totr','totrun','totvol']],['Node',['fl
ooddepth','floodvolume','flvol','pcvolbal','qincum','qinfnod','qnode
','qrain','vflood','vground','volbal','volume','DEPNOD']]]
```

This method is designed to return information useful when customising the parameters to max_results_binary_export.and max_results_csv_export

list_results_attributes

arr=sim.list_results_attributes

Returns an array of arrays. The arrays returned correspond to the tabs on the binary results export dialog e.g. 'Node', 'Link', 'Subcatchment'. Each array contains 2 elements, the first of which is the name of the Tab, whilst the 2nd is an array of attribute names e.g.

```
[["Node", ["flooddepth", "floodvolume", "flvol", "qinfnod", "qnode",
"qrain", "volume", "depnod"]]
 ["Link", ["ds_depth", "ds_flow", "ds_froude", "ds_vel", "HYDGRAD",
"Surcharge", "us_depth", "us_flow", "us_froude", "us_vel",
"qinflnk", "qlink", "volume", "pmpstate"]]
 ["Subcatchment", ["qfoul", "qsurf01", "qsurf02", "qsurf03",
"qtrade", "RAINFALL", "RUNOFF"]]
 ["Rain Gauge", ["raindpth", "RAINFALL"]]]
```

This method is designed to return information useful when customising the parameters to results_binary_export.and results_csv_export

list_results_GIS_export_tables

arr=sim.list_results_GIS_export_tables

Returns an array of the tables that may be exported to GIS using the results_GIS_export method. This method is designed to return information useful when customising the parameters to results_GIS_export. The list will essentially only change when more tables are added to InfoWorks. The current list is:

```
_2DElements
_links
hw_1d_results_point
hw_2d_bridge
hw_2d_linear_structure
hw_2d_results_line
hw_2d_results_point
hw_2d_results_polygon
hw_2d_sluice
hw_bridge
hw_bridge_opening
hw_node
hw_river_reach
```

hw_subcatchment
hw_tvd_connector
Note the results for 2D elements is _2DElements and that the table name used for all links, which are combined into one GIS layer, is _links.

list_timesteps

arr=list_timesteps

For a normal simulation, this returns an array of the results timesteps for the simulation. See the 'Dates and Times' section above for details on how absolute and relative times are handled.
For a risk analysis results object, this returns an array of the return periods for the object.

max_flood_contours_export

max_flood_contours_export(format,groundModel,theme,filename)

Exports the flood contours to files (GIS or ASCII). The ASCII format is the same as produced via the user interface.

Format: MIF, TAB, SHP or ASCII – as with the user interface GeoDatabases are not supported.

GroundModel: this may either be the scripting path, the ID or the WSModelObject representing a ground model (either grid or TIN). If the ID is negative then it represents a TIN ground model i.e. -7 represents the TIN ground model with ID 7. If the ID is positive it represents a gridded ground model.

For the ASCII export, this must be a grid ground model.

Theme: the script path, ID or WSModelObject representing theme to use for the contours.

Filename: the filename to be exported.

max_results_binary_export

max_results_binary_export(selection,attributes,file)

Exports the maximum results (and other summary results) for the simulation in a binary file format intended for reading by program written by a user or 3rd party. The format is documented in a separate document available from Innovyze.

The selection parameter may contain the ID or scripting path of a selection list object, or the WSModelObject itself, or may be nil in which case results for the whole network will be exported.

The attributes parameter may be nil, in which case all attributes are exported, or may be an array as described in the list_max_results_attributes method above.

results_binary_export

results_binary_export(selection,attributes,file)

Exports the results for each results timestep for the simulation in a binary file format intended for reading by program written by a user or 3rd party. The format is documented in a separate document available from Innovyze.

The selection parameter may contain the ID or scripting path of a selection list object, or the WSModelObject itself, or may be nil in which case results for the whole network will be exported.

The attributes parameter may be nil, in which case all attributes are exported, or may be an array as described in the list_max_results_attributes method above.

max_results_csv_export

results_csv_export(selection, attributes, folder)

Exports the results for the simulation in the CSV format corresponding to that used in the CSV results export menu option.

The selection parameter may contain the ID or scripting path of a selection list object, or the WSModelObject itself, or it may be nil in which case results for the whole network will be exported.

The attributes parameter may be nil, in which case all attributes are exported, or may be an array as described in the list_max_results_attributes method above (i.e. as max_results_binary_export).

(Note that the parameters for this method are the same as those for results_csv_export_ex)

results_csv_export

results_csv_export(selection,folder)

Exports the results for the simulation in the CSV format corresponding to that used in the CSV results export menu option.

The selection parameter may contain the ID or scripting path of a selection list object, or the WSModelObject itself, or it may be nil in which case results for the whole network will be exported.

results_csv_export_ex

This is different for ICM and WS Pro.

For ICM:

results_csv_export_ex(selection, attributes, folder)

As for results_csv_export but takes an extra attributes parameter which may be nil, in which case all attributes are exported, or may be an array as described in the list_results_attributes method above (i.e. as results_binary_export).

For WS Pro:

results_csv_export_ex(selection, folder, options)

As for results_csv_export but takes an extra options parameter which must be nil or ar a hash. If a hash the only valid key is 'Group By Time' (default value false). If set to true then the results are grouped by time in the same way as the UI option.

results_GIS_export

results_GIS_export(format,timesteps,params,folder)

Exports simulation results to GIS.
The parameters are as follows:

Format

This parameter must be one of the following:

- SHP – Shape file
- TAB – Tab file (ICM / InfoAsset only)
- MIF – MIF file
- GDB – GeoDatabase

Timesteps:

The timesteps parameter may take a number of values of different types as follows:

- nil – if the parameter is nil this is the equivalent of the 'None' option when selecting timesteps in the UI i.e. it only makes sense if the appropriate option in the params parameter hash is set so that maximum results are exported for the simulation.
- 'All' – if the parameter is set to the string 'All' then all timesteps will be exported (this does not include the maximum results – if these are wanted then the appropriate parameter in the params hash should be set)
- 'Max' – if the parameter is set to the string 'Max' then the maximum results will be exported. This can also be achieved by setting the appropriate value in the params hash (not applicable to WS which does not have max results)
- An integer ('Fixnum') representing a timestep with 0 representing the first timestep. The number of timesteps may be found via the timestep_count method or by counting the number of timesteps in the result of the list_timesteps method. The maximum value permitted is the number of timesteps minus 1.
- An array of integers representing the timesteps as described above. The values must all be valid and it may not contain duplicates. This can be used to flexibly choose which timesteps to export in a similar manner to the more complex UI options.

Params:

This parameter can either be a hash, described below, or nil. If the parameter is nil then the default values for its various options will be used.

Folder:

This parameter contains the base folder for the files to be exported, except for the GDB format where it is the name of the GeoDatabase.

The params hash can contain a number of keys. If the hash is nil or the value is not specified the default behaviour applies as described below.

2DZoneSQL – Array as described below – the default is not to export any extra fields for 2D elements. (doesn't apply to WS)

AlternativeNaming – Boolean – if this is set then the subfolders / feature datasets used for the export are given simpler but less attractive names which may be helpful if the aim is process the files with software rather than to have a user select and open them in a GIS package. The simple names are <model object id>_<timestep> with the timesteps numbered from zero as with the timesteps parameter of the method and with <model object id>_Max for the maxima. The default is to use the same naming convention as the UI.

ExportMaxima – Boolean – if this is set to true the maximum results are exported. The default is false i.e. to not export them. (doesn't apply to WS)

Feature Dataset – String – for GeoDatabases, the name of the feature dataset. The default is an empty string.

Tables – Array as described below – the default is to export results for all tables.

Threshold – Double – the depth threshold below which a 2D element is not exported. This is the equivalent of checking the check-box in the UI and entering a value. The default is to behave as though the check-box is unchecked i.e. all elements are exported. (doesn't apply to WS)

UseArcGISCompatibility – Boolean – this is the equivalent of selecting the check-box in the UI. The default is false i.e. the equivalent of not checking the check-box in the UI. (doesn't apply to WS)

The tables element of the hash must be an array of strings which must be names of tables as returned by the list_results_GIS_exports_table. Duplicates and unrecognised values are not permitted.

The 2DZoneSQL element of the hash must be an array of arrays. Each of those arrays must in turn contain 2 or 3 values.

The first value is the name of the field to be exported as a string.

The second value is the SQL expression as a string.

The optional 3rd value must be an integer between 0 and 9 inclusive representing the number of decimal places. If this is not set then a default of 2 decimal places is used.

run

sim.run

This method runs a simulation, waiting until the simulation finishes, however long that may be. There are alternative mechanisms for running simulations which permit greater control over the process. The simulation will be run on the current machine.

run_ex

sim.run_ex(server,number_of_threads) [ICM only] OR
sim.run_ex(hash) [ICM only] OR
sim.run_ex(hash) [WS Pro only]

This method behaves as run above, except that you may control the server on which the simulation is run and various other aspects.

In ICM:

You can either have two parameters, a string and an integer, or one parameter, a hash.

In the former case (a string and an integer):

The first parameter server name may be a machine name or one of '.' or '*'. '.' has the meaning of local machine and '*' has the meaning of any available server.

The second parameter may be an integer representing the number of threads. Using 0 as the number of threads causes as many threads as possible to be used as with the user interface option.

In the latter case (a hash):

The keys are as follows:

Server – string - server name - may be a machine name or one of '.' or '*'. '.' has the meaning of local machine and '*' has the meaning of any available server, default value '.'

Threads – integer - the number of threads. Using 0 as the number of threads causes as many threads as possible to be used as with the user interface option.

SU – Boolean – if you are using InfoWorks One you must set this to true. The default value is false.

ResultsOnServer – Boolean - true to store the results on the server, false otherwise – default value false.

In WS Pro:

The first and only parameter must be a hash.

The keys are as follows:

RunOn – string - the name of the machine the simulation may run on, or one of '.' or '*'. '.' has the meaning of local machine and '*' has the meaning of any available server. – default value '*'

MaxConcurrentJobs – integer - the maximum number of concurrent jobs – default value 0.

MaxRuntimeSeconds – integer - the maximum runtime in seconds, default value 1 day.

ResultsOnServer – Boolean – true to store the results on the server, false otherwise – default value true.

Note that the default values for ResultsOnServer and Server are different between ICM and WS Pro.

status
n=sim.status

Returns the status of simulation, one of:

- "None"
- "Active"
- "Success"
- "Fail"

success_substatus
n=sim.success_substatus

If the simulation succeeded, returns its substatus, one of:

- "Incomplete"
- "Warnings"
- "OK"

timestep_count
n=sim.timestep_count

Returns the number of results timesteps for a simulation.

WSOpenNetwork

add_scenario (InfoAsset / ICM only)
on.add_scenario(name,based_on,notes)

e.g.

```
on.add_scenario 'My Scenario',nil,'This is my scenario'
```

The parameters are:

Name – name of new scenario (string)

Based_on – name of scenario (string) or nil for a scenario not based on another one

Notes – the notes for the scenario

clear_selection
on.clear_selection

Clears the selection

close (Exchange only)
on.close

Closes the WSOpenNetwork. It is not necessary to call this, but it will ensure that memory is freed at this point rather than when Ruby garbage collection occurs. Once a WSOpenNetwork has been closed, any objects within that network (e.g. of type WSRowObject) become invalid and any attempt to access methods for them will cause an exception to be thrown).

current_timestep (modelling products only)

n=on.current_timestep

The WSOpenNetwork object has a current timestep corresponding to the current timestep results have when opened in the software's UI. It determines the timestep for which the 'result' method of the WSRowObject returns its value. This method returns the index of the current timestep, with the first timestep being index 0 and the final timestep begin timestep_count – 1. The value of -1, representing the 'maximum' 'timestep' is also possible. The initial value when a sim is opened in ICM Exchange will be 0 if there are time varying results, otherwise -1 for the 'maximum' 'timestep'.

current_timestep_time (modelling products only)

t=on.current_timestep_time

Returns the time of the current timestep.

current_timestep= (Exchange only for modelling products)

on.current_timestep=7

This method sets the current timestep to the timestep with the index given e.g. 0 sets the current timestep to the first timestep, -1 sets it to the 'maximum' 'timestep'.

csv_export

nno.csv_export(filename,options)

Exports the WSOpenNetwork to the file specified

The options object must be a nil or a hash contain some or all of the following fields

Use Display Precision	If true, uses the display precision, otherwise outputs the number with more digits	True
Field Descriptions	If true, outputs a line of field descriptions (for the information of anyone reading the file)	False
Field Names	If true, outputs a header line with the field names – these are used when files are reimported	True
Flag Fields	If true, the flag fields are exported	True
Multiple Files	If true, the version control object is exported to multiple files, one for each network object type	False

Native System Types	If true, the internal names for system types are used	False
User Units	If true, the user selected units rather than the internal ones (SI) are used	True
Object Types	If true, the type of each object exported is exported in a separate column	False
Selection Only	If true, only the selected objects are exported	False
Units Text	If true, a line containing the units for each field is exported	False
Coordinate Arrays Format	The mechanism whereby arrays of coordinates (e.g. polylines and polygons) are exported. Choices are 'None', 'Packed' and 'Unpacked'	Packed
Other Arrays Format	The mechanism whereby other arrays (e.g. shapes / demand diagrams) are exported. Choices are 'None', 'Packed' and 'Separate'	Packed

csv_import

nno.csv_import(filename,options)

Updates the WSOpenNetwork from the CSV file specified.

The options parameter must be nil, or a hash which can contain any or all of the following.

Force Link Rename	If nodes are renamed, then associated objects have the node renamed	True
Flag Genuine Only	Only genuine changes to the field are flagged	False
Load Null Fields	If the field in an object to be updated is not blank and the field in the CSV file is blank, then if this is set to	True

	true, the data will be updated otherwise it won't	
Update With Any Flag	If this is set to true, then any field can be updated, if it is set to false then only fields with the flag set to the 'update flag' will be updated	True
Use Asset ID	Use the asset ID as the key for updates rather than the normal 'primary key'	False
User Units	Use the current 'user units' rather than the native (SI) units	True
Action	Action to perform Update Only – will only update existing objects Update And Add – will update and add objects, but not perform deletions Mixed – will add, update and delete objects depending on an 'action' field in the CSV file Delete – will delete objects	Mixed
Header	What header the importer will expect in the file(s) ID – ID only ID Description – Ids on one line, followed by descriptions on the next ID Units – Ids on one line, followed by units on the next ID Descriptions Units – Ids on one line, followed by descriptions on the next, followed by units on the 3rd	ID

New Flag	Flag used for new / updated fields	Blank
Update Flag	Flag used to determine what objects are updated if the 'Update With Any Flag' field is set to false	Blank

current_scenario

puts on.current_scenario

Returns the WSOpenNetwork object's current scenario as a string.

If the current scenario is the base scenario, returns 'Base' (in English).

current_scenario=

on.current_scenario=new_scenario

Sets the WSOpenNetwork's current scenario. The new value should be a string or can be nil in which case the current scenario is set to the base scenario. The new scenario must exist.

delete_scenario (ICM and InfoAsset Only)

on.delete_scenario(scenario_name)

e.g.

```
on.delete_scenario('myscenario')
```

Deletes the scenario specified in the string parameter. If the specified scenario is the current scenario, the current scenario is set to be the base scenario.

delete_selection

on.delete_selection

Deletes the current selection from the network in the current scenario.

delete_superfluous_dummy_nodes (WS Pro only)

delete_superfluous_dummy_nodes(tables,field)

Given an array of table names, which must be link tables (e.g. 'wn_pst') and the name of a text field, a user field being the most likely candidate for this will delete all objects in the node table tables where that field is set to Y and the object is not the upstream or downstream node in one of the tables in the array.

The aim of this method is to be used in conjunction with the open data import centre to support the situation where dummy nodes have been added at the ends of links (e.g. because they have been imported from a GIS where they are represented as point objects) but the links have since been deleted.

download_mesh_job_log (ICM Exchange only)

on.download_mesh_job_log(job_id,path)

Given one of the jobs listed in the return value of mesh_async, copies the log file created by that job to the path given.

each

on.each do |x|

Iterates through all the objects in the WSOpenNetwork e.g.

```
on.each do |x|
      puts x.id
end
```

each_selected

on.each_selected do |x|

As each but iterates through the selected objects only.

expand_short_links (WS Pro Exchange only)

on.expand_short_links(hash)

Expands short links in the same way as the UI.

The parameter is a hash which must exist.

The parameters correspond closely to those on the dialog in WS Pro.

The two float values will either be in user length units or system length units (m) depending on the global setting.

The array of table names are the internal names of the link types to which it applies e.g. wn_meter.

The 'Recalculate length' and 'Use user flag' Booleans correspond to the 3 choices on the dialog for flagging changed lengths – if you set the first to false the value of the second is ignored since lengths won't be changed there is not need to flag them.

Key	Type	Default
Expansion threshold	Float	1.0
Minimum resultant length	Float	1.0
Flag	String	
Protect connection points	Boolean	false
Recalculate length	Boolean	false
Use user flag	Boolean	false
Tables	Array of strings	empty array

e.g.

```
db=WSApplication.open
net=db.model_object_from_type_and_id('Geometry',276)
on=net.open
on.run_SQL('_links','1')
myHash=Hash.new
#myHash['badger']='duck'
myHash['Expansion threshold']=1.0
myHash['Minimum resultant length']=1.0
myHash['Recalculate Length']=true
myHash['Use user flag']=true
myHash['Tables']=['wn_meter']
myHash['Flag']='SL'
on.expand_short_links(myHash)
```

export_IDs

on.export_IDs(filename,parameters)

Exports the IDs of all the objects or the current selection to a file, grouped by table.

The parameters must be nil or a hash. If the parameters are a hash then there are two possible values:

Selection Only – if set to true only the IDs of the current selection will be exported (default false i.e. all exported)

UTF8 – if set to true the file will be UTF8 encoded (default false i.e. current locals)

field_names(type)

arr=on.field_names(type)

e.g.

```
arr=on.field_names('hw_node')
arr.each do |f|
      puts f
end
```

Given the internal name of a type, returns a list of names of the fields for that object type – the object type cannot be a class.

gauge_timestep_count (ICM Only)

n=on.gauge_timestep_count

This method returns the number of gauge timesteps. If there are no gauge timesteps, either because no objects are 'gauged' or because the gauge timestep for the run was set to 0 it returns 0.

gauge_timestep_time (ICM Only)

t=on.gauge_timestep_time(timestep_no)

This method returns the time for the gague timestep with the index given as the method's parameter.

GIS_export

on.GIS_export(format,params,location)

Exports the network data to GIS.

See the method of the same name for WSNumbatNetworkObject for an explanation of the parameters.

InfoDrainage_import

on.InfoDrainage_import(filename, logFile)

Imports an InfoDrainage model into the network.

Parameters:

filepath - full path to the InfoDrainage iddx file.

logfile - full filepath where the import log is written. The log is in rich text format.

list_gauge_timesteps (ICM only)

arr=on.list_gauge_timesteps

This method returns an array containing the times of all the gauge timesteps for the sim in order.

list_GIS_export_tables

arr=on.list_GIS_export_tables

This method lists the tables that will be exported to GIS as an array of strings. This method is designed to return information useful when customising the parameters to GIS_export.

list_timesteps

arr=on.list_timesteps

This method returns an array containing the times of all the timesteps for the sim in order.

load_mesh_job (ICM Exchange only)

on.load_mesh_job(job_id)

Given one of the jobs listed in the return value of mesh_async, loads the completed mesh into the network.

load_selection

on.load_selection(selection_list)

Selects objects based on the selection list object. The parameter may be a WSModelObject, an object path or the numerical ID of the model object – providing in all cases that the model object is a selection list.

mesh (ICM Exchange Only)

myHash=on.mesh(params)

This method meshes one or more 2D zones.

The parameter is a hash, containing the following keys:

GroundModel – the ground model

VoidsFile –GIS file containing voids

VoidsFeatureClass – for GeoDatabases, the feature class within the GeoDatabase

VoidsCategory – the voids polygon category

BreakLinesFile –GIS file containing break lines

BreakLinesFeatureClass - for GeoDatabases, the feature class within the GeoDatabase

BreakLinesCategory – the break lines polyline category

WallsFile –GIS file containing walls

WallsFeatureClass - for GeoDatabases, the feature class within the GeoDatabase

WallsCategory – the walls polyline category

2DZones - one or more 2D zones

2DZonesSelectionList – a selection list of 2D zones

LowerElementGroundLevels –whether or not the process will lower 2D mesh elements with ground levels higher than the adjacent bank levels.

RunOn – the computer to run the jobs on

LogFile – the path of the log file

LogPath – the path of a folder for the log files

The values associated with the keys are as follows:

GroundModel
This may either be the scripting path, the ID or the WSModelObject representing a ground model (either grid or TIN). If the ID is negative then it represents a TIN ground model i.e. -7 represents the TIN ground model with ID 7. If the ID is positive it represents a gridded ground model.
This parameter is required.

VoidsFile
The path of a GIS file containing the voids - String – ignored if empty

VoidsCategory
The category of polygon within the network used for voids – string (ignored if empty)

VoidsFeatureClass
For a GeoDatabase, the feature class within the GeoDatabase for the voids

BreakLinesFile
The path of a GIS file containing the break lines - String – ignored if empty

BreakLinesCategory
The category of polyline within the network used for break lines – string (ignored if empty)

BreakLinesFeatureClass
For a GeoDatabase, the feature class within the GeoDatabase for the break lines

WallsFile
The path of a GIS file containing the walls - String – ignored if empty

WallsCategory
The category of polyline within the network used for walls – string (ignored if empty)

WallsFeatureClass
For a GeoDatabase, the feature class within the GeoDatabase for the walls

2DZones
If the 2DZonesSelectionList parameter is absent and this parameter is absent or nil all 2D zones will be meshed. Otherwise can contain
b) the name of a 2D zone as a string
c) an array of strings containing the names of 2D zones

2DZonesSelectionList
A selection list of 2D zones to mesh – must be a selection list as a numerical ID, a scripting path or a WSModelObject.

LowereElementGroundLevel
If present and evaluates to true, the process will lower 2D mesh elements with ground levels higher than the adjacent bank levels

RunOn
The computer to run the job on – '.' for 'this computer', '*' for any computer

LogFile
The path of the log file (an HTML file) is blank one will not be produced (well, actually one is produced anyway, but it won't be copied to this path). This can only be used if only one 2D zone is meshed.

LogPath
The path of a folder for the log files. This may be used however many 2D zones are meshed. The file will be given the name of the 2D zone with the file type HTML.

For the pairs of keys (voids, break lines and wall) only one of the two values may be set.

If any of the VoidsFile, WallsFile or BreakLinesFile values are set, i.e. if any voids, walls or break lines are to be read in from a GIS files, the GIS component must be set with WSApplication.map_component= The user must have the GIS component they are selecting.

The FeatureClass keys can only be set if the corresponding File key is set and the map control is set and is not MapXTreme.

Only one of the 2DZones and 2DZonesSelectionList keys may be present.

Only one of the LogFile and LogDir keys may be present.

If there are no 2D zones in it this will be treated as an error.

This method performs the meshing synchronously i.e. it does the meshing and then returns.

This method returns a hash from the name of the 2D zone to a Boolean indicating success or failure.

mesh_async (ICM Exchange only)

arr=on.mesh_async(params)

This method is the same as 'mesh' except that it sets the meshing off to run asynchronously.

It does not have the LogFile and LogDir keys

It returns an array of job IDs which may be used in the load_mesh_job, cancel_mesh_job, download_mesh_job_log and mesh_job_status methods, as well as the WSApplication.wait_for_jobs method.

mesh_job_status (ICM Exchange only)

a=mesh_job_status(job_id)

Given one of the jobs listed in the return value of mesh_async, returns the job's current status as a string.

model_object

mo=on.model_object

Returns a WSModelObject (or derived class) associated with the WSOpenNetwork. If a sim was opened to obtain the WSOpenNetwork, the model object of that sim will be returned.

mscc_export_cctv_surveys (InfoAsset only)

on.mscc_export_cctv_surveys(export_file, export_images, selection_only, log_file)

e.g.

```
success = on.mscc_export_cctv_surveys('D:\output.xml', true, false,
'D:\log.txt')
```

This method exports CCTV survey data from a Collection Network to the MSCC4 XML format. The *export_file* argument specified the output XML file and *log_file* the location of a text file for errors. The other two arguments take Boolean values. *export_images* controls whether defect images are to be exported and *selection_only* will limit the export to selected objects. The function returns *true* if successful.

mscc_export_manhole_survey (InfoAsset only)

on.mscc_export_manhole_surveys(export_file, export_images, selection_only, log_file)

e.g.

```
success = on.mscc_export_manhole_surveys('D:\output.xml', true,
false, 'D:\log.txt')
```

This method exports manhole survey data from a Collection Network to the MSCC5 XML format. The *export_file* argument specified the output XML file and *log_file* the location of a text file for errors. The other two arguments take Boolean values. *export_images* controls whether defect images are to be exported and *selection_only* will limit the export to selected objects. The function returns *true* if successful.

mscc_import_cctv_surveys (InfoAsset only)

on.mscc_import_cctv_surveys(import_file, import_flag, import_images, id_gen, overwrite, log_file)

e.g.

```
success = on.mscc_import_cctv_surveys('D:\import.xml', 'IM', true,
2, false, 'D:\log.txt')
```

This method imports CCTV survey data into a Collection Network from the MSCC4 XML format. The *import_file* argument specifies the XML file and *log_file* the location of a text file for errors. The *import_flag* text specifies the data flag for imported fields. *import_images* controls whether defect images are to be imported. To prevent the overwriting of existing surveys in the event of name clashes, set *overwite* to false. The id generation parameter, *id_gen*, uses the following values (these correspond to the user interface options in the help).

1 – StartNodeRef, Direction, Date and Time

2 – StartNodeRef, Direction and an index for uniqueness

3 – US node ID, Direction, Date and Time

4 – US node ID, Direction and an index for uniqueness

5 – ClientDefined1

6 – ClientDefined2

7 – ClientDefined3

mscc_import_manhole_surveys (InfoAsset Only)

on.mscc_import_manhole_surveys(import_file, import_flag, import_images, id_gen, overwrite, log_file)

e.g.

```
success = on.mscc_import_manhole_surveys('D:\import.xml', 'IM',
true, 2, false, 'D:\log.txt')
```

This method imports manhole survey data into a Collection Network from the MSCC5 XML format. The *import_file* argument specifies the XML file and *log_file* the location of a text file for errors. The *import_flag* text specifies the data flag for imported fields. *import_images* controls whether defect images are to be imported. To prevent the overwriting of existing surveys in the event of name clashes, set *overwite* to false. The id generation parameter, *id_gen*, uses the following values (these correspond to the user interface options in the help).

1 – Manhole/Node reference, Date and Time

2 – Manhole/Node reference and an index for uniqueness

new_row_object

ro=on.new_row_object(type)

e.g.

```
ro=on.new_row_object('hw_node')
```

This method creates a new row object in the network. This must be done within a transaction.

network_model_object

mo=on.network_model_object

Returns a WSModelObject (or derived class) associated with the WSOpenNetwork. If a sim was opened to obtain the WSOpenNetwork, the model object of the network will still be returned.

objects_in_polygon

vec=on.objects_in_polygon(polygon,type_or_types)

Returns a Ruby array of objects in a polygon. This is essentially the same as the WSRowObject version except the polygon is passed in as the first parmeter. The purpose of this method is to allow the polygon to be in one network and the objects found to be in another e.g. the current network and the background network.

odic_import_ex

vec=on.odic_import_ex(format,config_file,params,...)

As for WSNetworkObject / WSNumbatNetworkObject above except that it returns an array of WSRowObject objects, one for each object created or updated by the import (this would not be possible in the WSNetworkObject / WSNumbatNetworkObject version, since the network is not left open after the method, and WSRowObject objects may not be open when their network is not).

odec_export_ex

on.odec_export_ex(format,config_file,params,...)

As for WSNetworkObject / WSNumbatNetworkObject above.

ribx_export_surveys (InfoAsset only)

on. ribx_export_surveys (export_file, selection_only, log_file)

e.g.

```
success = on.ribx_export_surveys('D:\output.xml',false,'D:\log.txt')
```

This method exports manhole survey and cctv survey data from a Collection Network to the RIBX XML format. The *export_file* argument specified the output XML file and *log_file* the location of a text file for errors. The *selection_only* argument is a Boolean value and will limit the export to selected objects if it is *true*. The function returns *true* if successful.

ribx_import_surveys (InfoAsset only)

on.ribx_import_surveys(import_file, import_flag, id_gen, overwrite, log_file)

e.g.

```
success = on.ribx_import_surveys('D:\import.xml', 'IM', true, 2,
false, 'D:\log.txt')
```

This method imports CCTV survey & manhole survey data into a Collection Network from the RIBX XML format. The *import_file* argument specifies the XML file and *log_file* the location of a text file for errors. The *import_flag* text specifies the data flag for imported fields. To prevent the overwriting of existing surveys in the event of name clashes, set *overwite* to false. The id generation parameter, *id_gen*, uses the following values (these correspond to the user interface options in the help).

1 – StartNodeRef, Direction, Date and Time

2 – StartNodeRef, Direction and an index for uniqueness

3 – US node ID, Direction, Date and Time

4 – US node ID, Direction and an index for uniqueness

row_object

ro=on.row_object(table,id)

e.g.

```
ro=on.row_object('hw_node','MH111111')
```

Given a table name and an ID (with the parts separated by . if it is a type with multiple key fields) returns a WSRowObject or derived class. Returns nil if there is no such object in the network.

row_objects

arr=on.row_objects(table)

e.g.

```
vec=on.row_objects('hw_node')
```

Given a table name returns a vector of WSRowObject (possibly including instances of derived classes). Returns a vector of zero length if there are no objects of that type in the network.

row_objects_selection

arr=on.row_objects_selection('hw_node')

e.g.

```
vec=on.row_objects_selection('hw_node')
```

As row_objects but only returns objects currently selected in the WSOpenNetwork

row_objects_from_asset_id

arr=on.row_objects_from_asset_id(type,asset_id)

Given an asset ID and a type returns a vector of WSRowObject (possibly including instances of derived classes) with that asset ID. Returns a vector of zero length if there are no objects of that type in the network.

row_object_collection

roc=on.row_object_collection(type)

Given a type returns a WSRowObjectCollection (see below) representing a collection of objects in the network which can be iterated through. The parameter can be the type or class of the object, or nil in which case all objects will be returned.

row_object_collection_selection

roc=on.row_object_collection_selection(type)

As row_object_collection above, but only returns objects currently selected in the WSOpenNetwork.

run_inference

on.run_inference(inference,ground_model, mode, zone_or_zone_category, error_file)

Runs the inference object on the network, which must be a collection asset network or a distribution asset network.

The parameters are as follows:

Inference – the inference to be run –may be a WSModelObject, an object path or the numerical ID of the model object. This must be of the appropriate inference object type for the network.

Ground_model – optional ground model (used for inferring heigh). May be a WSModelObject, an object path or the numerical ID of the model object. The WSModelObject or path may be of a TIN ground model or a grid ground model, the numerical ID may only be that of a grid ground model i.e. to use a TIN ground model you must use a WSModelObject or path.

Mode – the mode of the inference, corresponding to the options on the dialog shown when inference is run in the user interface. The permitted values are as follows:

- nil, false or the string 'Network' – run the inference on the whole network
- true or the string 'Selection' – run the inference on the current selection (which, of course, must be set up within the script)
- the string 'Zone' – run the inference for the zone specified in the following parameter.
- The string 'Category' – run the inference for zones with the zone specified in the following parameter.

Zone_or_zone_category – a string representing the zone or zone category. This parameter is only used if the previous parameter is 'Zone' or 'Category'.

Error_file – if this is a string, then error messages are written to the file named (which is deleted prior to the method being run)

When run within the UI, the ground model parameter must be nil. If there is a ground model loaded into the network (either TIN or grid), it will be used instead.

run_SQL

on.run_SQL(object,query)

This method runs the SQL query with the given table being the 'current table' as it appears in the SQL dialog i.e. the default table if the SQL query does not further qualify the table name.

The names '_nodes' or '_links' can be used to run the SQL over all node or link tables in the same way as can be achieved by selecting 'All nodes' or 'All links' in the dropdowns in the SQL dialog.

run_stored_query_object (ICM / InfoAsset only)

on.run_stored_query_object(stored_query_object)

Runs a stored query object. The parameter is the stored query object to be run –may be a WSModelObject, an object path or the numerical ID of the model object.

From version 6.5, when run from the UI, spatial queries and those with UI may be run from the UI. This was not the case with previous versions and is not the case when they are run from the Exchange products.

save_selection

on.save_selection(selection_list)

Saves the current selection (in the current scenario) to an already existing selection list model object. The parameter may be a WSModelObject, an object path or the numerical ID of the model object – providing in all cases that the model object is a selection list.

e.g.

```
mosl=db.model_object_from_type_and_id 'Selection List',14
mosl2=db.model_object_from_type_and_id 'Selection List',15
on.save_selection mosl.path
on.save_selection mosl2
on.save_selection 16
```

e.g.

```
on.save_selection mySelectionList.path
```

Saves the current selection to the already existing selection list.

scenarios (ICM / InfoAsset only)

on.scenarios.each do |s|

e.g.

```
on.scenarios do |s|
      puts s
end
```

This method provides a block of names of scenarios to be iterated through. The base scenario is included in the results as the string 'Base', in English.

search_at_point(x,y,distance,types)

roc=on.search_at_point(x,y,distance,types)

Finds the objects within a distance of a given point, returning a vector of WSRowObject. Types may be nil (in which case all tables are searched), a string or an array of strings, these strings may be the name of a type or a category. It may not contain duplicates, and may not contain a category and a table within that category.

selection_size

n=on.selection_size

Returns the number of objects selected.

snapshot_import

on.snapshot_import(filename) (ICM / InfoAsset)
on.snapshort_import(filename,parameters_hash) (WS Pro)

For ICM and InfoAsset Manager, this Imports a snapshot file into the network from the given filename. This has the same effect as calling snapshot_import_ex with the second parameter being nil

For WS Pro this takes a second argument which is a parameters hash.

snapshot_import_ex (ICM and InfoAsset Only)

on.snapshot_import_ex(filename,parameters_hash)

Imports a snapshot file into the network from the given filename. The second parameter must be a hash from strings to values as shown below or nil. If it is nil then the defaults will be used.

Valid keys for the hash are as follows:

Tables – Array of strings - If present, a list of the internal table names (as returned by the table_names method of this class) If not present then all tables will be exported.

AllowDeletes - Boolean

ImportGeoPlanPropertiesAndThemes - Boolean

UpdateExistingObjectsFoundByID - Boolean

UpdateExistingObjectsFoundByUID – Boolean

ImportImageFiles - Boolean

snapshot_export (ICM / InfoAsset only)

on.snapshot_export(filename)

Exports a snapshot of the network to the given filename.

All objects are exported from all tables, but image files and GeoPlan properties and themes are not exported.

The method snapshot_export_ex allows more flexibility over what is exported.

Snapshots cannot be exported from networks with uncommitted changes.

snapshot_export_ex (ICM InfoAsset Only)

on.snapshot_ex(filename,parameters_hash)

Exports a snapshot of the network to the given filename. The second parameter must either be nil or a hash from strings to values as follows:

SelectedOnly – (Boolean). If present and true, only the currently selected objects are exported, otherwise by default all objects of the appropriate tables are exported.

IncludeImageFiles – (Boolean).If present and true, includes the data for image files in the network, otherwise by default images are not exported.

IncludeGeoPlanPropertiesAndThemes – (Boolean) If present and true, includes the data for GeoPlan properties and themes, otherwise by default they are not exported

ChangesFromVersion – (integer) If present, the snapshot will be of the different from the network's version with this commit ID, otherwise by default the current version of the network will be exported.

Tables – (array of strings) If present, a list of the internal table names (as returned by the table_names method of this class) If not present then all tables will be exported.

The SelectedOnlyOptions must not be mixed with the Tables option or the ChangesFromVersion option.

Any other keys for the hash are treated as an error and an exception raised.

If the second parameter is nil then the default are used i.e. all objects from all tables are exported but image files and GeoPlan properties and themes are not, yielding the same result as calling snapshot_export.

Snapshots cannot be exported from networks with uncommitted changes.

snapshot_scan

details_hash=on.snapshot_scan(filename)

Given a snapshot file, this method scans it and returns a hash containing the details as follows. The keys are all strings.

NetworkGUID – string – the GUID of the network from which the snapshot was exported.

CommitGUID – string – the GUID of the commit of the network from which the snapshot was exported.

CommitID – integer – the ID of the commit of the network from which the snapshot was exported.

NetworkTypeCode – string – the type of network from which the snapshot was exported. This matches the name of the network type e.g. 'Collection Network'

DatabaseGUID – string – the GUID associated with the database version from which the snapshot was exported.

DatabaseSubVersion – integer – the 'subversion' associated with the database version from which the snapshot was exported.

UnknownTableCount – integer – the number of tables in the snapshot not recognised by the software, this will only be greater than 0 if the snapshot were exported from a more recent version of the software.

FileCount – integer –the number of image files contained within the snapshot.

ContainsGeoPlanPropertiesAndThemes – Boolean – true if the snapshot was exported with the option to included GeoPlan properties and themes.

Tables – hash – a hash containing information about the tables exported as described below.

The Tables hash is a hash from the table names (strings) to a hash containing information about each table.

The hash of information about each table is as follows. The keys are all strings and the values are all integers.

ObjectCount – the number of objects in the snapshot for the table.

ObjectsWithOldVersionsCount -

ObjectsFoundByUID -

ObjectsFoundByID -

DeleteRecordsCount -

UnknownFieldCount –the number of unknown fields for the table, this will be zero unless the export is from a more recent version of the software than the user is using to import the data.

table_names

arr=on.table_names

e.g.

```
on.table_names.each do |n|
      puts n
end
```

This method returns an array of the tables names for the WSOpenNetwork – these are the internal names.

table

t=on.table(name)

Given a table name, this method returns a WSTableInfo object for that table.

tables

arr=on.tables

This method returns an array of WSTableInfo objects for the WSOpenNetwork.

timestep_count
n=on.timestep_count

This method returns the number of timesteps in the results for the sim, it does not include the 'maximum' 'timestep' so for sims without any time varying results this method will return 0.

timestep_time
t=on.timestep_time(timestep_no)

This method returns the time for the timestep with the index given as the method's parameter.

transaction_begin
on.transaction_begin

Changes to objects in the network should in general be included within transactions. A transaction begins with transaction_begin, then may be ended either with transaction_commit, which commits the changes since transaction_begin, or with transaction_rollback, which cancels the changes since transaction_begin

transaction_commit
on.transaction_commit

Commits the transaction – see above.

transaction_rollback
on.transaction_end

Ends the transaction – see above.

update_cctv_scores
on.update_cctv_scores

Calculates CCTV scores for all surveys in the network using the current standard.

validate
v=on.validate(scenarios)

This method validates the scenario or scenarios given in the parameter, returning a WSValidation object.

The parameter may be nil, in which case the Base scenario is validated, a string in which case the named scenario (which may be 'Base') will be validated, or an array of strings, in which case all the named scenarios (which may include 'Base') will be validated.

XPRAFTS_import (ICM Exchange only)
on.XPRAFTS_import(filepath,useLargeSize,splitOnLagLinks,combineSubcat,logFilepath)

Updates the WSOpenNetwork from the XPRAFTS xpx file specified.

filepath refers to the xpx file exported from XPRAFTS.

Set useLargeSize if the XPRAFTS model is configured to use the Large unit size.

Set splitOnLagLinks to true if you want to split networks down stream of channel links and maintain lag link data, otherwise set to false to maintain network connectivity by converting downstream lag links to channel links.

Set combineSubcat to true to combine the 1st and 2nd subcatchment as a single subcatchment polygon. This would set the Per-surface RAFTS B option and setting the Rafts adapt factor and Manning's roughness at the runoff surface level.

e.g.

```
model_group.XPRAFTS_import('d:\\temp\\1.xpx',true,
false,true,'d:\\temp\\log.txt')
```

WSRowObject

All the methods of this class can be used in the UIs and in all three Exchange products.

[]
[]=

```
ro['field']=value
v=ro['field']
ro.field_name = value
v=ro.field_name
ro['_tag']=value
v=ro['_tag']
ro._tag = value
v=ro._tag
```

The above are used to set and get values of fields for WSRowObject objects as described in the 'getting and setting values' section above.

category
s=ro.category

Returns the category name (as described above) of the object's table as a string.

contains?
b=ro.contains?(ro2)

Returns true if polygon ro contains the object ro2.

delete
ro.delete

Deletes the row object.

field
f=ro.field(fieldname)

Returns the WSFieldInfo object associated with the named field in the WSRowObject's table. This returns information about the named field, which does not depend on the actual WSRowObject or its value for the field – it is essentially a short cut to getting the table and then getting the field info from that.

gauge_results

arr=ro.gauge_results(results_field_name)

Returns an array of the results for the given results field name for the object at all gauge timesteps – the field must have time varying results. If the object or field does not have gauge results it will return the 'normal' results.

id

id=ro.id

Returns the ID of the object as a string. If the object has a multi-part primary key, then the key will be output with part separated by the '.' character.

id=

ro.id=newid

Updates the ID of the row object.

is_inside?

b=ro.is_inside?(ro2)

Returns true if ro2 is inside the polygon ro2.

navigate
navigate1

vec=ro.navigate(navigation_type)
ro2=ro.navigate(navigation_type)

The navigate and navigate1 methods are used to navigate between objects and other objects to which they are physically related.

The navigate method may be used to navigate using one-to-one and one-to-many links and returns an array of WSRowObject objects. The navigate1 method may only be used to navigate using one-to-one links and returns a WSRowObject or nil if there is no object related to the object by that particular navigation type.

The navigation types are the same as those used within the SQL in the software as follows:

joined	No
us_node	No
ds_node	No
custom	No
node	No
lateral_pipe	No
pipe	No
us_links	No
ds_links	No

sanitary_manhole	No
storm_manhole	No
sanitary_pipe	No
storm_pipe	No
property	No
data_logger	No
smoke_test	No
drain_tests	Yes
manhole_surveys	Yes
manhole_repairs	Yes
gps_surveys	Yes
incidents	Yes
monitoring_surveys	Yes
pipe_repairs	Yes
smoke_tests	Yes
dye_tests	Yes
cctv_surveys	Yes
properties	Yes
smoke_defects	Yes
pipe_samples	Yes
joined_pipes	Yes
pipe_cleans	Yes
maintenance_records	Yes
hydrant_tests	Yes
meter_tests	Yes
meters	Yes
all_us_links	No
all_ds_links	No
general_maintenance_records	Yes

tasks	No
resources	No
manhole	No
valve	No
hydrant	No
meter	No
materials	No
orders	No
connection_pipe	No
asset_name_groups	No
asset	No
leak_detections	No
general_surveys	No
fog_inspections	No

objects_in_polygon

vec=ro.objects_in_polygon(type_or_types)

Returns a vector of the objects in the polygon (from the same network). The type or types parameter may be nil (in which case all tables are searched), a string or an array of strings, these strings may be the name of a type or a category. It may not contain duplicates, and may not contain a category and a table within that category. This is the same as the similar parameter in the WSNumbatNetworkObject's search_at_point method.

result

f=ro.result(result_fields_name)

Returns the result for the given results field name for the object at the current timestep.

results

arr=ro.results(result_field_name)

Returns an array of the results for the given results field name for the object at all timesteps – the field must have time varying results.

selected?

b=ro.selected?

returns True if the object is currently selected, False otherwise

selected=

ro.selected=b

e.g.

```
ro.selected=true

ro2.selected=false
```

If the value on the right of the = evaluates to true, the object is selected, otherwise it is deselected.

table

s=ro.table

Returns the internal name of the object's table as a string.

table_info

ti=ro.table_info

Returns the WSTableInfo object for the object's table, this may be used to get the list of fields for the object.

write

ro.write

After changing field values it is necessary to call the write method, otherwise the changes will not take effect.

WSNode

This class is derived from the WSRowObject class, and represents objects of 'category' node as described above. As with the WSRowObject class, all the methods of this class can be used in the UIs and in all three Exchange products. It has two extra methods:

us_links

roc=node.us_links

Returns a WSRowObjectCollection (see below) representing all the upstream links of the node. If there are no upstream links a collection of length zero is returned.

ds_links

roc=node.ds_links

Returns a WSRowObjectCollection (see below) representing all the downstream links of the node. If there are no upstream links a collection of length zero is returned.

WSLink

This class is derived from the WSRowObject class, and represents objects of 'category' link as described above. As with the WSRowObject class, all the methods of this class can be used in the UIs and in all three Exchange products. It has two extra methods:

us_node

ro=link.us_node

Returns a WSRowObject representing the link's upstream node, or nil if it doesn't have one.

ds_node

ro=link.ds_node

Returns a WSRowObject representing the link's downstream node, or nil if it doesn't have one.

WSRiskAnalysisRunObject

This class is only applicable to ICM and its method can only be used in ICM Exchange.

run

raro.run

Performs the risk analysis run.

WSRowObjectCollection

All the methods of this class can be used in the UIs and in all three Exchange products.

length

n=roc.length

Returns the number of WSRowObjects in the collection

[]

ro=roc[n]

Returns the nth WSRowObject in the collection. The index is 0 based i.e. valid values are from 0 to length-1.

each

roc.each{|ro|}

Allows the user to iterate through all objects in the collection e.g.

```
my_network.row_object_collection('_nodes').each do |ro|
      puts  ro.id
end
```

WSTableInfo

All the methods of this class can be used in the UIs and in all three Exchange products.

description

s=ti.description

returns the description of the table.

fields
arr=ti.fields

returns an array of the fields for this table – the fields are of type WSFieldInfo

- flags are treated as separate fields.

name
s=ti.name

returns the internal name of the table.

results_fields
arr=ti.results_fields

Returns an array of results fields as instances of the WSFieldInfo class.

Note that the fields returned and the values of their has_time_varying_results? and has_max_results? methods reflects the results of the specific simulation run you are looking at the results of, the details of which can vary considerably depending on the options selected when performing the run. This is different from the fields method which returns the fields for the network which do not on the whole change for a network type for a particular release of the software, except for user defined fields and tables in InfoAsset.

WSFieldInfo

All the methods of this class can be used in the UIs and in all three Exchange products.

data_type
s=fi.data_type

Returns the data type of the field as a string. The method is named 'data_type' to distinguish it from the Ruby type. The data types are described in InfoWorks terms, not Ruby data types. The types are as follows:

- Flag
- Boolean
- Single
- Double
- Short
- Long
- Date
- String
- Array:Long
- Array:Double
- WSStructurei
- GUID

Flag fields have their type returned as 'Flag', their underlying Ruby type is the Ruby string.

description
s=fi.description

returns the description of the table

fields

arr=fi.fields

returns an array of fields if the field is itself a 'structure blob'. If the field is not a structure blob, this method returns nil.

has_time_varying_results?

b=fi.has_time_varying_results?

This method returns true if the field has time varying results i.e. results for the timesteps in the simulation. It will return false for network fields. Please see the WSTableInfo results_fields method for more discussion.

has_max_results?

b=fi.has_max_results?

This method returns true if the field has a maximum / summary result i.e. if it has a maximum / summary result displayed when the simulation is open in the UI and the timestep control is set to the 'maximum' 'timestep'. It will return false for network fields. Please see the WSTableInfo results_fields method for more discussion.

name

s=fi.name

returns the name of the field

read_only?

bReadOnly=fi.read_only?

Returns true if the field is read only, false if it isn't

size (from version 7.0)

n=fi.size

Returns 4 if the field is a flag field, the length of a string field for string fields or 0 otherwise.

WSCommit

This class is only applicable to ICM and InfoAsset. The methods may only be used in ICM Exchange and InfoAsset Exchange.

The methods of this class are all read only and each of them returns the value in one of the fields that appears in the commit grid as follows:

branch_id

comment

commit_id

date

deleted_count

inserted_count

modified_count

setting_changed_count

user

WSCommits

This class is only applicable to ICM and InfoAsset. The methods may only be used in ICM Exchange and InfoAsset Exchange.

This class is a collection class and thus has the [], each and length methods defined. Each individual object in the collection is an instance of the WSCommit class.

WSStructure

All the methods of this class can be used in the UIs and in all three Exchange products.

each

sb.each {|v|}

Iterates through the collection.

length

n=sb.length

Returns the number of WSStructureRow objects in the WSStructure collection

length=

sb.length=n

Sets the number of WSStructureRow objects in the WSStructure collection

size

n=sb.size

Returns the number of WSStructureRow objects in the WSStructure collection (synonym for length)

size=

sb.size=n

Sets the number of WSStructureRow objects in the WSStructure collection (synonym for length)

write

sb.write

Causes the changes to the WSStructure, i.e. changes to its length, data in any new rows, and changes to data in existing rows to be 'written back' to the WSRowObject. If this is not called then changes to the WSStructure will not take effect.

Once any changes to the structure blob have been written back to the WSRowObject, the WSRowObject must be written to the database with its write method (and then, of course, the transaction must be committed and the network written to the database with the commit method or in the user interface!).

[]
v=sb[n]

Returns the nth WSStructureRow object in the collection (zero based).

WSStructureRow

All the methods of this class can be used in the UIs and in all three Exchange products.

[]
v=row[val]

Returns the value of the named field in the WSStructureRow object.

[]=
row[val]=v

Sets the value of the named field in the WSStructureRow object.

WSValidations

These methods are relevant to ICM Exchange and WS Pro Exchange

error_count
n=vals.error_count

Returns the number of errors found when performing the validation.

warning_count
n=vals.warning_count

Returns the number of warnings found when performing the validation.

length
n=vals.length

Returns the number of WSValidation objects in the WSValidations collection

each
n.each {|v|}

Iterates through the collection.

[]
v=vals[n]

Returns the nth WSValidation object in the collection (zero based).

WSValidation

The methods of this class are are all read only and each of them returns the value in one of the fields that appears in the validation windows when the network is validated within the user interface.

These methods are relevant to ICM Exchange and WS Pro Exchange

code
n=v.code

Returns the code of the validation message.

field
s=v.field

Returns the field from the validation message. This may not be a real database field, but if it is then the internal name rather than the description will be returned.

field_description
s=v.field_description

Returns the field column from the validation message as it appears in the validation window.

object_id
s=v.object_id

Returns ID for the object in the validation message, if any.

object_type
s=v.object_type

Returns the description from the object type column of the validation message, if any.

message
s=v.object_message

Returns the validation message.

priority
n=v.priority

Returns the priority of the validation message.

type
n=v.type

Returns the type of the validation message: 'error', 'warning' or 'information'.

scenario
s=v.scenario

Returns the scenario name for the validation message – returns 'Base' for the base scenario.

(ICM Exchange only)

WSRunScheduler
This class is used to set up and modify WS Pro runs. A number of run types are supported – see Appendix 4

new

rs=WSRunScheduler.new

Creates a new WSRunScheduler object.

set_parameters

rs.set_parameters(parameter_hash)

Sets run parameters using values from the hash parameter_hash

load

b=rs.load(run_id)

Loads the parameters from run with ID run ID (must be an integer). Returns true if successful, false otherwise.

create_new_run

b=rs.create_new_run(group_id)

Creates a new run in the specified run group using the currently-set parameters. Returns true if successful.

save

b=rs.save(autorename)

Saves current parameters to a previously-loaded run. May create a new (renamed) run if autoRename is true and the loaded run is readonly. Returns true if successful. Throws an exception if autoRename is false and the loaded run is readonly.

validate

b=rs.validate(filename)

Validates the run parameters saving any validation errors to the specified file. Returns true if validated successfully with no errors.

get_run_mo

mo=rs.get_run_mo

Returns a WSModelObject that is the run that was associated with the WSRunScheduler by the most recent call to any of load, create_new_run or save

WSRun

run

run.run

Performs the run. Note than only normal runs are current supported.

release

run.release

Can be called after the run has finished. Removes the run from the run queue.

WSSWMMRunBuilder

This class is used to set up and modify SWMM runs in ICM Exchange

new

rb= WSSWMMRunBuilder.new

Creates a new WSSWMMRunBuilder object.

list_parameters

arr=rb.list_parameters

This method returns a list of run parameters (which will be the same for all runs). This is for convenience when used in conjunction with [] and []=

set_parameters

rb.set_parameters(parameter_hash)

Sets run parameters using values from the hash parameter_hash

load

b=rb.load(run)

Loads the parameters from the run specified which as usual can be an ID, a path or a WSModelObject.. Returns true if successful, false otherwise.

create_new_run

b=rb.create_new_run(group_id)

Creates a new run in the specified run group using the currently-set parameters. Returns true if successful.

validate

b=rb.validate(filename)

Validates the run parameters saving any validation errors to the specified file. Returns true if validated successfully with no errors.

get_run_mo

mo=rb.get_run_mo

Returns a WSModelObject that is the run that was associated with the WSSWMMRunBuilder by the most recent call to any of load, create_new_run or save

[] / []=

rb['key']=value
puts rb['key'].to_s

Sets and gets run parameters.

Appendix 1 – Pollutograph codes

List of pollutograph codes (relevant only for ICM).

P2D
P2A
P1D
P1A
NHD
COD
COA
PH_
SAL
NO3
NO2
DO_
COL
TW_
BOD
TPD
TPA
TKD
TKA
SF2
SF1
P4D
P4A
P3D
P3A
BOA

Appendix 2 – ICM InfoWorks Run Parameters

The fields that are set on the run dialog in the user interface are, in ICM Exchange, set as key value pairs within the hash passed in as the 6th parameter of the new_run method, called on the asset group in which the run is to be created.

The keys of the hash are all strings, the values are of a number of different types as described below.

Where the values have units, they must always be specified in S.I. units.

The run parameters used for ICM Exchange broadly speaking correspond to those set in the user interface. The list below therefore includes the field's location in the run dialog and its sub-dialogs, and its description in the user interface if the difference is noteworthy.

Consult InfoWorks ICM's main help for more details.

The behaviour of unspecified values for run parameters is as follows: When the run is created a number of the run parameters are supplied with default values. This means that were you to create a run, passing an empty hash in as the last parameter, and then to look at the values for the parameters using the [] method of the run object i.e.

```
db.list_read_write_run_fields.each do |p|
      if !run[p].nil?
            puts "#{p} #{run[p]}"
      end
end
```

you would see that a number of the fields have non-nil values as follows:

```
Duration 60
DWFMultiplier 32
EveryNode false
EveryOutflow false
EverySubcatchment false
GaugeMultiplier 1
IncludeBaseFlow false
IncludeLevel false
IncludeNode false
IncludeOutflow false
IncludeRainfall false
```

```
IncludeRunoff false
LevelLag 0
LevelThreshold 0.0
NodeLag 0
NodeThreshold 0.0
OutflowLag 0
OutflowThreshold 0.0
RainfallLag 0
RainfallThreshold 0.0
RainType false
ResultsMultiplier 6
RTCLag 0
Start Time 0.0
SubcatchmentLag 0
SubcatchmentThreshold 0.0
TimeStep 60
```

All other fields are treated as nil by default. However, for a number of fields a nil value is treated as a particular default value for that field as specified in the detailed notes for the fields in question.

Where the name of the field below contains spaces or underscores the spaces or underscores should be used when setting the value in the hash.

Always Use Final State

Data type: Boolean

Location in UI: Main page col 2

Description in UI: Always use state without initialisation

Buildup Time

Data type: Long Integer

Location in UI: Water Quality Form

Notes: BuildUp time – used in water quality simulations, may either be nil (in which case it is not used) or a value between 1 and 1000000

CheckPumps

Data type: Boolean

Location in UI: Timestep Control Sheet - Control Page

Comment

Data type: String

Location in UI: Tree object property page

ConcentrationWarning

Data type: Double

Location in UI: Diagnostics Form

Description in UI: Concentration

Depth

Data type: Double

Location in UI: 2D Sheet, Tolerance Tab

Nil treated as: 0.001

Valid range: 0 - 99999999

Notes: Depth and InnudationMapDepthThreshold – Depth must be greater than 0 and less then 99999999, and also must be less then InnundationMapDepthThreshold.

Depth_threshold

Data type: Double

Location in UI: 2d Sheet, Steady State Tab

Description in UI: Threshold for 1-hour change in depth

DontApplyRainfallSmoothing

Data type: Boolean

Location in UI: Main Page col 2

Nil treated as: TRUE

Notes: NB, this is the opposite sense to the check box on the dialog

DontLogModeSwitches

Data type: Boolean

Location in UI: Diagnostics Form

Notes: NB, this is the opposite sense to the check box on the dialog

DontLogRTCRuleChanges

Data type: Boolean

Location in UI: Diagnostics Form

Notes: NB, this is the opposite sense to the check box on the dialog

DontOutputRTCState

Data type: Boolean

Location in UI: Diagnostics Form

Notes: NB, this is the opposite sense to the check box on the dialog

Duration

Data type: Long Integer

Location in UI: Main Page col 1

Default: 60

Notes: Duration of simulation, in units used in duration unit

DurationUnit

Data type: String

Location in UI: Main Page col 1

Notes: The DurationType field must be nil or one of the strings 'Minutes', 'Hours', 'Days', 'Weeks', 'Years'. It is important to realise that the value of this fields does NOT affect the meaning of the Duration field, which is always in minutes, it merely affects the way the

duration is displayed e.g. to run a simulation for a day, and have the time in the run view displayed as '1 day' you should enter the values 1440 in the Duration fields and 'Days' in the Duration Type field.

DWFDefinition

Data type: String

Location in UI: Timestep Control Sheet - RTC Page

Default: '

Description in UI: DWF Mode Definition

DWFModeResults

Data type: Boolean

Location in UI: Timestep Control Sheet - Control Page

Description in UI: Store results when in DWF mode

DWFMultiplier

Data type: Long Integer

Location in UI: Timestep Control Sheet - Control Page

Default: 32

Notes: Must be a power of 2 between 1 and 2048

End Duration

Data type: Boolean

Location in UI: Main Page col 1

Notes: True for time/date, false for duration

End Time

Data type: Double / DateTime (see note 1)

Location in UI: Main Page col 1

Notes: See note 1

EveryNode

Data type: Boolean

Location in UI: Timestep Control Sheet - Node Page

Default: false

Notes: True = Total flow into system, False = flow at each node

EveryOutflow

Data type: Boolean

Location in UI: Timestep Control Sheet - Outflows Page

Default: false

Notes: True = Total flow from system, False = flow at each outfall

EverySubcatchment

Data type: Boolean

Location in UI: Timestep Control Sheet - Subcatchment Page

Default: false

Notes: True = Total flow into system, False = flow at each subcatchment

ExitOnFailedInit

Data type: Boolean

Location in UI: Main Page col 2

Notes: Exit if initialisation fails

ExitOnFailedInitCompletion

Data type: Boolean

Location in UI: Main Page col 2

Description in UI: Exit if initialisation complete in (mins)

Notes: If the ExitOnFailedInitCompletion field is set to true, the InitCompletionMinutes field must be set to a value between 1 and 99999999

GaugeMultiplier

Data type: Long Integer

Location in UI: Main Page col 1

Default: 1

Valid range: 0 - 99999

Notes: Gauge timestep multiplier

Gauges

Data type: Index

Location in UI: Main Page col 1

Description in UI: Additional links to be gauged

Notes: a selection list object

GetStartTimeFromRainEvent

Data type: Boolean

Location in UI: Main Page col 2

Ground Infiltration

Data type: Index

Location in UI: Main Page col 3

Notes: Ground Infiltration object used for simulation

IncludeBaseFlow

Data type: Boolean

Location in UI: Timestep Control Sheet - Subcatchments Page

Default: false

Description in UI: Include Base Flow

IncludeLevel

Data type: Boolean

Location in UI: Timestep Control Sheet - Level Page

Default: false

Description in UI: Check for levels

IncludeNode

Data type: Boolean

Location in UI: Timestep Control Sheet - Node Page

Default: false

Description in UI: Check for inflows

IncludeOutflow

Data type: Boolean

Location in UI: Timestep Control Sheet - Outflows Page

Default: false

Description in UI: Check for ouflows

IncludeRainfall

Data type: Boolean

Location in UI: Timestep Control Sheet - Rainfall Page

Default: false

Description in UI: Check for rainfall

IncludeRTC

Data type: Boolean

Location in UI: Timestep Control Sheet - RTC Page

Description in UI: Check RTC

IncludeRunoff

Data type: Boolean

Location in UI: Timestep Control Sheet - Subcatchments Page

Default: false

Description in UI: Include Runoff

Inflow

Data type: Index

Location in UI: Main Page col 3

Notes: Inflow object used for simulation

InitCompletionMinutes

Data type: Long Integer

Location in UI: Main page col 2

Initial Conditions 2D

Data type: Index

Location in UI: Main Page col 3

Notes: 2D Initial conditions object used for simulation

InundationMapDepthThreshold

Data type: Double

Location in UI: 2D Sheet, Advanced Tab

Nil treated as: 0.01

Valid range: >=0

Notes: see Depth above.

Level

Data type: Index

Location in UI: Main Page col 3

Notes: Level object used for simulation

LevelLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - Level Page

Default: 0

LevelThreshold

Data type: Double

Location in UI: Timestep Control Sheet - Level Page

Default: 0

Valid range: 0 – 99999999

MaxVelocity

Data type: Double

Location in UI: 2D Sheet, Advanced Tab

Nil treated as: 10

Valid range: 0 - 99999999

Minor Timestep 2D

Data type: Boolean

Location in UI: 2D Sheet, Advanced Tab

Description in UI: Link 1D-2D calculations at minor timestep

Momentum

Data type: Double

Location in UI: 2D Sheet, Tolerance Tab

Nil treated as: 0.01

Valid range: 0 - 99999999

NodeLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - Node Page

Default: 0

NodeThreshold

Data type: Double

Location in UI: Timestep Control Sheet - Node Page

Default: 0

OutflowLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - Outflows Page

Default: 0

OutflowThreshold

Data type: Double

Location in UI: Timestep Control Sheet - Outflows Page

Default: 0

Pipe Sediment Data

Data type: Index

Location in UI: Main Page col 3

Notes: Pipe sediment data object used for simulation

Pollutant Graph

Data type: Index

Location in UI: Main Page col 3

Notes: Pollutant graph object used for simulation

QM Dependent Fractions

Data type: Boolean

Location in UI: Water Quality Form

Description in UI: Dependent Sediment Fractions

QM Hydraulic Feedback

Data type: Boolean

Location in UI: Water Quality Form

Description in UI: Erosion Deposition Affects Hydraulics

QM Model Macrophytes

Data type: Boolean

Location in UI: Water Quality Form

QM Multiplier

Data type: Long Integer

Location in UI: Water Quality Form

Valid range: 0 – 10

QM Native Washoff Routing

Data type: Boolean

Location in UI: Water Quality Form

QM Oxygen Demand

Data type: Text

Location in UI: Water Quality Form

Nil treated as: BOD

QM Pollutant Enabled

introduced in 8.0.1

Date Type: Array

This is an array of strings e.g. e.g. 'BOD', 'BODSF1', 'PL1SF2'. The 2d character pollultants e.g. PH have an underscore i.e. 'PH_'. The pollutants are:

```
"BOD",
"COD",
```

```
"TKN",
"NH4",
"TPH",
"PL1",
"PL2",
"PL3",
"PL4",
"DO_",
"NO2",
"NO3",
"PH_",
"SAL",
"TW_",
"COL",
"ALG",
"SI_"
```

With SF1 or SF2 appended for the sediment fractions.

The easiest way to understand this field is to set up a run in the UI and export the array.

RainfallLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - Rainfall Page

Default: 0

RainfallThreshold

Data type: Double

Location in UI: Timestep Control Sheet - Rainfall Page

Default: 0

RainType

Data type: Boolean

Location in UI: 2D Sheet, Advanced Tab

Default: false

Description in UI: Ignore rain falling on dry elements

ReadSubeventNAPIAndAntecedDepth

Data type: Boolean

Location in UI: Main Page col 2

Description in UI: Read subevent NAPI and Antecdent Depth

ReadSubeventParams

Data type: Boolean

Location in UI: Main Page col 2

Description in UI: Read subevent UCWI & Evaporation

Regulator

Data type: Index

Location in UI: Main Page col 2

Notes: Regulator object used for simulation

ResultsMultiplier

Data type: Long Integer

Location in UI: Main Page col 1

Default: 6

Valid range: 0 - 99999

Description in UI: Results timestep multiplier

RTCLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - RTC Page

Default: 0

Valid range: 0 - 99999999

RTCRulesOverride

Data type: Boolean

Location in UI: Main Page col 2

Description in UI: RTC rules override pump on levels

RunoffOnly

Data type: Boolean

Location in UI: Main Page col 2

Notes: Restrictions as in UI

Save Final State

Data type: Boolean

Location in UI: Main Page col 2

Sediment Fraction Enabled
introduced in 8.0.1

Data type: Array

This parameter must be an array of 2 Boolean values, true if you want that sediment fraction and false if you don't.

Sim

Data type: Index

Location in UI: Main Page col 2

Notes: Sim object used for the initial state

SpillCorrection

Data type: Boolean

Location in UI: 2D Sheet, Advanced Tab

Nil treated as: TRUE

Description in UI: Adjust bank levels based on adjacent element ground levels

Start Time

Data type: Double / DateTime

Location in UI: Main Page col 1

Default: 0

Notes: See note 1

StopOnEndOfTimeVaryingData

Data type: Boolean

Location in UI: Timestep Control Sheet - Control Page

Description in UI: Stop simulation at the end of time varying data

StorePRN

Data type: Boolean

Location in UI: Main Page col 2

Description in UI: Summary (PRN) results

StormDefinition

Data type: String

Location in UI: Timestep Control Sheet - RTC Page

Default: '

Description in UI: Storm Mode Condition

SubcatchmentLag

Data type: Long Integer

Location in UI: Timestep Control Sheet - Subcatchments Page

Default: 0

SubcatchmentThreshold

Data type: Double

Location in UI: Timestep Control Sheet - Subcatchments Page

Default: 0

Theta

Data type: Double

Location in UI: 2D Sheet, Advanced Tab

Nil treated as: 0.9

Valid range: 0 - 99999999

Time_lag

Data type: Double

Location in UI: 2d Sheet, Steady State Tab

Nil treated as: 60

TimeStep

Data type: Long Integer

Location in UI: Main Page col 1

Default: 60

Valid range: 1 - 99999

Description in UI: Timestep (s)

timestep_stability_control

Data type: Double

Location in UI: 2D Sheet, Advanced Tab

Nil treated as: 0.95

Valid range: 0 - 1

TimestepLog

Data type: Boolean

Location in UI: Diagnostics Form

Trade Waste

Data type: Index

Location in UI: Main Page col 3

Notes: Trade Waste object used for simulation

Use_local_steady_state

Data type: Boolean

Location in UI: 2d Sheet, Steady State Tab

Description in UI: Deactivate steady state areas

UseGPU

Data type: Long Integer

Location in UI: 2D Sheet, GPU Tab

Notes: 0 or nil = never, 1 = if available, 2 = always

UseQM

Data type: Boolean

Location in UI: Main Page col 3

Velocity

Data type: Double

Location in UI: 2D Sheet, Tolerance Tab

Nil treated as: 0.01

Valid range: 0 - 99999999

Velocity_threshold

Data type: Double

Location in UI: 2d Sheet, Steady State Tab

Description in UI: Threshold for 1-hour change in velocity

VelocityWarning

Data type: Double

Location in UI: Diagnostics Form

Description in UI: Velocity

VolumeBalanceWarning

Data type: Double

Location in UI: Diagnostics Form

Description in UI: Volume balance

WarningBag
introduced in 8.1

Data type: Hash

This value holds the warning thresholds for water priority parameters. It is a hash from strings to floating point numbers.

The keys are as described in the 'QM Pollutant Enabled' key above.

As with that key, the best way to understand this parameter is to set up values in the UI and export them in a script.

Waste Water

Data type: Index

Location in UI: Main Page col 3

Description in UI: Waste Water object used for simulation

Working

Data type: Boolean

Location in UI: Main Page col 1

Description in UI: Allow re-runs using updated network

Notes: Must be set to true before the update_to_latest method may be used.

Notes

1. As ICM Exchange does not have a Ruby data time to represent the use of times in ICM simulations, in which both relative times and absolute times are used, the following convention is used for the start time and end time:
   Absolute times are represented as a DateTime object, relative times as a negative double – a time in seconds. This is similar to the representation used in the results binary file. Therefore to set a relative time, negate the number of seconds and set the field to this value, to set an absolute time use a ruby DateTime object as described earlier in this document.

When reading a value from the database to determine whether the start time is relative of absolute you will want to use code like this:

```
myStartTime=working['Start Time']

if myStartTime.nil?

      myText='nil'

elsif myStartTime.kind_of? DateTime

      myText="absolute -
#{myStartTime.year}/#{myStartTime.month}/#{myStartTime.day}"

elsif myStartTime.kind_of? Float

      myText="relative - #{-myStartTime} seconds"

else

      myText="unexpected type"

end

puts "#{myText}"
```

2. The percentage volume balance is not available from ICM Exchange.

Appendix 3 – ICM SWMM Run Parameters

The fields that are set on the run dialog in the user interface are, in ICM Exchange, set as key value pairs within the hash passed in as the 6th parameter of the new_run method, called on the asset group in which the run is to be created.

The keys of the hash are all strings, the values are of several different types as described below.

Where the values have units, they must always be specified in S.I. units.

The run parameters used for ICM Exchange broadly speaking correspond to those set in the user interface. The list below therefore includes the field's location in the run dialog and its sub-dialogs, and its description in the user interface if the difference is noteworthy.

Consult InfoWorks ICM's main help for more details.

The behaviour of unspecified values for run parameters is as follows: When the run is created most of the run parameters are supplied with default values. This means that were you to create a run, passing an empty hash in as the last parameter, and then to look at the values for the parameters using the [] method of the run object i.e.

you would see that a number of the fields have non-nil values as follows:

```
runBuilder=WSSWMMRunBuilder.new
runBuilder.list_parameters.each do |p|
  if !runBuilder[p].nil?
    puts "#{p} = #{runBuilder[p]}"
  end
end
```

```
network = 0

network_commit_id = 0

working = false

date_time_start_date = 2020-04-01T00:00:00+00:00

date_time_end_date = 2020-04-02T00:00:00+00:00

date_time_report_start = 2020-04-01T00:00:00+00:00

date_time_start_sweep = 2000-01-01T00:00:00+00:00

date_time_end_sweep = 2000-12-31T00:00:00+00:00

dry_days = 0.0

time_step_report = -900.0

time_step_dry = -3600.0
```

```
time_step_wet = -300.0
time_step_control = 0.0
time_step_route = 30.0
stdy_flow_skip_stdy_state = false
stdy_flow_sys_tol = 5.0
stdy_flow_lat_tol = 5.0
proc_parm_rain = true
proc_parm_rdii = true
proc_parm_snow = true
proc_parm_gw = true
proc_parm_route = true
proc_parm_wq = true
dyn_wave_use_var_step = true
dyn_wave_variable_step = 75.0
dyn_wave_length_step = 0.0
dyn_wave_minimum_step = 0.5
rpt_parm_det_obj_id = 0
rpt_parm_continue = true
rpt_parm_flow_stats = true
rpt_parm_input = false
rpt_parm_controls = false
inflow = 0
level = 0
time_pattern = 0
climatology = 0
surcharge_method_type = Extran
save_state_at_end = false
initial_state_sim = 0
```

All other fields are treated as nil by default. However, for a number of fields a nil value is treated as a particular default value for that field as specified in the detailed notes for the fields in question.

Where the name of the field below contains spaces or underscores the spaces or underscores should be used when setting the value in the hash.

climatology

Data type: Index

Location in UI: Main page col 3

Description in UI: SWMM climatology

Notes: SWMM Climatology object used in simulation

date_time_end_date

Data type: DateTime

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: End analysis

Default: Current date +1 day at time 00:00:00

date_time_end_sweep

Data type: DateTime

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: End sweeping

Default: 31 Dec

Notes: The year and time are ignored

date_time_report_start

Data type: DateTime

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: Start reporting

Default: Current date at time 00:00:00

date_time_start_date

Data type: DateTime

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: Start analysis

Default: Current date at time 00:00:00

date_time_start_sweep

Data type: DateTime

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: Start sweeping

Default: 01 Jan

Notes: The year and time are ignored

dry_days

Data type: Float

Location in UI: Timestep Control Sheet - Dates Page

Description in UI: Antecedent dry days

Default: 0,0

Notes: Must be zero or a positive float value.

dyn_wave_length_step

Data type: Float

Location in UI: Main Page col 4 – Dynamic wave group box

Description in UI: Conduit lengthening timestep

Default: 0.0

dyn_wave_minimum_step

Data type: Float

Location in UI: Main Page col 4 – Dynamic wave group box

Description in UI: Minimum timestep

Default: 0.5

dyn_wave_use_var_step

Data type: Boolean

Location in UI: Main Page col 4 – Dynamic wave group box

Description in UI: Adjust variable timesteps by (%) - checkbox

Default: true

dyn_wave_variable_step

Data type: Float

Location in UI: Main Page col 4 – Dynamic wave group box

Description in UI: Adjust variable timesteps by (%) - edit box

Default: 75.0

Range: 10.0 – 200.0

inflow

Data type: Index

Location in UI: Main Page col 2

Description in UI: Inflow

Notes: Inflow object used in simulation

initial_state_sim

Data type: Index

Location in UI: Main Page col 1

Description in UI: Sim providing initial state

Notes: The simulation object that provides the initial state to host-start the simulation. The simulation providing state must have saved its state and its simulation succeeded.

level

Data type: Index

Location in UI: Main Page col 2

Description in UI: Level

Notes: Level object used in simulation

name

Data type: String

Location in UI: Top of main Page

Description in UI: Run title

Notes: If the name is Nil then one is randomly generated

network

Data type: Index

Location in UI: Main Page col 1 – Network group box

Description in UI: SWMM network

Notes: Network object used in simulation

network_commit_id

Data type: Long Integer

Location in UI: Main Page col 1 – Network group box

Description in UI: SWMM network

Notes: Network commit ID or version number used in simulation that appears in parenthesis following the network name

pollutographs

Data type: Array of Index

Location in UI: Main Page col 3

Description in UI: SWMM pollutograph

proc_parm_gw

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Groundwater

Default: true

proc_parm_rain

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Rainfall / runoff

Default: true

proc_parm_rdii

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Rainfall dependent I/I

Default: true

proc_parm_route

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Flow routing

Default: true

proc_parm_snow

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Snowmelt

Default: true

proc_parm_wq

Data type: Boolean

Location in UI: Options Sheet – Processes Page

Description in UI: Water quality

Default: true

rainfall

Data type: Index

Location in UI: Main Page col 2

Description in UI: Rainfall event / Flow survey

Notes: Rainfall or flow survey object used in simulation

regulator

Data type: Index

Location in UI: Main Page col 2

Description in UI: Regulator

Notes: Regulator object used in simulation

rpt_parm_averages

Data type: Boolean

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Average results

Default: false

rpt_parm_continue

Data type: Boolean

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Continuity checks

Default: true

rpt_parm_controls

Data type: Boolean

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Control actions

Default: false

rpt_parm_det_obj_id

Data type: Array of Index

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Objects for detailed reporting – Selection list

rpt_parm_flow_stats

Data type: Boolean

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Summary flow statistics

Default: true

rpt_parm_input

Data type: Boolean

Location in UI: Main Page col 4 – Reporting group box

Description in UI: Input summary

Default: false

save_state_at_end

Data type: Boolean

Location in UI: Main Page col 1

Description in UI: Save state at end of simulation

Default: false

scenarios

Data type: Array of String

Location in UI: Main Page col 1 – Network group box

Description in UI: Scenarios

stdy_flow_lat_tol

Data type: Float

Location in UI: Timestep Control Sheet - Timesteps Page – Steady flow periods group box

Description in UI: Lateral flow tolerance (%)

Default: 5.0

stdy_flow_skip_stdy_state

Data type: Boolean

Location in UI: Timestep Control Sheet - Timesteps Page – Steady flow periods group box

Description in UI: Skip steady flow periods

Default: false

stdy_flow_sys_tol

Data type: Float

Location in UI: Timestep Control Sheet - Timesteps Page – Steady flow periods group box

Description in UI: System flow tolerance (%)

Default: 5.0

surcharge_method_type

Data type: String

Location in UI: Options Sheet – Surcharge Method Page

Description in UI: Surcharge Method

Default: Extran

Notes: Must be either Extran or Slot.

time_pattern

Data type: Index

Location in UI: Main Page col 3

Description in UI: SWMM time patterns

Notes: SWMM time pattern object used for simulation.

time_step_control

Data type: Float

Location in UI: Timestep control sheet – Timesteps Page

Description in UI: Control rule step

Default: 0.0

Notes: Relative time in seconds. Must be a negative value.

time_step_dry

Data type: Float

Location in UI: Timestep control sheet – Timesteps Page

Description in UI: Dry weather runoff step

Default: -3600.0

Notes: Relative time in seconds. Must be a negative value.

time_step_report

Data type: Float

Location in UI: Timestep control sheet – Timesteps Page

Description in UI: Reporting timestep

Default: -900.0

Notes: Relative time in seconds that timesteps are reported. Must be a negative value.

time_step_route

Data type: Float

Location in UI: Timestep control sheet – Timesteps Page

Description in UI: Routing timestep (s)

Default: 30.0

Notes: Unlike other time_steps, this is an explicit number of seconds.

time_step_wet

Data type: Float

Location in UI: Timestep control sheet – Timesteps Page

Description in UI: Wet weather runoff step

Default: -300.0

Notes: Relative time in seconds. Must be a negative value.

working

Data type: Boolean

Location in UI: Main Page col 1

Description in UI: Allow re-runs using updated network

Default: false

Notes: Must be set to true before the update_to_latest method may be used.

Appendix 4 – WS Pro run parameters

There are a lot of parameters, but they map onto the field names in a fairly straight-forward fashion. Note that some of the parameters are used in types of runs which are not current supported. Note that currently only certain run types may be set up and run – see below.

The naming scheme is as follows (with a few exceptions noted below):

aa_b_xxxx

aa is the run type

ro Run Options (required for all runs)

py Physical params (all runs)

op Optimiser

hs Hot start

pd PRD

in 'Independent' i.e. other

ca Calibration

wq Water Quality

ws WatSed

ff FireFlow

cl Critical Link

bs Break / Shutdown

b is the data type

l long

n integer

s string

dte date

f float

b bool

d double

Note that the following runs may be set up and run – the run type is set in the key ro_l_run_type - the run types are the integers following the type names:

Normal – 0
Calibration – 1
Water Quality – 2
Watsed – 4
Critical Link –6
Break shutdown – 7

Key	Note
ro_l_run_type	See note above
ro_s_run_title	
ro_l_geometry_id	
ro_l_geometry_commit_id	
ro_l_demand_diagram_id	
ro_l_demand_scaling_id	
ro_l_alt_demand_id	
ro_l_alt_demand_commit_id	
ro_l_electricity_tariff_id	
ro_l_rtc_id	
ro_l_control_id	
ro_l_control_commit_id	
ro_dte_end_date_time	
ro_dte_start_date_time	
ro_l_time_step	
ro_l_results_selector_id	
ro_n_results_selection_mode	
ro_l_max_iterations	
ro_f_computational_accuracy	
ro_n_demand_timestep	
ro_b_pressure_related_demand	
ro_b_disconnected_system	
ro_b_optimise	
ro_b_eghgf	
ro_b_experimental	
ro_b_results_on_server	
ro_b_store_details	
ro_b_store_max	
ro_b_gmr_enable	

ro_l_gmr_config_id	
ro_l_test_cases_per_thread	
py_d_viscosity	
py_d_density	
py_d_gravity	
op_l_population_size	
op_d_crossover_prob	
op_d_mutation_prob	
op_d_profile_time_interval	
op_b_start_from_existing	
op_b_update_control_data	
hs_l_simulation_id	
hs_s_start_times	
hs_s_save_times	
hs_b_save_state	
pd_l_profile_id	
pd_s_demand_curve	
pd_s_leakage_curve	
In_b_validate_model	
In_b_validate_run	
ca_l_live_data_id	
ca_l_live_data_commit_id	
ca_s_friction_type	
ca_f_min_friction	
ca_f_max_friction	
ca_dte_snapshot_time	
ca_f_init_scaling_factor	
wq_b_conservative_substance	
wq_f_init_concentration	
wq_f_min_flow	
wq_l_timestep	
wq_b_langrangian_solver	
wq_d_age_tolerance	
wq_d_conc_tolerance	
wq_d_trace_tolerance	
wq_d_turbidity_tolerance	
wq_b_turbidity_analysis	
wq_l_solute_data_id	
wq_trace_node_0	String
wq_trace_node_1	String (... and upto node_29)

wq_b_langrangian_solver	
wq_d_age_tolerance	
wq_d_conc_tolerance	
wq_d_trace_tolerance	
wq_d_turbidity_tolerance	
wq_b_turbidity_analysis	
ws_s_sediment_name	
ws_f_sediment_density	
ws_f_sediment_diameter	
ws_f_deposition_limit	
ws_f_suspension_limit	
ff_l_selection_id	
ff_l_data_id	
ff_f_hydrant_diameter	
ff_f_local_loss	
ff_f_fire_flow	
ff_f_residual_pressure	
ff_n_Enforce	
ff_n_simulation_type	
ff_n_data_usage	
ff_dte_fire_time	
ff_b_cancel_existing_flow	
ff_b_zone_constraints	
ff_b_system_constraints	
ff_b_pressure_at_min_and_max	
ff_b_calculate_max_flow	
ff_b_calculate_hydrant_curve	
ff_b_apply_constraints_demand_nodes	
ff_b_insert_node	
ff_s_existing_node_id	
ff_s_split_pipe_id	
ff_f_split_pipe_distance	
ff_n_close_pipe_option	
ff_f_max_velocity	
ff_f_min_node_pressure	
ff_f_min_system_pressure	
ff_test_flow_0	Double
ff_test_flow_1	Double
ff_test_flow_2	Double
ff_test_flow_3	Double

ff_test_flow_4	Double
ff_test_flow_5	Double
ff_test_flow_6	Double
ff_test_flow_7	Double
ff_test_flow_8	Double
ff_test_flow_9	Double
cl_f_min_pressure	
cl_f_duration	
cl_f_max_pressure	
cl_f_demand_efficiency	
cl_l_ignore_count	
cl_n_count_affected	
cl_f_burst_rate	
cl_f_outage_duration	
cl_f_burst_duration	
cl_n_link_outage_period	
cl_dte_specified_time	
cl_b_report_outage_only	
cl_b_include_burst	
cl_b_allow_flow	
cl_b_update_criticality	
cl_l_include_links_selection_id	
cl_l_exclude_links_selection_id	
bs_l_close_link_selection_id	
bs_l_base_simulation_id	
bs_f_min_pressure	
bs_f_min_press_duration	
bs_f_max_pressure	
bs_f_max_press_duration	
bs_f_max_inc_upper_threshold	
bs_f_upper_threshold_duration	
bs_f_max_dec_lower_threshold	
bs_f_lower_threshold_duration	
bs_b_whole_simulation_outage	
bs_dte_shutdown_start	
bs_dte_shutdown_end	
ro_l_result_time_step	

Appendix 5 –'Add-ons' (ICM / InfoAsset only)

It is possible to store a CSV file containing the names of a number of scripts along with a name of a menu item to invoke them. These appear as sub-menu items of the 'Run add-on' menu item, which also appears on the 'Network' menu.

The CSV file must be stored in a directory below that used by the user's application data named 'scripts' and must be called 'scripts.csv'.

The name of the directory used by the user's application data will vary according to the user's set up, version of Windows etc. and can be found in the about box of the software as 'NEP (iws) Folder'.

Having found this folder e.g.

C:\Users\badgerb\AppData\Roaming\Innovyze\WorkgroupClient

Add a sub-directory called 'scripts'.

The folder may also be determined by using the WSApplication.add_on_folder method, this will return the path of the scripts folder i.e. C:\Users\badgerb\AppData\Roaming\Innovyze\WorkgroupClient\scripts in this case.

In this scripts.csv file you should add a CSV file containing 2 columns, the first being the menu item for the script, the second the path for the script file itself.

The paths for the script files may either be fully qualified paths (i.e. beginning with a drive letter or the name of a network share) in which case that path will be used, or a non-fully qualified path in which case the software will assume the file is in the folder containing the csv file or a subdirectory of it.

Changes to this file only take effect when the application is restarted

Appendix 6 - Open Data Import / Export Centre UI Customisation (ICM / InfoAsset only)

The Ruby scripting can be used to make the import / export of data via the open data import centre more streamlined for users of the software by using Ruby scripts from the UI in conjunction with pre-prepared configuration files and the Ruby scripting's UI elements.

At its simplest, if you can hard-code the paths of all files, then this can be done with 2 lines of code e.g.

```
net=WSApplication.current_network
net.odic_import_ex('CSV','d:\temp\odic.cfg',nil,'Node','d:\temp\goat
.csv','Pipe','d:\temp\stoat.csv')
```

for import and

```
net=WSApplication.current_network
net.odec_export_ex('CSV','d:\temp\odxc.cfg',nil,'Node','d:\temp\goat
2.csv','Pipe','d:\temp\stoat2.csv')
```

for export.

As described above, both methods take a variable number of parameters. If you are importing a large number of files you may find it less unwieldy to call the method multiple times importing one file at a time e.g.

```
net=WSApplication.current_network
import=[['Node','goat'],['Pipe','stoat']]
import.each do |f|
      net.odic_import_ex('CSV','d:\temp\odic.cfg',nil,f[0],'d:\temp\\
'+f[1]+'.csv')
end
```

for import and

```
net=WSApplication.current_network
export=[['Node','goat'],['Pipe','stoat']]
export.each do |f|
      net.odec_export_ex('CSV','d:\temp\odxc.cfg',nil,f[0],'d:\temp\\
'+f[1]+'2.csv')
end
```

for export.

It should be noted that

a) You will not see any of the error messages on import that would appear in the text box. Exceptions are not thrown for that sort of error, only for more serious errors in the processing.

b) By using nil in the 3rd parameter of each method, default behaviour will be used for the options set on the dialog, this may not be what you want.

The first of these can be solved by specifying an error text file e.g.

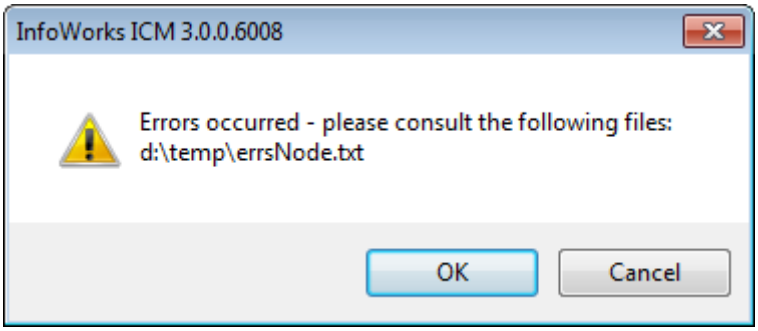
```
net=WSApplication.current_network
import=[['Node','goat'],['Pipe','stoat']]
import.each do |f|
      params=Hash.new
      params['Error File']='d:\\temp\\errs'+f[0]+'.txt'
      net.odic_import_ex('CSV','d:\temp\odic.cfg',params,f[0],'d:\tem
p\\'+f[1]+'.csv')
end
```

The aim here is to produce one file per table. The files will be created but of zero bytes long if there are no errors for that table.

You will probably want to communicate the errors to the user. In its simplest form this could be done by checking the size of the files and displaying a message box at the end of the process e.g.

```
require 'FileUtils'
net=WSApplication.current_network
import=[['Node','goatwitherrs'],['Pipe','stoat']]
errFiles=Array.new
import.each do |f|
      params=Hash.new
      errFile='d:\\temp\\errs'+f[0]+'.txt'
      params['Error File']=errFile
      net.odic_import_ex('CSV','d:\temp\odic.cfg',params,f[0],'d:\tem
p\\'+f[1]+'.csv')
      if File.size(errFile)>0
            errFiles << errFile
      else
            FileUtils.rm errFile
      end
end
if errFiles.size>0
      msg="Errors occurred - please consult the following files:"
      errFiles.each do |f|
            msg+="\r\n"
            msg+=f
      end
      WSApplication.message_box msg,nil,nil,nil
end
```

This will display a message reporting to the user the error files which should be consulted e.g.



Note the inclusion of FileUtils and the use of the FileUtils.rm method to delete files of zero length.

If you wish to show the user the actual messages then this can be achieved either by reading the files and outputting them to the standard output e.g.

```
require 'FileUtils'
net=WSApplication.current_network
import=[['Node','goatwitherrs','nodes'],['Pipe','stoat','pipes']]
errInfo=Array.new
import.each do |f|
      params=Hash.new
      errFile='d:\\temp\\errs'+f[0]+'.txt'
      if File.exists? errFile
            FileUtils.rm errFile

      end
      params['Error File']=errFile
      net.odic_import_ex('CSV','d:\temp\odic.cfg',params,f[0],'d:\tem
p\\'+f[1]+'.csv')
      if File.size(errFile)>0
            temp=Array.new
            temp << errFile
            temp << f[2]
            errInfo << temp
      else
            FileUtils.rm errFile
      end
end
if errInfo.size>0
      puts "Errors importing data:"
      errInfo.each do |ei|
            puts "Errors for #{ei[1]}:"
            outputString=''
            File.open ei[0] do |f|
                  f.each_line do |l|
                        l.chomp!
                        outputString+=l
                        outputString+="\r"
                  end
```

```
            end
            puts outputString
      end
end
```

Or by using the open_text_view method, in which case the block beginning `with if ErrInfo.size>0` would be replaced with the following:

```
if errInfo.size>0
      consolidatedErrFileName='d:\\temp\\allerrs.txt'
      if File.exists? consolidatedErrFileName
            FileUtils.rm consolidatedErrFileName
      end
      consolidatedFile=File.open consolidatedErrFileName,'w'
      errInfo.each do |ei|
            consolidatedFile.puts "Errors for #{ei[1]}:"
            File.open ei[0] do |f|
                  f.each_line do |l|
                        l.chomp!
                        consolidatedFile.puts l
                  end
            end
      end
      consolidatedFile.close
      WSApplication.open_text_view 'Open Data Import Centre
Errors',consolidatedErrFileName,false
end
```

You may wish to not hard code the path of the config file but to store it with the Ruby script. This may be done by obtaining the path of the folder containing the script then adding the configuration file name onto the name e.g.

```
configfile=File.dirname(WSApplication.script_file)+'\\odicwithsource
.cfg'
```

This works via the following 3 steps:

Get the file name of the script file e.g. d:\temp\myscript.rb

Use the File.dirname method to obtain the folder name e.g. d:\temp

Add the configuration file name e.g. d:\temp\odicwitsource.cfg

Alternatively you may wish to allow the user to choose a config file using the WSApplication.file_dialog method e.g. by beginning the script with

```
net=WSApplication.current_network
configfile=WSApplication.file_dialog(true,'cfg','Open Data Import
Centre Config File',nil,false,false)
if configfile.nil?
      WSApplication.message_box 'No config file selected - no import
will be performed',nil,nil,false
```

```
Else
```

and then using configfile in the call to odic_import_ex

Similarly you may wish to allow the user to choose the location of the data files or database tables etc. This may be done in numerous ways depending on the data type and/or how things are structured.

Possible mechanisms include:

1 Allowing the user to select a folder and then using hard-coded names based on that folder.

2 Allowing the user to choose one file and then selecting similarly named files in the same folder (e.g. if we are expecting a file with the suffix 'stoat' and we find a file called 'northwest_stoat' we will also look for files called 'northwest_goat' etc.)

3 Allowing the user to select multiple files and choosing the data type to import based on the file names.

Here are brief examples of the 3 mechanisms:

Mechanism 1:

```
require 'FileUtils'
net=WSApplication.current_network
configfile=WSApplication.file_dialog(true,'cfg','Open Data Import
Centre Config File',nil,false,false)
if configfile.nil?
      WSApplication.message_box 'No config file selected - no import
will be performed',nil,nil,false
else
      folder=WSApplication.folder_dialog 'Select a folder containing
the files to import',false
      if folder.nil?
            WSApplication.message_box 'No folder selected - no import
will be performed'
      else

      import=[['Node','goatwitherrs','nodes'],['Pipe','stoat','pipes'
]]
            errInfo=Array.new
            import.each do |f|
                  params=Hash.new
                  errFile=folder+'\\errs'+f[0]+'.txt'
                  if File.exists? errFile
                        FileUtils.rm errFile
                  end
                  params['Error File']=errFile
```

```
      net.odic_import_ex('CSV',configfile,params,f[0],folder+'\\'+f[1
]+'.csv')
                  if File.size(errFile)>0
                        temp=Array.new
                        temp << errFile
                        temp << f[2]
                        errInfo << temp
                  else
                        FileUtils.rm errFile
                  end
            end
            if errInfo.size>0
                  puts "Errors importing data:"
                  errInfo.each do |ei|
                        puts "Errors for #{ei[1]}:"
                        outputString=''
                        File.open ei[0] do |f|
                              f.each_line do |l|
                                    l.chomp!
                                    outputString+=l
                                    outputString+="\r"
                              end
                        end
                        puts outputString
                  end
            end
      end
end
```

Mechanism2:

```
require 'FileUtils'
net=WSApplication.current_network
configfile=configfile=File.dirname(WSApplication.script_file)+'\\odi
cwithsource.cfg'
import=[['Node','goat','nodes'],['Pipe','stoat','pipes']]
file=WSApplication.file_dialog(true,'csv','CSV
File',nil,false,false)
if file.nil?
      WSApplication.message_box 'No file selected - no import will be
performed','OK',nil,false
elsif file[-4..-1].downcase!='.csv'
      WSApplication.message_box 'Not a csv file - no import will be
peformed','OK',nil,false
else
      folder=File.dirname(file)
      name=File.basename(file)[0..-5]
      prefix=''
      found=false
      import.each do |i|
            if name.downcase[-i[1].length..-1]==i[1].downcase
                  prefixlen=name.length-i[1].length
```

```
                  if prefixlen>0
                        prefix=name[0..prefixlen-1]
                  end
                  found=true
                  break
            end
      end
      if !found
            WSApplication.message_box 'File name does not have an
expected suffix - no import will be performed','OK',nil,false
      else
            # errInfo is an array of arrays, with one entry added for
each imported CSV file with some sort of issue
            # it will either contain the error file name and a name
to be used for the table in error messages
            # or nil and a filename for any expected files which are
missing
            errInfo=Array.new
            import.each do |f|
                  csvfilename=folder+'\\'+prefix+f[1]+'.csv'
                  if !File.exists? csvfilename
                        temp=Array.new
                        temp << nil
                        temp << csvfilename
                        errInfo << temp
                  else
                        params=Hash.new
                        errFile=folder+'\\errs'+f[0]+'.txt'
                        if File.exists? errFile
                              FileUtils.rm errFile

                        end
                        params['Error File']=errFile

      net.odic_import_ex('CSV',configfile,params,f[0],csvfilename)
                        if File.size(errFile)>0
                              temp=Array.new
                              temp << errFile
                              temp << f[2]
                              errInfo << temp
                        else
                              FileUtils.rm errFile
                        end
                  end
            end
            if errInfo.size>0
                  puts "Errors importing data:"
                  errInfo.each do |ei|
                        if ei[0].nil?
                              puts "Expected file #{ei[1]} not found"
                        else
                              puts "Errors for #{ei[1]}:"
                              outputString=''
```

```
                                    File.open ei[0] do |f|
                                          f.each_line do |l|
                                                l.chomp!
                                                outputString+=l
                                                outputString+="\r"
                                          end
                                    end
                                    puts outputString
                              end
                        end
                  end
end
end
```

Mechanism3:

```
require 'FileUtils'
net=WSApplication.current_network
configfile=configfile=File.dirname(WSApplication.script_file)+'\\odi
cwithsource.cfg'
import=[['Node','goat','nodes'],['Pipe','stoat','pipes']]
files=WSApplication.file_dialog(true,'csv','CSV
File',nil,true,false)
if files.nil? || files.length==0
      WSApplication.message_box 'No file selected - no import will be
performed','OK',nil,false
else
      nErrs=0
      errInfo=Array.new
      files.each do |file|
            folder=File.dirname(file)
            name=File.basename(file)
            if name[-4..-1].downcase=='.csv'
                  name=name[0..-5]
                  import.each do |i|
                        if i[1].downcase==name.downcase[-
i[1].length..-1]
                              params=Hash.new
                              nErrs+=1

      errFile=folder+'\\errs'+nErrs.to_s+'.txt'
                              if File.exists? errFile
                                    FileUtils.rm errFile
                              end
                              params['Error File']=errFile

      net.odic_import_ex('CSV',configfile,params,i[0],file)
                              if File.size(errFile)>0
                                    temp=Array.new
                                    temp << errFile
                                    temp << i[2]
                                    errInfo << temp
                              else
                                    FileUtils.rm errFile
```

```
                        end
                        break
                    end
                end
            end
        end
        if errInfo.size>0
            puts "Errors importing data:"
            errInfo.each do |ei|
                if ei[0].nil?
                    puts "Expected file #{ei[1]} not found"
                else
                    puts "Errors for #{ei[1]}:"
                    outputString=''
                    File.open ei[0] do |f|
                        f.each_line do |l|
                            l.chomp!
                            outputString+=l
                            outputString+="\r"
                        end
                    end
                    puts outputString
                end
            end
        end
    end
```

Appendix 7 – Character encoding

Note that this behaviour was changed in ICM version 7.0 / InfoNet version 17.0.

The behaviour of strings passed into the ICM / InfoAsset Ruby methods or returned from them is determined by the 'use UTF8' setting set by the WSApplication.use_utf8 method. The default value is false.

If this setting is set to true, the methods will expect strings passed into methods to have the UTF8 encoding, and will return UTF8 strings.

If this setting is set to false, the methods will expect strings passed into methods will have the locale appropriate encoding, and will return strings in that encoding.

The strings are expected to be passed in with the correct encoding – the encoding is not checked, and strings with a different encoding do not have their encoding changed.

If you are using constant strings in your Ruby scripts you will find things go much more smoothly if you use the corresponding encoding in your script. As well as ensuring that the script file is in the encoding you think it is you need to communicate this to Ruby by setting the encoding in the first line of the script e.g

encoding: CP936

(for Simplified Chinese)

OR

encoding: UTF-8

Language	Encoding	Synonym
Bulgarian	Windows-1251	
Japanese	Shift_JIS	CP932
Korean	CP949	
Simplified Chinese	GBK	CP936
Turkish	Windows-1254	CP857
Western European	Windows-1252	CP1252

Appendix 8 – Interacting with Jet databases

In the 32 bit versions of the programs only, Jet databases may be read and written to using the DAO interface, accessed via Ruby's Win32OLE mechanism e.g. to import data

```
require 'WIN32OLE'
DAO=WIN32OLE.new "DAO.DBEngine.36"
WSs=DAO.Workspaces
WS=WSs[0]
db=WS.OpenDatabase 'd:\\temp\\test.mdb'
net=WSApplication.current_network
rs=db.OpenRecordset 'MANHOLES'
net.transaction_begin
if !rs.BOF && !rs.EOF
      rs.MoveFirst
      while !rs.EOF
            ro=net.new_row_object('cams_manhole')
            ro.id=rs.Fields.Item('Name').Value
            ro.x=rs.Fields.Item('x').Value
            ro.y=rs.Fields.Item('y').Value
            ro.cover_level=rs.Fields.item('cover_level').Value
            ro.write
            rs.MoveNext
      end
end
net.transaction_commit
rs.Close
db.Close
WS=nil
WSs=nil
```

And to export data:

```
require 'WIN32OLE'
DAO=WIN32OLE.new "DAO.DBEngine.36"
WSs=DAO.Workspaces
WS=WSs[0]
db=WS.OpenDatabase 'd:\\temp\\test.mdb'
tabledefs=db.TableDefs
tabledefs.each do |t|
      if t.Name=='MANHOLES'
            db.Execute 'DROP TABLE MANHOLES'
            break
      end
end
tabledefs.Refresh
db.Execute "CREATE TABLE MANHOLES (Name VARCHAR(80),x DOUBLE,y
DOUBLE,cover_level DOUBLE)"
rs=db.OpenRecordset 'MANHOLES'
net=WSApplication.current_network
```

```
net.row_objects('cams_manhole').each do |n|
      rs.AddNew
      rs.Fields.Item('Name').Value=n.id
      rs.Fields.Item('x').Value=n.x
      rs.Fields.Item('y').Value=n.y
      if !n.cover_level.nil?
            rs.Fields.Item('cover_level').Value=n.cover_level
      end
      rs.Update
end
db.close
WS=nil
WSs=nil
```

Appendix 9 – Short Codes

ICM and InfoAsset combined database

Name	ShortCode
Action List	ACTL
Alert Definition List	ADL
Alert Instance List	AIL
Asset Group	AG
Asset Network	ASSETNET
Asset Network Template	ASSETTMP
Asset Validation	ASSETVAL
Assimilation	ASSIM
Calibration	PDMC
Collection Cost Estimator	COST
Collection Inference	CINF
Collection Network	CNN
Collection Network Template	CNTMP
Collection Validation	VAL
Custom Graph	CGDT
Custom Report	CR
Damage Calculation Results	DMGCALC
Damage Function	DMGFUNC
Dashboard	DASH
Distribution Cost Estimator	WCOST
Distribution Inference	WINF
Distribution Network	NWNET
Distribution Network Template	WNTMP
Distribution Validation	WVAL
Episode Collection	EPC
Flow Survey	FS
Geo Explorer	NGX
Graph	GDT
Gridded Ground Model	GGM
Ground Infiltration	IFN
Ground Model	GM
Infinity Configuration	INFINITY
Inflow	INF
Initial Conditions 1D	IC1D
Initial Conditions 2D	IC2D
Initial Conditions Catchment	ICCA
Label List	LAB

Name	ShortCode
Layer List	LL
Level	LEV
Lifetime Estimator	LIFEE
Live Group	LG
Manifest	MAN
Manifest Deployment	MAND
Master Group	MASG
Model Group	MODG
Model Inference	INFR
Model Network	NNET
Model Network Template	NNT
Model Validation	ENV
Observed Depth Event	OBD
Observed Flow Event	OBF
Observed Velocity Event	OBV
Pipe Sediment Data	PSD
Point Selection	PTSEL
Pollutant Graph	PGR
Print Layout	PTL
Rainfall Event	RAIN
Regulator	REG
Rehabilitation Planner	REHABP
Risk Analysis Run	RAR
Risk Assessment	RISK
Risk Calculation Results	RISKCALC
Run	RUN
Selection List	SEL
Sim	SIM
Sim Stats	STAT
Statistics Template	ST
Stored Query	SQL
Theme	THM
Time Varying Data	TVD
Trade Waste	TW
TSDB	TSDB
TSDB Spatial	TSDBS
UPM River Data	UPMRD
UPM Threshold	UPTHR
Waste Water	WW
Workspace	WKSP

WS Pro database

Name	ShortCode
Alt_Demand	ALTDMD
Baseline	BLINE
Baseline Explorer	BLINEEX
Catchment Group	CG
Control	CON
Custom Report	CR
Custom Report Group	CRG
Demand Diagram	DDG
Demand Diagram Group	DDGG
Demand Scaling	DSCL
Demand Scaling Group	DSCLG
Electricity Tariff	ETAR
Electricity Tariff Group	ETARG
Energy GHG Factors	EGHGF
Energy GHG Factors Group	EGHGFG
Engineering Validation	ENV
Engineering Validation Group	ENVG
Export Style	ES
Export Style Group	ESG
FireFlow Data	FF
FireFlow Data Group	FFG
Flushing Schedule	FSCH
Flushing Schedule Group	FSCHG
Gen Multi Run Cfg	GMRC
Gen Multi Run Cfg Group	GMRCG
Geo Explorer	GEOEX
Geometry	GMT
Geometry Template	GMTTMPL
Graph	GDT
Graph Group	GDTG
Gridded Ground Model	GGM
Gridded Ground Model Group	GGMG
Ground Model	GM
Ground Model Group	GMG
Inference	INF
Inference Group	INFG
IWL Switch Controller	IWLSC
IWL Switch Controller Group	IWLSCG
IWLive RunInfo	IWLRI

Name	ShortCode
IWLive RunInfo Group	IWLRIG
Label List	LAB
Label List Group	LABG
Layer List	LL
Layer List Group	LLG
Model 360 Cfg	M360C
Model 360 Cfg Group	M360CG
Polygon	POL
Polygon Group	POLG
Report Cfg	RC
Report Cfg Group	RCG
ResultsSelector	RESSEL
ResultsSelector Group	RESSELG
RTC Group	RTCG
RTC Scenario	RTC
Selection List	SEL
Selection List Group	SELG
SoluteData	SD
SoluteData Group	SDG
Stored Query	SQL
Stored Query Group	SQLQ
Theme	THM
Theme Group	THMG
Warning Template	WT
Warning Template Group	WTG
Wesnet Live Data	WNLIVE
Wesnet Run	WNRUN
Wesnet Run Group	WNRUNG
Wesnet Sim	WNSIM
Workspace	WKSP
Workspace Group	WKSPG
Zone Explorer	ZONEEX