ICM Exchange / InfoAsset Exchange / WSPro Exchange

Version 2021.2

February 2021



# Table of Contents

[Table of Contents 2](#_Toc64014888)

[Disclaimer 3](#_Toc64014889)

[Introduction 4](#_Toc64014890)

[UI and Exchange 4](#_Toc64014891)

[Notational Conventions 4](#_Toc64014892)

[Running Scripts 5](#_Toc64014893)

[Running Scripts from the IExchange programs 5](#_Toc64014894)

[Running scripts from the UI 5](#_Toc64014895)

[The Exchange Object Model 7](#_Toc64014896)

[WSNumbatNetworkObject / WSNetworkObject vs WSOpenNetwork 10](#_Toc64014897)

[Methods 10](#_Toc64014898)

[Dates and Times 10](#_Toc64014899)

[Handling objects at the master database level 11](#_Toc64014900)

[Scripting paths 13](#_Toc64014901)

[Handling objects within a network 14](#_Toc64014902)

[Getting and setting values in ‘row objects’. 24](#_Toc64014903)

[Navigating between objects 37](#_Toc64014904)

[Method Reference 40](#_Toc64014905)

[WSApplication 40](#_Toc64014906)

[WSDatabase 55](#_Toc64014907)

[WSModelObjectCollection 58](#_Toc64014908)

[WSModelObject 59](#_Toc64014909)

[Network Classes: 72](#_Toc64014910)

[WSNumbatNetworkObject / WSNetworkObject 72](#_Toc64014911)

[Common 72](#_Toc64014912)

[WSNetworkObject Only 80](#_Toc64014913)

[WSNumbatNetworkObject Only Version Control Methods 81](#_Toc64014914)

[WSNumbatNetworkObject Only Other Methods 83](#_Toc64014915)

[WSSimObject 86](#_Toc64014916)

[WSOpenNetwork 92](#_Toc64014917)

[WSRowObject 112](#_Toc64014918)

[WSNode 116](#_Toc64014919)

[WSLink 117](#_Toc64014920)

[WSRiskAnalysisRunObject 117](#_Toc64014921)

[WSRowObjectCollection 117](#_Toc64014922)

[WSTableInfo 117](#_Toc64014923)

[WSFieldInfo 118](#_Toc64014924)

[WSCommit 119](#_Toc64014925)

[WSCommits 120](#_Toc64014926)

[WSStructure 120](#_Toc64014927)

[WSStructureRow 121](#_Toc64014928)

[WSValidations 121](#_Toc64014929)

[WSValidation 122](#_Toc64014930)

[WSRunScheduler 123](#_Toc64014931)

[WSRun 123](#_Toc64014932)

[WSSWMMRunBuilder 124](#_Toc64014933)

[Appendix 1 – Pollutograph codes 125](#_Toc64014934)

[Appendix 2 – ICM InfoWorks Run Parameters 126](#_Toc64014935)

[Appendix 3 – ICM SWMM Run Parameters 146](#_Toc64014936)

[Appendix 4 – WS Pro run parameters 158](#_Toc64014937)

[Appendix 5 –‘Add-ons’ (ICM / InfoAsset only) 163](#_Toc64014938)

[Appendix 6 - Open Data Import / Export Centre UI Customisation (ICM / InfoAsset only) 164](#_Toc64014939)

[Appendix 7 – Character encoding 173](#_Toc64014940)

[Appendix 8 – Interacting with Jet databases 174](#_Toc64014941)

[Appendix 9 – Short Codes 176](#_Toc64014942)

[ICM and InfoAsset combined database 176](#_Toc64014943)

[WS Pro database 178](#_Toc64014944)

# Disclaimer

Version 2021.1  
December 2020

*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

1. the user interface of InfoWorks ICM, InfoAsset Desktop and WS Pro
2. 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](file:///\\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:

1. Ones that get and set settings global to the application
2. Ones that create and open databases represented by the WSDatabase class
3. 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 thenselecting 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

1. 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'.
2. 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','pipe\_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.

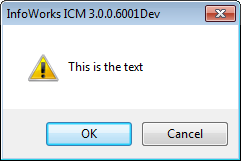
Icon – must be nil or one of the following strings: ‘!’, ‘?’, ‘Information’. If the parameter is nil then the ‘!’ icon is used.

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.

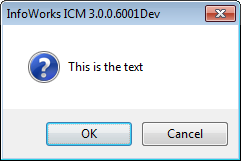
The method returns ‘Yes’,’No’,’OK’ or ‘Cancel’ as a string, unless cancel is hit and the ‘hard wire cancel’ parameter is set to true or nil in which case execution of the script is stopped.

The icons appear as follows:

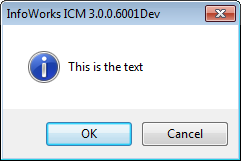
‘!’ (and nil)



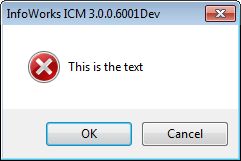
‘?’



‘Information’

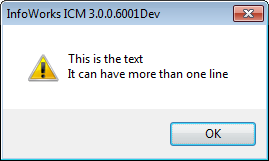


‘Stop’

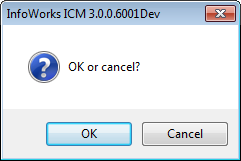


The buttons appear as follows:

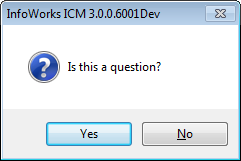
‘OK’



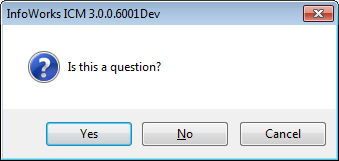
‘OKCancel’ (and nil)



‘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),DateTime.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',[DateTime.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'],

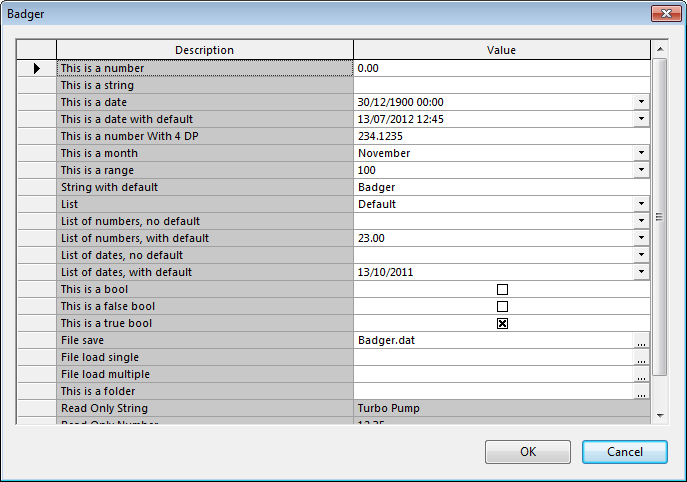
['Read Only Number','Readonly',12.3456678],

['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

1. The object as a WSModelObject (or class derived from it)
2. The ID of the object
3. 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

1. Creating an empty object and importing all the data items using this method
2. 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:

1. The ‘Working’ field must have been set to true
2. There must be no uncommented changes for the network for the run.
3. 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.inserted\_count},#{c.modified\_count},#{c.setting\_changed\_count},|#{c.user}|"

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.inserted\_count},#{c.modified\_count},#{c.setting\_changed\_count},|#{c.user}|"

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',['flooddepth','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)

This method behaves as run above, except that you may control (a) the server on which the simulation is run and (b) 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. The 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.

#### 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, the data will be updated otherwise it won’t | True |
| 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 tableswhere 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. e*xport\_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. e*xport\_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,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 downstream of channel links and maintain lag link data, otherwise set to false to maintain network connectivity by converting downstream lag links to channel links.

e.g.

model\_group.XPRAFTS\_import('d:\\temp\\1.xpx',true, false,'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 DurationType 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}"

1. 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\goat2.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

1. 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.
2. 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:\temp\\'+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:\temp\\'+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"

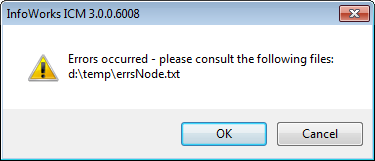
msg+=f

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:\temp\\'+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

Mechanism 2:

require 'FileUtils'

net=WSApplication.current\_network

configfile=configfile=File.dirname(WSApplication.script\_file)+'\\odicwithsource.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

Mechanism 3:

require 'FileUtils'

net=WSApplication.current\_network

configfile=configfile=File.dirname(WSApplication.script\_file)+'\\odicwithsource.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 |
| 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 |
| FireFlowData | FF |
| FireFlowData 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 |
| 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 |