



PIPESIM

STEADY-STATE MULTIPHASE FLOW SIMULATOR

PIPESIM 2017

Version 2017.1

PYTHON TOOLKIT USER GUIDE

Schlumberger

Copyright Notice

Copyright © 2017 Schlumberger. All rights reserved. This work contains the confidential and proprietary trade secrets of Schlumberger and may not be copied or stored in an information retrieval system, transferred, used, distributed, translated or retransmitted in any form or by any means, electronic or mechanical, in whole or in part, without the express written permission of the copyright owner.

Trademarks Service Marks

Schlumberger, the Schlumberger logotype, and other words or symbols used to identify the products and services described herein are either trademarks, trade names or service marks of Schlumberger and its licensors, or are the property of their respective owners. These marks may not be copied, imitated or used, in whole or in part, without the express prior written permission of Schlumberger. In addition, covers, page headers, custom graphics, icons, and other design elements may be service marks, trademarks, and/or trade dress of Schlumberger, and may not be copied, imitated, or used, in whole or in part, without the express prior written permission of Schlumberger. Other company, product, and service names are the properties of their respective owners.

PIPESIM® is a mark of Schlumberger.

An asterisk (*) is used throughout this document to designate other marks of Schlumberger.

Security Notice

The software described herein is configured to operate with at least the minimum specifications set out by Schlumberger. You are advised that such minimum specifications are merely recommendations and not intended to be limiting to configurations that may be used to operate the software. Similarly, you are advised that the software should be operated in a secure environment whether such software is operated across a network, on a single system and/or on a plurality of systems. It is up to you to configure and maintain your networks and/or system(s) in a secure manner. If you have further questions as to recommendations regarding recommended specifications or security, please feel free to contact your local Schlumberger representative.

The PIPESIM Python Toolkit includes Enthought Canopy, a Python development environment. You may also develop your own python code and run them on your local machine using the same Python environment. However, as a precaution, avoid writing any distributed applications accessible to external machines, or running any Python code from unknown sources to avoid any potential security risks.

CONTENTS:

1	Overview	1
1.1	Functionality	1
1.2	Prerequisites	1
1.3	Python	1
1.4	Enthought Canopy	2
1.5	Upgrading from OpenLink	2
1.6	Toolkit Stability	2
1.7	Licensing	2
2	Introduction	3
2.1	Components	3
2.2	Model Contexts	4
2.3	Model Context Nuances	6
2.4	Model Components	7
3	Quick Start	8
3.1	Open, Save and Close	8
3.2	Query the Model	9
3.3	Read and Write Parameters	9
3.4	Read and Write Data Tables	11
3.5	Add and Delete	11
3.6	Copy, Convert, and Rename	12
3.7	Connect and Disconnect	12
3.8	Manipulate Fluids	13
3.9	Import and Export Wells	16
3.10	Catalog and GIS	16
3.11	Simulation Settings	16
3.12	Simulations	18
3.13	Units of Measurement	20
3.14	Data Export	20
3.15	Batch Update	21
3.16	Utility Functions	21
4	Exploring the Model Further	23
5	Quick Reference	24
5.1	Model Class	24
5.2	About Class	25
5.3	Simulation Settings Class	26
5.4	Fluids Class	26
5.5	Compositional Fluid Class	27

5.6	Simulation Tasks Class	27
5.7	Definitions Sub-Module	28
6	Important Notes	29
6.1	Component Conversion	29
6.2	Component Copying	29
7	Working with Excel	30
7.1	Interacting with Excel	30
7.2	PIPESIM Tab Features	31
7.3	Embedded Python Scripts	32
7.4	Making Python Methods appear in Excel	34
7.5	Utility Functions	35
8	Example Code	38
8.1	Example Scripts	38
8.2	Case Studies	39
9	Troubleshooting	42
9.1	Python Toolkit fails to install	42
9.2	PIPESIM Tab does not appear in Excel	43
9.3	Integrated Excel script fails to run	46
10	Known Issues	47
10.1	Boundary Simplification Sources	47
10.2	Canopy Package Manager does not work	47
11	Reference Documentation	49
11.1	sixgill package	49
11.2	API Documentation	162

OVERVIEW

The PIPESIM Python Toolkit is a software development kit (SDK) for working with PIPESIM. It uses the Python programming language, one of the most common scripting languages. The SDK provides classes and methods for working with PIPESIM models so that typical PIPESIM tasks can be scripted and automated. It also includes integration with Microsoft Excel in a VBA style environment so that data can be shared easily between workbooks and PIPESIM models. With respect to PIPESIM Classic, PIPESIM Python Toolkit is the equivalent to the OpenLink programming interface.

1.1 Functionality

The features available through the PIPESIM Python Toolkit include:

- Add and delete model components
- Change and update the parameters of model components
- Convert Junctions to other model components such as Well and Source
- Import and export wells
- Perform tasks such as Network Simulation and PT Profile simulations

Note: While the intent is for the PIPESIM Python Toolkit to have the same features as the PIPESIM user interface, this is not yet the case. Some features may not be available and will be added in future releases.

1.2 Prerequisites

The requirements for using the PIPESIM Python Toolkit are:

- PIPESIM Release 2017.1 or higher must be installed.
- Python Toolkit must be installed. This option is available as part of the PIPESIM installer.
- PIPESIM and PIPESIM Python Toolkit licenses must be available.

While the PIPESIM Python Toolkit includes integration with MS Excel, you do not have to have Excel installed.

You should have some knowledge of PIPESIM and scripting or programming.

1.3 Python

Python is an open source scripting language available from several different distributions and well documented both online and in literature. It has many third party extensions, referred to as modules, which can be downloaded and

installed. It is one of the top ten languages in use around the world with active development and maintenance. The open source nature and large library of third party extensions makes it popular among the scientific community.

1.4 Enthought Canopy

The PIPESIM Python Toolkit is included as part of Canopy, the Python distribution from Enthought. You must install the specific Canopy release from Schlumberger as included in the PIPESIM installer. It is not possible to have multiple installations of Canopy on the same computer.

You may manually install the file into other Python packages such as Continuum Anaconda or CPython.

1.5 Upgrading from OpenLink

The PIPESIM Python Toolkit is the equivalent to the PIPESIM Classic OpenLink interface. It was designed and developed from the ground up as part of PIPESIM and is not a reimplement of the former interface. While both aim to provide programmatic ways of interacting with Python, they differ substantially in their methodology. As such there is no direct comparison between their respective classes and methods, though some key points are:

- OpenLink uses an object orientated representation of PIPESIM where model components and operations are operated on. Updating a value involves manipulating the objects and their properties. Data is always passed between the model and Excel as single values.
- PIPESIM Python Toolkit is an input/output (I/O) interface that sends and receives information about the model. Updating a value involves calling a function to set the component parameter to a value. Data is passed between the model and Excel as single values and as DataFrames. The toolkit may be used independently of Excel, for example, as scripts run from the desktop.

1.6 Toolkit Stability

The first release of the PIPESIM Python Toolkit is with PIPESIM 2017.1. Every aspect has been developed from scratch, including the underlying application programming interface (API), modules, classes and methods. While the validated in testing, feedback is key in driving future development. Changes to the way the interface works may be implemented based on the real world usage of the SDK. These changes will occur in a two year cycle starting with PIPESIM 2018.

1.7 Licensing

The PIPESIM Python Toolkit requires the PIPESIM Development SDK license option. Additional third party software is included in the PIPESIM Python Toolkit and is covered under this license. This software includes Canopy, PyXLL and EnPyXLL, as distributed by Enthought, Inc.

INTRODUCTION

The PIPESIM Python Toolkit is a collection of Schlumberger proprietary and publicly available Python modules. It is deployed through Canopy, a Python distribution from Enthought, a third party distributor. Canopy includes an integrated development environment (IDE) for Python and bundles together many data and scientific Python modules, such as Numpy, SciPy and pandas. You can easily download additional Python modules using the Canopy Package Manager.

This User Guide documents the usage of the PIPESIM Python Toolkit. It includes reference documentation for the Schlumberger proprietary modules with introductions to the publicly available modules. The publicly available Python modules are well documented both online and in literature. Links to existing Python documentation are included where applicable.

The key Python modules included in the PIPESIM Python Toolkit are:

- Sixgill (Proprietary): The primary Python programming classes and functions provided by the PIPESIM Python Toolkit.
- Manta (Proprietary): A low level Python interface with PIPESIM. Reference documentation is included but you are not expected to typically use this module.
- PyXLL: (Public): Pronounced 'pixel', it is a Python module for interfacing with Excel.
- xlwings: (Public): A Python module for interfacing with Excel.
- EnPyXLL: (Public): An extension to the PyXLL library.
- pandas: (Public): A Python module for working with data sets and arrays.

The Canopy IDE is provided as part of the installation and you can use any editor to develop scripts. For example, you may choose to edit scripts using Atom, Visual Studio, Visual Studio Code, PyCharm, NotePad or NotePad++.

You can run the PIPESIM Python Toolkit scripts from the desktop or you can embed them in Excel workbooks. If Excel is installed, a tab is added to the Toolkit ribbon during the installation process. The functionality provided by the ribbon is designed to replicate the VBA features and includes tools to edit, import/export, execute and assign scripts to command buttons.

2.1 Components

The PIPESIM Python Toolkit runs independent of the PIPESIM user interface (UI). It opens a PIPESIM model, referred to as a session, and starts interacting with the model. It is not possible to edit a model through both the PIPESIM UI and PIPESIM Python Toolkit simultaneously. The PIPESIM Python Toolkit can, however, open any number of sessions and work on many models at the same time.

The underlying technology for interacting with the PIPESIM model is the web API. This is a RESTful interface that allows third party vendors to integrate with PIPESIM. The PIPESIM Python Toolkit is a software development

kit (SDK) and you use it to interact with the PIPESIM web API. The web API implementation is proprietary to Schlumberger and is documented separately.

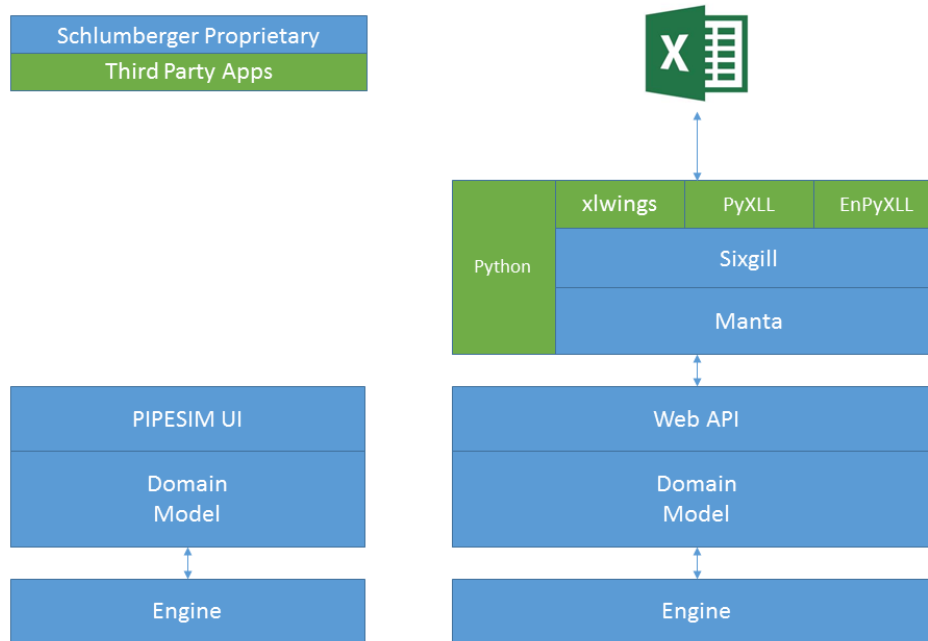


Figure: PIPESIM Python Toolkit Architecture

A PIPESIM Python Toolkit session can be considered an input/output (I/O) communications channel with a PIPESIM model. You can query the model for its contents and parameters; add, delete and modify the model components; and perform tasks on the model. The Sixgill module of the PIPESIM Python Toolkit is designed to easily perform these actions with a similar user perspective to the PIPESIM UI. The Manta module is a direct reflection of the underlying PIPESIM domain model and is more complex but provides more control.

The Sixgill module is implemented as a stable interface for working with PIPESIM. As PIPESIM evolves with new features and architecture improvements, the Sixgill module adapts and maintains a constant programming interface. This ensures that scripts are compatible across PIPESIM releases.

2.2 Model Contexts

The PIPESIM Python Toolkit has been developed around the concept of querying the model and performing actions on the data. The key points to this are:

- **Context:** This can be considered the area of interest when performing the query. It could be an area of the model (for example, Well-01), a set of similar components (for example, choke valves), or variations thereof (for example, all the choke valves on wells).
- **Parameter:** This is a property of a model component within the context. You do not need to consider which PIPESIM object the property belongs to, just that it is presented as part of the context. For example, you can search for bean sizes on wells, and the toolkit will query for chokes on wells, knowing that bean size is a choke valve property.
- **Value:** This is the value of the parameter. It could be a single value (for example float), or a table (for example, array).

The context method was designed to present the complex PIPESIM domain model in a simplified method similar to the user interface. In particular, there are domain aspects that exist in the data model but not in the UI which can be both confusing and add complexity. Through the context method, you do not need to consider on what object a parameter resides. You only need to present a query that identifies the parameter.

Context queries are built up using the model components. For example:

```
# Find all the wells
Well=ALL

# Find the well named Well01
Well='Well01'

# Find the item named CK1
Name='CK1'

# Find the flowline named FL-03
Flowline='FL-03'

# Find all the tubing in Well01
Well='Well01', Tubing=ALL
```

There are several ways that you can enter contexts.

2.2.1 Keyword Argument

Keyword Argument is the typical entry method where you enter the component type along with the search value (for example, Well='Well01'). You can chain keyword arguments together and enter several at once (for example, Well='Well01', Completion=ALL).

```
find(Well='Well01', Tubing=ALL)
```

2.2.2 Dictionary

Dictionary is the Python dictionary equivalent to the keyword argument, for example, {'Well':'Well01', 'Completion':ALL}. You can pass a dictionary in through the Python expansion, and a few methods can take dictionary directly:

```
find(**{'Well':'Well01', 'Completion':ALL})
```

2.2.3 String

You can uniquely identify individual components by a context string with a path style representation. This uses the colon delimiter to indicate the hierarchy in the model. For example, "Well01:VertComp" identifies the completion VertComp on Well01.

```
find(context="Well01:VertComp")
```

2.2.4 Model Context Object

The Sixgill core catalog uses the AbstractModelContext() class for cataloging and context manipulations. The model context object can be passed to any method that requires a context as an argument.

2.3 Model Context Nuances

The context is a powerful method for filtering and finding components through its different search methods.

The string context identifies one unique component in the model. Methods that use it will only manipulate that one component. It is therefore useful for specific updates to specific components, or for providing the specific context as a parameter. Examples are setting the fluid on a source, or making connections between components.

The keyword argument is the most versatile with identifying anything from the full model through to individual components. Each component in the model is tagged with keywords and the looking up the context returns all the components that are tagged with the keywords.

To find a specific named component, pass in the name:

```
>>> model.find(Name='Well')
```

It is recommended to use unique names for all top level items in the PIPESIM model. If there are several items named 'Well', it will list them all.

```
>>> model.find(Name='TopsyTurvy')
['TopsyTurvy', 'TopsyTurvy', 'TopsyTurvy']
```

To find the unique item, the component is specified:

```
>>> model.find(Name='TopsyTurvy', component=ModelComponents.PUMP)
['TopsyTurvy']
```

The difference between passing the 'Name' and component is also present for subsurface equipment. Any component in a well is considered part of the well context and is therefore returned:

```
>>> model.find(Well="ESP_Well")
[
    'ESP_Well',
    'ESP_Well:Casing',
    'ESP_Well:ESP_7',
    'ESP_Well:Packer',
    'ESP_Well:Source 1',
    'ESP_Well:Tubing',
    'ESP_Well:VW_7',
    'ESP_Well:VW_7:IPRBACKPRESSURE',
    'ESP_Well:VW_7:IPRDARCY',
    'ESP_Well:VW_7:IPRFETKOVITCH',
    'ESP_Well:VW_7:IPRFORCHHEIMER',
    'ESP_Well:VW_7:IPRHYDRAULICFRACTURE',
    'ESP_Well:VW_7:IPRJONES',
    'ESP_Well:VW_7:IPRPIModel',
    'ESP_Well:VW_7:IPRVERTICALPI',
    'ESP_Well:VW_7:VW_7:IPRVERTICALPI'
]
```

Whereas passing in the name just returns the well:

```
>>> model.find(Name="ESP_Well")
['ESP_Well']
```

This is useful for querying and manipulating parameters on a high level, without needing to specify the exact component:

```
>>> model.get_value(Well='ESP_Well', parameter='IPRModel')
'IPRHydraulicFracture'
```

Which could also be done through other more detailed contexts:

```
>>> model.get_value(Well='ESP_Well', Completion='VW_7', parameter='IPRModel')
>>> model.get_value(**{'Well':'ESP_Well', 'Completion':'VW_7'}, parameter='IPRModel')
>>> model.get_value('ESP_Well:VW_7', parameter='IPRModel')
```

2.4 Model Components

All the available model components can be found through the context. The `find(Name=ALL)` method provides a list of all of the components. The list includes the context for everything that is considered an object in PIPESIM, such as:

- Surface equipment such as pumps, separators, heat exchangers and flowlines.
- Subsurface items such as casings, completions and ESPs.
- Compositional fluid components, both predefined and psuedocomponents.
- Sources, sinks and junctions.

QUICK START

The following section provides a quick reference guide to the properties, methods and classes available in the PIPESIM Python Toolkit. Working example scripts can be found in the PIPESIM installation under the Python Toolkit folder.

The easiest way to investigate a model is through an interactive Python session. Code is executed as it is entered and the results are immediately seen.

From the Windows Start menu, open Enthought Canopy and start Canopy. Type the following commands into the interactive window to see the effect of performing them.

3.1 Open, Save and Close

Open a model:

```
>>> from sixgill.pipesim import Model
>>> model = Model.open('C:/Temp/examples/models/CSN_301_Small Network.pips')
```

Open a model and use the SI units of measurement for passing data:

```
>>> model = Model.open(
    filename='C:/Temp/examples/models/CSN_301_Small Network.pips',
    units=Units.SI)
```

Query details about the model:

```
>>> model.about()
```

Close the model:

```
>>> model.close()
```

Save the model:

```
>>> model.save()
```

Save the model as a new filename:

```
>>> model.save('C:/Temp/myproject/Modified Small Network.pips')
```

Create a new model:

```
>>> model = Model.new('C:/Temp/myproject/New Small Network.pips')
```

Open the current model in the UI: This will close the model in the Python Toolkit as the model cannot be edited at the same time in the Toolkit and UI:

```
>>> model.open_ui()
```

Open a specific model in the UI:

```
>>> Model.open_ui('C:/Temp/myproject/New Small Network.pips')
```

3.2 Query the Model

Operations on the model start by providing the context for the operation. The context defines what model components are then acted on. The simplest method is `find()` which returns a list of the model components within the context.

List out all the components in the model:

```
>>> model.find(Name=ALL)
```

Look for all the chokes in the model:

```
>>> from sixgill.definitions import *
>>> model.find(component=ModelComponent.Choke)
```

Alternatively, if we already know the choke keyword:

```
>>> model.find(Choke=ALL)
```

Find if an equipment exists by looking up its name:

```
>>> model.find(Name='PMP')
```

Find the tubing in Well_1:

```
>>> model.find(component=ModelComponent.Tubing, Well='Well_1')
```

The tubing can also be referenced by its context:

```
>>> model.find(context="Well_1:Tubing_1")
```

Find the context for all the completions on all the wells:

```
>>> model.find(Well=ALL, component=ModelComponents.COMPLETION)
```

Large models can be reduced into smaller models through a filtered model which contains just those items in the context. The filtered model has all the same methods available as the full model. For example, selecting a well to perform actions on:

```
>>> wellmodel = model.filter(Well="Well_1")
>>> wellmodel.find(component=ModelComponents.COMPLETION)
```

3.3 Read and Write Parameters

The parameters for the model components are accessed through the `get_value()` and `set_value()` methods to read and write values respectively. There are also `get_values()` and `set_values()` methods to read and write many parameter components at the same time.

Get the bean size of a choke:

```
>>> from sixgill.definitions import *
>>> beansize = model.get_value(Name = 'Choke', parameter = Parameters.Choke.BEANSIZE)
```

In this case the context for the choke is the same as its name, so alternatively:

```
>>> model.get_value('Choke', Parameters.Choke.BEANSIZE)
```

The Parameters static class defines the parameters as strings, and these can be used:

```
>>> model.get_value('Well_1:VertComp', 'GeometryProfileType')
```

Set the new bean size for the choke:

```
>>> model.set_value(Name = 'Choke', parameter = Parameters.Choke.BEANSIZE, value = 12)
```

Or simply, by using the context and argument ordering:

```
>>> model.set_value('Choke', Parameters.Choke.BEANSIZE, 12)
```

Parameters can be referenced from the top level context, so long as they are unique. To set a new reservoir pressure on Well_1 with only one completion:

```
>>> model.set_value(
    Well='Well_1',
    parameter='Parameters.Completion.RESERVOIRPRESSURE',
    value='3210')
```

Which is the equivalent to:

```
>>> model.set_value('Well_1:VertComp', 'ReservoirPressure', '3210')
```

To retrieve multiple parameters from multiple components, the get_values() method is used. For example to get specific parameters for the choke:

```
>>> model.get_values(
    Name='Choke',
    parameters=[Parameters.Choke.BEANSIZE,
                Parameters.Choke.CRITICALPRESSURERATIO])
```

This returns a dictionary of the components, each having a dictionary of the parameters. To retrieve the parameters from all the chokes:

```
>>> model.get_values(
    Choke=ALL,
    parameters=[Parameters.Choke.BEANSIZE,
                Parameters.Choke.CRITICALPRESSURERATIO])
```

Passing in an empty parameter list or not passing in a list returns all the parameters for the context. To get all the parameters for all the compressors in the model:

```
>>> params = model.get_values(Compressor=ALL)
```

To update the compressors with new parameters, pass in a dictionary of the compressor parameters:

```
>>> updated_params = {'CentComp': {
    'HonourStonewallLimit': True,
    'Route': CompressorThermodynamics.ADIABATIC,
    'Efficiency': 100.0,
    'IsActive': True,
    'PressureDifferential': 2400.0}
>>> model.set_values(updated_params)
```

The `set_value()` method sets the value for a single component and will throw an error if there is more than one component in the context. To set the same parameter value for all the items in the context, use the `set_all_value()` method:

```
>>> model.set_all_value(
    Choke=ALL,
    parameter="Beansize",
    value=2)
```

3.4 Read and Write Data Tables

Parameters such as well trajectories, flowline geometries, and geothermal surveys are handled as arrays. The data is passed to and from PIPESIM as pandas DataFrames so that it can be easily manipulated.

To read a flowline geometry using its context:

```
>>> model.get_geometry(context="F_10")
```

Or by passing the name as a context:

```
>>> model.get_geometry(Name="F_10")
```

The well trajectory is handled similarly as:

```
>>> model.get_trajectory(Name="GI_Well")
```

For a geothermal survey:

```
>>> model.get_geothermal_profile(Name="F_10")
```

There are corresponding methods for setting the data tables. The methods take the context and the DataFrame with the tabulated data. The method to set a flowline geometry is:

```
>>> model.set_geometry(context="F_20", dataframe=df)
```

Where `df` is the pandas DataFrame containing the tabulated flowline profile data. The DataFrame must be in the same format that the corresponding `get()` function uses, must have the same column headings, and must be indexed from 0.

For getting a pressure/flowrate curve on a source:

```
>>> model.get_pq_curve(Source="SRC1")
```

3.5 Add and Delete

Add a new source to the model:

```
>>> model.add(component=ModelComponents.SOURCE, name='Src1')
```

The parameters for the model component can be passed as an argument:

```
>>> model.add(
    ModelComponents.CHOKE,
    'CK1', context="Well 2",
    parameters=
    {
        Parameters.Choke.TOPMEASUREDDEPTH: 500,
```

```
Parameters.Choke.BEANSIZE:2,
Parameters.Choke.SUBCRITICALCORRELATION:SubCriticalFlowCorrelation.ASHFORD
})
```

For downhole equipment, pass in the context and aspects such as measured depth:

```
>>> model.add(
    component=ModelComponents.ESP,
    name="ESP1",
    context="Well 2",
    parameters=
    {
        Parameters.ESP.TOPMEASUREDDEPTH:1500,
        Parameters.ESP.OPERATINGFREQUENCY:60,
        Parameters.ESP.MANUFACTURER:"ALNAS",
        Parameters.ESP.MODEL:"ANA580",
        Parameters.ESP.NUMBERSTAGES:100,
        Parameters.ESP.HEADFACTOR:1,
        Parameters.ESP.POWERFACTOR:0.95,
        Parameters.ESP.USEVISCOSITYCORRECTION:True
    })
```

Delete a model component:

```
>>> model.delete(Name='Src1')
```

3.6 Copy, Convert, and Rename

Copy a well and include all the associated equipment:

```
>>> model.copy(context='Well_1', name='Well_2')
```

Convert a junction into a well:

```
>>> model.convert(
    context="Manifold",
    to_component=ModelComponents.WELL)
```

Rename a component by setting the Name parameter:

```
>>> model.set_value(
    context="Src1",
    parameter=Parameters.Source.NAME,
    'SOURCE_1')
```

3.7 Connect and Disconnect

PIPESIM uses ports to reference the inlet and outlet to components. To connect, specify the source (from) components, the destination (to) components, and the port to be connected where applicable.

To connect a well 'Well_1' to a flowline 'FL-1':

```
>>> model.connect('Well_1', 'FL-1')
```

Separators have two or three outlet ports hence outlet connections from a separator need to explicitly state which is used:


```
>>> model.connect(
    source='MBD101',
    destination='FL-PMP-1',
    source_port=Connection.Separator.TOP)
```

Disconnecting ports is done in the same way:

```
>>> model.disconnect(source='MBD101', destination='FL-PMP-1')
```

The model connections for a context are provided as a dictionary. This can be converted into a pandas DataFrame for convenient reviewing. Omitting the context will list all the connections in the model:

```
>>> model.connections(context='MBD-101')
>>> # Or to see it as a nicely formatted list
>>> import pandas as pd
>>> pd.DataFrame(model.connections(context='MBD-101'))
```

The `get_connections()` method provides a list of the connections for each item in the context. This method is easier to code in what is connected to a model component

```
>>> mbd101_connections = model.get_connections(context="MBD-101")
>>> mbd101_connections[Connections.SOURCE]
>>> mbd101_connections[Connections.Separator.TOP]
```

3.8 Manipulate Fluids

Black oil and compositional fluids are treated as any other model component. To select fluid type and compositional fluid details use a fluid method.

Create a black oil fluid:

```
>>> model.add(ModelComponents.BLACKOILFLUID, "BK111")
```

Set the properties of the black oil fluid using, the `set_value()` method, for example:

```
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.GOR,
    value=200)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.WATERCUT,
    value=5)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.GASSPECIFICGRAVITY,
    value=0.7)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.WATERSPECIFICGRAVITY,
    value=1.05)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.API,
    value=21)
```

Set calibration data for black oil fluid using, the `set_values()` method, for example:

```
>>> single_point_calibration = Parameters.BlackOilFluid.SinglePointCalibration
>>> model.set_values(
    {"BK111": {
        single_point_calibration.SOLUTIONGAS:
        BlackOilCalibrationSolutionGas.VASQUEZANDBEGGS,
        single_point_calibration.BUBBLEPOINTSATGAS_PRESSURE: 2424,
        single_point_calibration.BUBBLEPOINTSATGAS_TEMPERATURE: 250,
        single_point_calibration.BUBBLEPOINTSATGAS_VALUE: 202.63}})
```

Set the thermal data for black oil fluid:

```
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.ThermalData.GASHEATCAPACITY,
    value=0.50)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.ThermalData.OILHEATCAPACITY,
    value=0.40)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.ThermalData.WATERHEATCAPACITY,
    value=1.01)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.ThermalData.ENTHALPYCALCMETHOD,
    value=EnthalpyCalcMethod.METHOD2009)
>>> model.set_value(
    BlackOilFluid="BK111",
    parameter=Parameters.BlackOilFluid.ThermalData.LATENTHEATOFVAPORIZATION,
    value=141)
```

Get the parameterization of the black oil fluid:

```
>>> model.get_values(BlackOilFluid="BK111")
```

Select using compositional fluid rather than black oil:

```
>>> model.fluids.fluid_type = FluidType.COMPOSITIONAL
```

Set the compositional fluid options:

```
>>> model.fluids.compositional.pvt_package = PVTPackage.MULTIFLASH
>>> model.fluids.compositional.equation_of_state = EquationOfState.CUBICPLUSASSOCIATION
>>> model.fluids.compositional.salinity_model = SalinityModel.IONANALYSIS
```

Create a custom pseudocomponent:

```
>>> c7plus_component_params = {
    Parameters.FluidComponent.HYDROCARBONFORM: FluidComponentType.HYDROCARBON,
    Parameters.FluidComponent.MOLECULARWEIGHT: 234,
    Parameters.FluidComponent.TBOIL: 250,
}
>>> model.fluids.compositional.add_pseudocomponent(
    "C7Plus",
    parameters=c7plus_component_params)
```

Select compositional fluid components:

```
>>> selected_components = [
    MultiflashComponent.METHANE,
    MultiflashComponent.ETHANE,
    MultiflashComponent.HEXANE,
    MultiflashComponent.ETHANOL,
    MultiflashComponent.PROPANE,
    MultiflashComponent.WATER,
    "C7Plus",
]
>>> model.fluids.compositional.select_components(selected_components)
```

Create a compositional fluid with with salt component:

```
>>> fluid_parameters = {
    Parameters.CompositionalFluid.LIQUIDVISCOSITYCALC:
        EmulsionViscosityMethod.CONTINUOUSPHASE,
    Parameters.CompositionalFluid.USERWATERCUTCUTOFF:40,
    Parameters.CompositionalFluid.IONSODIUM:1000,
    Parameters.CompositionalFluid.IONCALCIUM:5000,
    Parameters.CompositionalFluid.IONMAGNESIUM:3000,
    Parameters.CompositionalFluid.IONPOTASSIUM:4012,
    Parameters.CompositionalFluid.IONSTRONTIUM:5000,
    Parameters.CompositionalFluid.IONBARIUM:6000,
    Parameters.CompositionalFluid.IONIRON:17000,
    Parameters.CompositionalFluid.IONCHLORIDE:1000,
    Parameters.CompositionalFluid.IONSULPHATE:1235,
    Parameters.CompositionalFluid.IONBICARBONATE:11122,
    Parameters.CompositionalFluid.IONBROMIDE:544,
    Parameters.CompositionalFluid.IONBARIUM:5555,
    Parameters.CompositionalFluid.SALTWATERDENSITYTYPE:SaltWaterDensity.DENSITY,
    Parameters.CompositionalFluid.SALTWATERDENSITY:72,
}
```

```
>>> model.add(ModelComponents.COMPOSITIONALFLUID, "Comp1", parameters=fluid_parameters)
```

The compositional fluid is read through the method on the fluids:

```
>>> model.fluids.compositional.get_composition("Comp1")
```

And the corresponding method for setting the composition:

```
>>> model.fluids.compositional.set_composition(df)
```

Assign a black oil fluid to a well:

```
>>> model.set_value(
    Well = "Well 2",
    parameter=Parameters.Well.ASSOCIATEDBLACKOILFLUID,
    value="BK111")
```

Assign a compositional fluid to a well:

```
>>> model.set_value(
    Well = "Well 2",
    parameter=Parameters.Well.ASSOCIATEDCOMPOSITIONALFLUID,
    value="COMPl")
```

3.9 Import and Export Wells

You can import and export Wells to/from individual PIPESIM models. To export a well to a PIPESIM model:

```
>>> model.export_well(context="Well_1",
    folder="C:/temp/myproject/wells")
```

To export all the wells, leave out the context:

```
>>> model.export_well(folder="C:/temp/myproject/wells")
```

If the folder is not provided then it will export to the same folder as the currently open PIPESIM model.

Import a PIPESIM well model into the currently open model:

```
>>> model.import_well(filename="C:/temp/myproject/Well.pips", name="Well_1")
```

To use the fluid from the imported model:

```
>>> model.import_well(
    filename="C:/temp/myproject/Well.pips",
    name="Well_1", fluid_override=False)
```

To overwrite the existing “Well_1” in the current model with the imported well:

```
>>> model.import_well(
    filename="C:/temp/myproject/Well.pips",
    name="Well_1", overwrite_existing=True)
```

3.10 Catalog and GIS

Get the elevation for a coordinate (latitude, longitude):

```
>>> model.elevation(lat=44.4532, long=76.16231)
```

Reading entries from the catalog. Parameters required for the look-up such as Manufacturer, series, etc. must have been set on the component first:

```
>>> model.read_catalog("PCP")
```

3.11 Simulation Settings

You can operate on the simulation settings through both a dictionary and individual properties.

Get the current simulation settings as a dictionary:

```
>>> model.sim_settings()
```

Get the ambient temperature:

```
>>> model.sim_settings.ambient_temperature
```

Set the ambient temperature:

```
>>> model.sim_settings.ambient_temperature = 70
```

Alternatively, using the dictionary syntax:

```
>>> model.sim_settings[Parameters.SimulationSetting.AMBIENTTEMPERATURE] = 70
```

The global flow correlations are provided as a dictionary by:

```
>>> model.sim_settings.global_flow_correlations()
```

And to set the global flow correlations, pass back in a dictionary:

```
>>> multiphase = Parameters.FlowCorrelation.Multiphase
>>> model.sim_settings.global_flow_correlations({
    multiphase.Vertical.SOURCE:
        Constants.MultiphaseFlowCorrelation.BakerJardine.BEGGSBRILL,
    multiphase.Horizontal.SOURCE:
        Constants.MultiphaseFlowCorrelation.BakerJardine.BEGGSBRILL
})
```

To use the local flow correlations:

```
>>> model.sim_settings.use_global_flow_correlations = False
```

The local flowline correlations are accessed through context based get/set methods.

```
>>> model.sim_settings.get_flow_correlations(Flowline=ALL)
```

If the context is left empty then it returns all the flowline and well trajectories that be specified:

```
>>> model.sim_settings.get_flow_correlations()
```

To set the flow correlations for a flowline, pass in the dictionary of parameters for the flowlines or wells to be set:

```
>>> multiphase = Parameters.FlowCorrelation.Multiphase
>>> model.sim_settings.set_flow_correlations({
    "FL_10": {
        Parameters.FlowCorrelations.OVERRIDEGLOBAL: True,
        multiphase.Vertical.SOURCE:
            Constants.MultiphaseFlowCorrelation.BakerJardine.BEGGSBRILL,
        multiphase.Horizontal.SOURCE:
            Constants.MultiphaseFlowCorrelation.BakerJardine.BEGGSBRILL
    }})
```

The heat transfer options use the same methodology. To get the global heat transfer options:

```
>>> model.sim_settings.global_heat_transfer_options()
```

To set the heat transfer options:

```
>>> model.sim_settings.global_heat_transfer_options({
    Parameters.HeatTransferOption.PIPEBURIALMETHOD: Constants.PipeBurialMethod.METHOD2009})
```

To get and set the local heat transfer options:

```
>>> model.sim_settings.get_heat_transfer_options(Flowline=ALL)
>>> model.sim_settings.set_heat_transfer_options({
    "FL_10": {
        Parameters.HeatTransferOptions.OVERRIDEGLOBAL: True
        Parameters.HeatTransferOptions.PIPEBURIALMETHOD: Constants.PipeBurialMethod.METHOD2009
    }})
```

The list of available simulation setting properties is provided in the Quick Reference section below.

3.12 Simulations

Simulations are performed through the tasks class. Each task has the same core methods available, with additional methods depending on the task being performed.

Get the current boundary conditions for a network simulation:

```
>>> model.tasks.networksimulation.get_conditions()
```

For a specific study:

```
>>> model.tasks.networksimulation.get_conditions(study="Late Field Life")
```

Update the boundary conditions for the default network simulation study:

```
>>> boundaries =
    { "Well:VertComp":
      {
        Parameters.Boundary.PRESSURE:NAN,
        Parameters.Boundary.TEMPERATURE:150,
        Parameters.Boundary.FLOWRATETYPE:FlowRateType.LIQUIDFLOWRATE,
        Parameters.Boundary.LIQUIDFLOWRATE:200
      }
    }
>>> model.tasks.networksimulation.set_conditions(boundaries = boundaries)
```

To reset the boundary conditions to those of the model for the default study:

```
>>> model.tasks.networksimulation.reset_conditions()
```

To run the network simulation study:

```
>>> system_variables = [
    SystemVariables.PRESSURE,
    SystemVariables.TEMPERATURE,
    SystemVariables.VOLUMEFLOWRATELIQUIDSTOCKTANK,
    SystemVariables.VOLUMEFLOWRATEOILSTOCKTANK,
    SystemVariables.VOLUMEFLOWRATEWATERSTOCKTANK,
    SystemVariables.VOLUMEFLOWRATEGASSTOCKTANK,
    SystemVariables.BOTTOMHOLEPRESSURE,
    SystemVariables.OUTLETGLRSTOCKTANK,
    SystemVariables.OUTLETWATERCUTSTOCKTANK,
]
>>> profile_variables = [
    ProfileVariables.TEMPERATURE,
    ProfileVariables.ELEVATION,
    ProfileVariables.TOTALDISTANCE,
]
>>> results = model.tasks.networksimulation.run(
    system_variables=system_variables,
    profile_variables=profile_variables)
```

The study conditions can be passed into the run_simulation() method. This allows simulations to be queued up, each with their own study conditions:

```
>>> results = model.tasks.networksimulation.run(
    boundaries=boundaries,
    system_variables=system_variables, profile_variables=profile_variables)
```

Predefined results variable lists are available as available in the PIPESIM UI:

```
>>> results = model.tasks.networksimulation.run(
    boundaries=boundaries,
    system_variables=OutputVariables.System.FLOW_ASSURANCE,
    profile_variables=OutputVariables.Profile.FLOW_ASSURANCE)
```

The simulation results is an object with three properties: system, node and profile. For network simulations, the node and system results are dictionaries of each node or system point respectively while the profile is a dictionary of each branch. In a PT Profile simulation the system points is also a dictionary of the sensitivities.

The node results for a network simulation as a dataframe:

```
>>> pd.DataFrame(results.node)
```

The system results for a network simulation as a dataframe

```
>>> pd.DataFrame(results.system)
```

The profile results for flowline FL-45 in a network simulation as a DataFrame:

```
>>> pd.DataFrame(results.profile["FL-45"])
```

The run() method executes concurrently and the results are returned when it finishes. To run the simulation in the background, use the start_simulation() method. This allows the Python script to continue executing while the simulation is running:

```
>>> simulation_id = model.tasks.networksimulation.start(
    profile_variables=profile_variables,
    system_variables=system_variables)
```

In this case, the status of the simulation is returned by:

```
>>> model.tasks.networksimulation.get_state(simulation_id)
```

And when finished, the simulation message and results are retrieved:

```
>>> message = model.tasks.networksimulation.get_messages(simulation_id)
>>> results = model.tasks.networksimulation.get_results(simulation_id)
```

Note that the simulations using start() are queued by the PIPESIM engine and are run concurrently. The simulations are not run in parallel, only that the Python script can continue running while each simulation finishes.

PT Profile simulations are performed in the same manner. The arguments passed to the study and simulation methods are accordingly different:

```
>>> parameters = {
    "StartNode": "Well",
    "InletPressure": 2600, #psia
    "OutletPressure": 200, #psia
    "LiquidFlowRate": 3000, #stb/d
    "FlowRateType": "LiquidFlowRate",
    "CalculationVariableType": "calcCustom",
    "CalculateVariable": {
        "Component": "FL-1",
        "Variable": "InnerDiameter",
        "MinValue": 1,
        "MaxValue": 10,
    }
}
>>> results = model.tasks.ptprofilesimulation.run(
    parameters=parameters,
```

```
profile_variables=profile_variables,
system_variables=system_variables)
```

For a PT Profile sensitivity study:

```
>>> sensitivities={
    "Variable": "VertComp/ReservoirPressure",
    "Values": [2000, 3000, 4000] #psia
}
>>> parameters = {
    "StartNode": "Well",
    "InletPressure": 2600, #psia
    "OutletPressure": 200, #psia
    "CalculationVariableType": "calcFlowrate",
    "FlowRateType": "LiquidFlowRate",
}
>>> results = model.tasks.ptprofilesimulation.run(
    parameters=parameters,
    system_variables=system_variables,
    profile_variables=profile_variables,
    sensitivities=sensitivities)
```

With a PT Profile simulation, the list of available components and variables for a producer is found from:

```
>>> model.tasks.ptprofilesimulation.get_sensitivity_variables("Well_1")
```

Generate PIPESIM engine files in the current folder:

```
>>> model.tasks.networksimulation.generate_engine_files(study="Study 1")
```

To specify the folder for the engine files:

```
>>> model.tasks.ptprofilesimulation.generate_engine_files(
    study="Study 1",
    folder='c:/temp/myproject')
```

The method returns the list of files that has been generated.

3.13 Units of Measurement

Get the engineering unit and the symbol for a parameter:

```
>>> model.describe(context="Choke", parameter="Beansize").units
>>> model.describe(context="Choke", parameter="Beansize").units_symbol
```

Get the engineering unit conversion for a parameter:

```
>>> model.describe(context="Choke", parameter="Beansize").si_scale
>>> model.describe(context="Choke", parameter="Beansize").si_offset
```

3.14 Data Export

The model parameters can be exported to a csv file. Note that this does not include the tabulated data such as trajectories at this time:


```
>>> model.export_parameters(filename="C:/Temp/myproject/ExportedParams.csv")
```

3.15 Batch Update

Operations such as get/set are performed immediately and independently on the model. Each command must finish before the next one can start. There is an overhead on performing each operation through the underlying web API.

The batch update method caches the operations and perform them as a single operation on the web API. This provides improved performance over doing individual operations.

The syntax for using the batch operation is:

```
>>> with model.batch_update():
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.GOR,
        value=200)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.WATERCUT,
        value=5)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.GASSPECIFICGRAVITY,
        value=0.7)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.WATERSPECIFICGRAVITY,
        value=1.05)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.API,
        value=21)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.FRACTIONCO2,
        value=0.05)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.FRACTIONH2S,
        value=0.001)
    model.set_value(
        BlackOilFluid="BK111",
        parameter=Parameters.BlackOilFluid.FRACTIONN2,
        value=0.1)
```

3.16 Utility Functions

The following utility functions are included in the PIPESIM Python Toolkit.

Method	Description
convert_tubing_bmd()	Convert bottom measured depth to top measured depth and length.

3.16.1 Convert Bottom Measured Depth

The PIPESIM data model uses top measured depth (TMD) and length as the parameters for tubing, casing and liners. Data is often provided, however, as bottom measured depth (BMD). The method converts a list of BMD to a dictionary of TMD and length for setting as parameters.

```
>>> from sixgill.utilities import convert_tubing_bmd
>>> tmd_param = convert_tubing_bmd([2000.0, 2400.0, 2800.0, 3500.0])
>>> model.add(ModelComponents.CASING, "Csg1", context="Well1", \
    parameters=tmd_param[0])
>>> model.add(ModelComponents.Tubing, "Tb1", context="Well1", \
    parameters=tmd_param[1])
```

EXPLORING THE MODEL FURTHER

The Python Toolkit provides complete access to the underlying data model through the function calls. In many areas there are several ways to access the same data.

For instance, to use the surface conditions of a well for the boundaries:

```
>>> model.tasks.networksimulation.use_surface_conditions = True
```

This is also available through:

```
>>> model.set_value(component=ModelComponents.NETWORKSIMULATION, \
    Study='Study 1',
    parameter=Parameters.NetworkSimulation.USESURFACEBOUNDARYCONDITIONS,
    value = True)
```

QUICK REFERENCE

5.1 Model Class

5.1.1 Properties

Property	Description
api	PipesimSession. The web API object connected to PIPESIM
context	AbstractModelContext. The full (root) context of the model
is_open	Boolean. Whether the model is open (True) or not (False)
filename	String. The full path and filename of the model.

5.1.2 Methods

Method	Description
about()	Return the details about the model and PIPESIM version
add()	Add a component to the model
batch_update()	Cache the model updates and apply as a single operation
close()	Close the model
connect()	Connect two model components
connections()	List the connections in the model
convert()	Convert a component to a different type
copy()	Copy a model component
delete()	Delete a component from the model
delete_simulation()	Delete the simulation results
describe()	Get the details of a parameter, such as units of measurement
disconnect()	Disconnect two model components
elevation()	Get the elevation for a latitude / longitude location
export_well()	Export a well to a PIPESIM model file
export_parameters()	Export the model parameters to a csv file
filter()	Filter the model for a specific context
find()	Find components in the model
get_completion_test_points()	Get the completion test points
get_geometry()	Get the flowline geometry
get_geothermal_profile()	Get the geothermal profile
get_pq_curve()	Get the well performance curve
get_trajectory()	Get the well trajectory
get_value()	Get a parameter from a model component

Continued on next page

Table 5.1 – continued from previous page

Method	Description
get_values()	Get parameters from model components
import_well()	Import a well from a PIPESIM model
open()	Open a PIPESIM model
open_ui()	Open a PIPESIM model in the PIPESIM UI
read_catalog()	Update the component parameters from the catalog
save()	Save the PIPESIM model
set_all_value()	Set the parameter for all the model components
set_completion_test_points()	Set the completion test points table
set_geometry()	Set the flowline geometry
set_geothermal_profile()	Set the geothermal profile
set_pq_curve()	Set the well performance curve
set_trajectory()	Set the well trajectory
set_value()	Set a parameter on a model component
set_values()	Set a series of parameters on one or more model components
sim_settings()	Get the simulation settings

5.2 About Class

The properties and methods are accessed by prefixing with *model*.about.

5.2.1 Properties

Property	Description
filename	String. The full path and filename of the model.
toolkit_version	String. The release number of the PIPESIM Python Toolkit.
units_system	String. The units of measurement system.
webapi_version	String. The PIPESIM Web API release number

5.2.2 Methods

There are no methods on the About() class.

5.3 Simulation Settings Class

5.3.1 Properties

Property	Description
ambient_temperature	Float. The ambient temperature
atmospheric_pressure	Float. The atmospheric pressure.
corrosion_efficiency	Float. The corrosion efficiency
corrosion_model	String. The corrosion model (CorrosionModel static class)
erosional_velocity_constant	Float. The erosional velocity constant.
wind_speed	Float. The wind speed.
soil_type	String. The soil type (SoilType static class).
soil_conductivity	Float. The soil conductivity
metocean_data_location	String. The Metocean data location. (MetoceanDataLocation static class).
max_report_interval_length	Float. The maximum report interval length for pipe segmentation.
print_computation_segment_result	Boolean. Print the computational segment result for pipe segmentation.
max_computation_segment_length	Float. The maximum computational segment length for pipe segmentation.
network_solver_method	String. The network solver method (NetworkSolverMethod static class).
network_solver_tolerance	Float. The network solver tolerance.
network_solver_max_iterations	Integer. The network solver maximum number of iterations.
single_branch_keywords	String. Single branch engine keywords.
network_keywords_top	String. Network engine top keywords.
network_keywords_bottom	String. Network engine bottom keywords.
gis_elevation_interval	Float. GIS elevation interval.
gis_elevation_max_points	Integer. GIS elevation maximum number of points.
gis_elevation_data_source	String. GIS elevation data source (GISDataSource static class).
use_global_flow_correlation	Boolean. Use the global flow correlations.
use_global_heat_transfer	Boolean. Use the global heat transfer settings.

5.3.2 Methods

There are no methods on the SimulationSettings() class. The `model.sim_settings()` can return a dictionary, or be treated as a dictionary. See the examples under the Quick Start section.

5.4 Fluids Class

The properties and methods are accessed by prefixing with `model.fluids`.

5.4.1 Properties

Property	Description
fluid_type	The type of fluid used (FluidType static class).

5.4.2 Methods

There are no methods on the fluids class.

5.5 Compositional Fluid Class

The properties and methods are accessed by prefixing with *model.fluids.compositional*

5.5.1 Properties

Property	Description
equation_of_state	String. Equation of state (EquationOfState static class).
pvt_package	String. PVT Package (PVTPackage static class).
salinity_model	String. Salinity model (SalinityModel static class).
acf_correlation	String. ACF Correlation (ACFCorrelationMethod static class).
critical_property_correlation	String. Critical Property Correlation (CriticalPropertyCorrelationMethod class).
thermal_coefficient_correlation	String. Thermal Coefficient Correlation (ThermalCoefficientCorrelationMethod class).

5.5.2 Methods

Property	Description
add_pseudocomponent()	Add a pseudocomponent.
delete_pseudocomponent()	Delete a pseudocomponent.
select_components()	Select components for the compositional fluid.
unselect_components()	Unselect components for the compositional fluid.

5.6 Simulation Tasks Class

The properties and methods are accessed by prefixing with *model.tasks*. There is a class for each of the supported tasks.

5.6.1 Classes

Class	Description
networksimulation	Methods for running network simulation studies.
ptprofilesimulation	Methods for running PT profile simulation studies.

5.6.2 Methods

Property	Description
<code>delete_results()</code>	Delete the simulation results.
<code>generate_engine_files()</code>	Generate the engine files for a study
<code>get_calculated_variables()</code>	PT Profile. Get the available calculated variables.
<code>get_conditions()</code>	Get the study boundary conditions.
<code>get_messages()</code>	Get the simulation messages of the simulation.
<code>get_results()</code>	Get the simulation results.
<code>get_sensitivity_variables()</code>	PT Profile. Get the available sensitivity variables.
<code>get_state()</code>	Get the running state of the simulation.
<code>get_use_surface_conditions()</code>	Get whether the model is using surface conditions.
<code>reset_conditions()</code>	Reset the study boundary conditions.
<code>run()</code>	Run a simulation task and wait for it to finish.
<code>set_conditions()</code>	Set the study boundary conditions.
<code>set_use_surface_conditions()</code>	Set whether the model is using surface conditions.
<code>start()</code>	Start a simulation task.

5.7 Definitions Sub-Module

The definitions sub-module within Sixgill provides static classes for the model operations and parameterization. The enumerations of parameters such as `EmulsionViscosityMethod.CONTINUOUSPHASE` translating to ‘Continuous-Phase’ is provided for autocompletion and code editing intellisense.

5.7.1 Static Classes

Static Class	Description
<code>Connection</code>	Model component connection ports.
<code>Constants</code>	String enumerations used by the PIPESIM data model.
<code>ModelComponents</code>	Model components.
<code>OutputVariables</code>	Predefined groups of output variables.
<code>Parameters</code>	Parameters for each model component.
<code>ProfileVariables</code>	Output profile variables.
<code>SimulationState</code>	Simulation states.
<code>SystemVariables</code>	Output system variables.
<code>Tasks</code>	Tasks that can be performed on the model.
<code>Units</code>	Available unit systems when opening a model.

5.7.2 Special Keywords

The following special string definitions are provided as global variables:

Global Variable	Description
<code>NAN</code>	Mathematical operator “Not a Number” (NaN).
<code>ALL</code>	Match all the items in the context.
<code>NONE</code>	Match none of the items in the context.
<code>TEMPLATE</code>	Match template items in the context.
<code>GLOBAL</code>	The global setting (heat transfer and flow correlations).
<code>DEFAULT</code>	The default setting (heat transfer and flow correlations).

IMPORTANT NOTES

6.1 Component Conversion

The Python Toolkit only supports converting junctions, which can only be converted to a Well, Source or Sink.

6.2 Component Copying

The `copy()` method only works on wells at this time.

WORKING WITH EXCEL

Python has many libraries for working with Excel. Included with the PIPESIM Python Toolkit are the following packages that each bring various features:

- `pyxll` provides excellent features with user defined functions.
- `xlwings` makes reading and writing Excel worksheets easy.
- `enpyxll` provides features for integrated workbooks.

Python scripts and interactive Python can interact with Excel workbooks directly. In addition, they can be included in the Excel workbooks in a manner similar to VBA. Scripts can read and write data, format sheets, and be assigned to VBA features such as buttons.

7.1 Interacting with Excel

Reading and writing Excel ranges is done through the `xlwings` Python module, often abbreviated to `xw`. The following examples are provided for information with many more methods and features available in `xlwings`. See the [xlwings documentation](#) for more help.

Reading and writing tabulated data formats are subject to the following considerations:

- Pandas DataFrames can be written to Excel.
- Lists of lists or dictionaries cannot be written to Excel (as of `xlwings` 0.10.2).
- To read tabulated data formats from Excel you must use the utility functions provided in the PIPESIM Python Toolkit (see section below) for converting the lists of lists into DataFrames and dictionaries.

The PIPESIM Python Toolkit allows scripts to be embedded in workbooks, but any script can interact with Excel workbooks. The interactive Python command line is typically the easiest way to try out code.

Open a new workbook:

```
>>> import xlwings as xw
>>> app = xw.App()
```

Current workbook and worksheet:

```
>>> bk = xw.books.active
>>> sht = bk.sheets.active
```

Activate a worksheet:

```
>>> bk.sheets('Sheet1').activate()
```

Writing to a cell:

```
>>> xw.Range('A1').value = 'Hello world'
```

Reading from a cell:

```
>>> value = xw.Range('A1').value
```

Writing a DataFrame to a cell

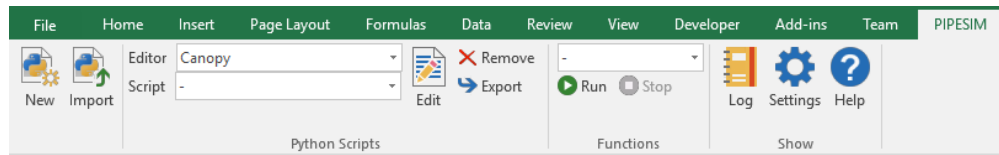
```
>>> xw.Range('B4').value = geometry_df
```

Reading a range and converting to DataFrame

```
# Automatically determine the table from the top left cell
values = xw.Range('B4').expand().value
# Or specify the top left and bottom right cells
values = xw.Range('B4', 'G7').value
from sixgill.utilities import range_to_dataframe
df = range_to_dataframe(values)
```

7.2 PIPESIM Tab Features

The PIPESIM tab on the Excel ribbon provides development environment features for working with Python and Excel.



- Add
 - New: Create a new script within the Excel workbook.
 - Import: Import an existing Python script to the Excel workbook.
- Scripts
 - Editor: Select which editor to use.
 - Script: Select a Python script from the workbook.
 - Edit: Edit the selected script.
 - Remove: Delete the selected script from the workbook.
 - Export: Export the selected script to a Python file.
- Functions
 - Select: Select the function to run.
 - Run: Run the selected function.
 - Stop: Stop running the selected function.
- Show
 - Log: Display the Python messages console.
 - Settings: Change the Python editor settings.

- Help: Open the PIPESIM Python Toolkit help documentation in a web browser.

7.3 Embedded Python Scripts

You can embed Python scripts in Excel workbooks in a way similar to embedding Visual Basic scripts. You can edit the scripts in a text editor of the your choice. You can run the macros from the PIPESIM tab, from an interactive Python session, or from objects, such as buttons, associated with actions in the workbook.

7.3.1 Visual Basic and Python Feature Comparison

The key similarities and differences between VBA and Python macros are:

Feature	Visual Basic	Python
Workbook Suffix	Saved as macro workbooks (.xlam).	Saved as standard workbook (.xlsx).
Save	Macros are automatically saved when Excel is saved.	Scripts need to be saved before Excel is saved.
Editor	Edited within the VBA Editor	Any editor can be used.

7.3.2 Add a Script to a Workbook

To add a new script to a workbook:

1. On the PIPESIM tab in the Add group, click the New Script icon.
2. Enter the script name.
3. Click OK.

The script is created in the workbook and opened for editing in the currently selected editor.

7.3.3 Import a Script into a Workbook

To import an existing Python script into a workbook:

1. On the PIPESIM tab in the Add group, click Import.
2. Select the script from the browser.
3. Click Open.

The script is imported into the workbook.

7.3.4 Edit a Python script

To edit an embedded script:

1. On the PIPESIM tab in the Scripts group, select the script from the Script dropdown menu.
2. Click Edit. The script opens in the currently selected editor.
3. When your edits are complete, click Save. Excel does not detect unsaved changes in scripts. You *must* save the script in the editor before saving the Excel file.

7.3.5 Run a Python Function

On the PIPESIM tab in the Functions group, you can run and stop macros. Methods that have been assigned are in the dropdown list. See “Assigning Functions to the PIPESIM tab” below for more details: - Click Run to run the selected macro. - Click Stop to stop the selected macro.

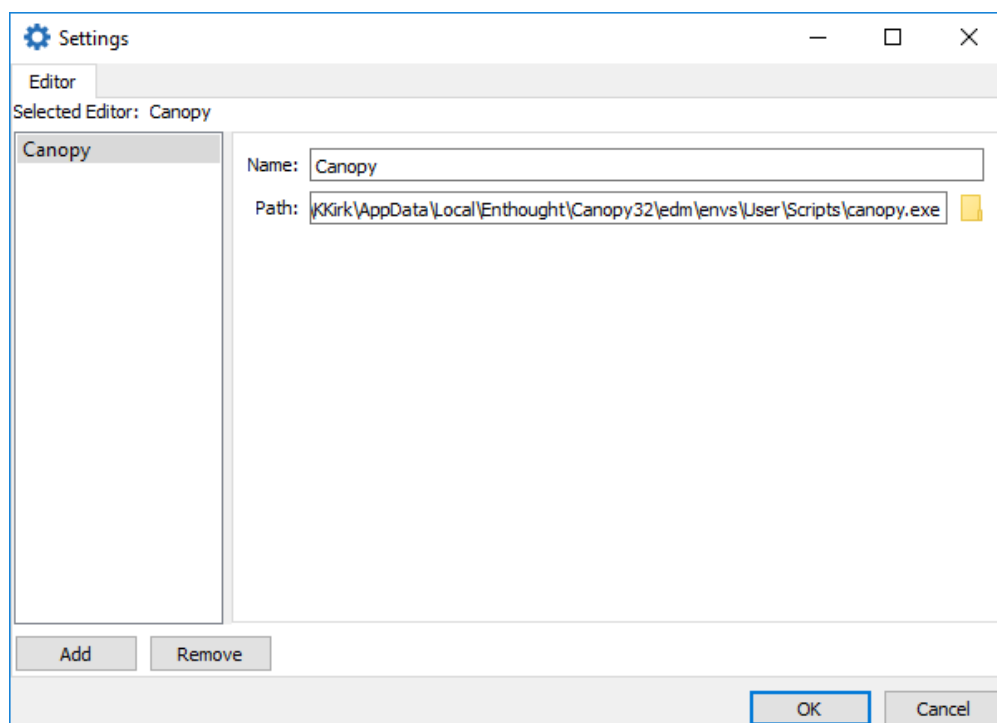
7.3.6 Display the Python messages console

On the PIPESIM tab in the Show group, click Log. The Log Viewer shows one tab per open workbook, with each tab showing the log messages issued by the scripts in the workbook. An additional tab shows the pyxll error log that may contain further information about running the embedded scripts.

To add custom log messages within scripts see the Logging section below.

7.3.7 Change the Python editor settings

The default editor for editing the embedded scripts is Canopy.



To change the current editor:

1. On the PIPESIM tab in the Show group, click Settings and select the Editor tab.
2. Click Add.
3. Browse to the editor application file, select it and click Open.
4. Enter the name of the Editor.
5. Click OK.

To change the default editor:

1. On the PIPESIM tab in the Show group, click Settings and select the Editor tab.
2. From the list of editors, highlight the editor to be the default.
3. Click OK.

7.3.8 Open the toolkit help in a web browser

To open the PIPESIM Python Toolkit help documentation in a web browser, on the PIPESIM tab in the Show group, click Help.

The “excel_macros_functions.xlsx” example found under the Python Toolkit examples under the PIPESIM installation folder, has examples of the following features.

7.4 Making Python Methods appear in Excel

The Python methods and messages in the embedded scripts are made available in Excel by applying Python decorators.

7.4.1 Designate a Python method as a VBA macro

You can designate a Python method as a VBA macro. In the script, enter the ‘@xl_macro’ decorator on the line preceding the definition of the method. The macro can be associated with actions on the worksheets such as buttons. The syntax is:

```
from pyxll import xl_macro
@xl_macro
def open_model():
    """ Open the model """
    mymodel = Model.open('C:/Temp/Test/MyModel.pips')
```

Within Excel, the macro can be associated with a button, for example.

7.4.2 Designate a Python method as an Excel custom function

You can designate a Python method as a custom function in Excel. In the script, enter the ‘@xl_func’ decorator on the line preceding the definition of the method. The method can then be used as a function in an Excel cell.

```
from pyxll import xl_func
@xl_func
def MultiplyByTwo(x):
    """ Multiply the number entered by two """
    result = x * 2
    return result
```

Within an Excel cell, the function can be called by ‘=MultiplyByTwo()’.

7.4.3 Assigning Functions to the PIPESIM tab

You can assign a Python method to appear in the dropdown list in the Functions group of the PIPESIM tab. Selecting the function and clicking Run will run the function. In the script, enter the @entry_point decorator on the line preceding the definition of the method.

```

from enpyxll import entry_point
@entry_point
def RandomList():
    import xlwings as xw
    import random
    randlist = [random.randint(0,1000) for r in range(10)]
    xw.Range("B9").value = randlist

```

7.4.4 Send errors to the Log Viewer

You can send errors to the Log Viewer. In the script, enter the '@log_error' decorator on the line preceding the definition of the method.

```

import logging
logger = logging.getLogger(__name__)

from enpyxll.util.logs import log_error
@log_error
def open_model():
    """ Open the model """
    logger.info("Opening the model...")
    mymodel = Model.open('C:/Temp/Test/MyModel.pips')

```

If there are several decorators this **must** be the one immediately preceding the def() statement, i.e. the last one listed:

```

@xl_macro
@log_error
def open_model():
    logger.info("Opening the model...")
    mymodel = Model.open('C:/Temp/Test/MyModel.pips')

```

7.5 Utility Functions

The PIPESIM Python Toolkit includes utility functions for working with Excel.

Method	Description
active_sheet()	Return the currently active Excel worksheet
current_folder()	Return the folder (path) to the currently open Excel workbook.
get_model_session()	Get the model associated with the workbook, opening it if necessary
range_to_dataframe()	Convert an Excel range to a Pandas DataFrame
range_to_dictionary()	Convert an Excel range to a dictionary
worksheet_last_row()	Return the last row of the worksheet
worksheet_last_column()	Return the last column of the worksheet

7.5.1 Returning the currently active Excel worksheet

To return the currently active sheet object, follow this example:

```

>>> from sixgill.utilities import active_sheet
>>> sht = active_sheet()

```

7.5.2 Returning the Excel workbook folder

The Python variable `__file__` returns the script location, but this is not the same as the Excel workbook folder. The `current_folder()` method returns the folder of the Excel workbook.

```
>>> from sixgill.utilities import current_folder()
>>> workbook_folder = current_folder()
```

7.5.3 Get the model associated with the workbook

Where spreadsheets have several functions that operate on models, the `get_model_session()` method can handle connections to the required models. It will open the model, if necessary, or simply pass back the model object if it is already open.

In the following example, a user can run `readparameters()`, `writeparameters()` and `savemodel()` and the opening of the model is handled automatically:

```
from sixgill.utilities import get_active_session
model_filename = 'C:/temp/mymodel.pips'

def readparameters():
    model = get_model_session(model_filename)
    print(model.get_values("Well"))

def writeparameters():
    model = get_model_session(model_filename)
    new_param = range_to_dictionary(xw.Range('B4').expand().value)
    model.set_values(new_param)

def savemodel():
    model = get_model_session(model_filename)
    model.save()
```

Further examples for `get_model_session()` can be found in the PIPESIM Python Toolkit Case Studies.

7.5.4 Convert an Excel range to a Pandas DataFrame

Use this for passing PIPESIM tabulated data such as trajectories, geometries, and such.

To convert the list of lists read in from an Excel range into a Pandas DataFrame, follow this example:

```
>>> from sixgill.utilities import range_to_dataframe
>>> geometry = xw.Range('B4').expand().value
>>> df = range_to_dataframe(geometry)
```

7.5.5 Convert an Excel range to a dictionary

Use this for handling tabulated data such as boundary conditions.

To convert the list of lists read in from an Excel range into a Python dictionary, follow this example:

```
>>> from sixgill.utilities import range_to_dictionary
>>> bc = xw.Range('B4').expand().value
>>> df = range_to_dictionary(bc)
```


7.5.6 Get the address of the last cell of the worksheet

The `worksheet_last_row()` returns the last row with data in it. Likewise, `worksheet_last_column()` returns the last column with data.

```
>>> from sixgill.utilities import worksheet_last_row, worksheet_last_column
>>> last_row = worksheet_last_row()
>>> last_column = worksheet_last_column()
```

They optionally take a specified worksheet name:

```
>>> last_row = worksheet_last_row("ModelData")
>>> last_column = worksheet_last_column("ModelData")
```

EXAMPLE CODE

The PIPESIM Python Toolkit includes example scripts that cover all of its available methods, classes and properties. The scripts demonstrate the features and usage, which is indicated in the name of the script. They are located in the PIPESIM installation folder, under Development Tools/Python Toolkit.

Along with the example scripts there are also several Case Studies. These are designed to demonstrate potential use cases for the PIPESIM Python Toolkit with more in depth code and scripting. The case studies are located in the PIPESIM installation folder, under Case Studies/Python Toolkit.

Before running the scripts or case studies, it is necessary to copy the Case Studies and Development Tools folders from the Program Files location into your Documents home folder. This is because the typical security settings for Windows limit read and write access in Program Files.

8.1 Example Scripts

The following example scripts are included with the PIPESIM Python Toolkit.

Script	Description
add_and_delete.py	Add and delete model components.
add_well.py	Create a new model and add a well.
check_model_components.py	Open the Oil Network case study and review the model components.
connect_disconnect.py	Connect and disconnect model component connections.
connections.py	Display model connections.
convert.py	Convert a junction into a well or source.
copy_well.py	Copy (duplicate) a well.
create_black_oil_fluid.py	Create a black oil fluid.
create_compositional_fluid.py	Create a compositional fluid.
create_GIS_network.py	Create a network model with GIS coordinates.
create_network.py	Create a network model.
create_well_from_template.py	Create a well from a template.
describe.py	Display engineering units for parameters.
elevation.py	Get the elevations of GIS locations.
elevations.py	Get the elevations of a list of GIS locations.
excel_macros_functions.xlsx	Excel workbook with macros, functions and logging.
export_parameters.py	Export model component parameters.
find.py	Find model components.
generate_engine_files.py	Generate engine files for network and PT profile studies.
get_set_batch_update.py	Perform a batch update on component parameters.
get_set_black_oil.py	Update black oil fluid parameters.
Continued on next page	

Table 8.1 – continued from previous page

Script	Description
get_set_completion_test_points.py	Update completion test point parameters.
get_set_flow_correlations.py	Update global and local flow correlations.
get_set_geometry.py	Update flowline profile.
get_set_geothermal_profile.py	Update geothermal profile.
get_set_oil_well_ipr.py	Update inflow performance relationship on a well completion.
get_set_pq_curve.py	Update a source PQ relationship curve.
get_set_simulation_settings.py	Update the simulation settings.
get_set_study_conditions.py	Update study conditions.
get_set_value.py	Update a single parameter.
get_set_values.py	Update many parameters.
get_set_well_trajectory.py	Update a well trajectory.
networksim.py	Perform a network simulation.
networksim_async.py	Perform a network simulation as a background task.
new.py	Create a new PIPESIM model.
open_ui.py	Open the PIPESIM model in the UI.
open_close.py	Open and close a PIPESIM model.
open_close_eu.py	Open and close a PIPESIM model with alternative units.
open_save.py	Open and save a PIPESIM model.
ptprofilesim_calc_custom_variable.py	Perform a PT Profile simulation with custom variable.
ptprofilesim_calc_outlet_pressure.py	Perform a PT Profile simulation with a specified outlet pressure.
ptprofilesim_sensitivity.py	Perform a PT Profile simulation sensitivity study.
read_catalog.py	Update an ESP from the catalog.
run_all.py	Run all the examples.
well_export.py	Export a well to a new PIPESIM model.
well_export_all.py	Export all the wells to individual PIPESIM models.
well_import.py	Import a well into the PIPESIM model.

8.2 Case Studies

The following Case Studies are included with the PIPESIM Python Toolkit.

Case Study	Complexity	Description
Build Network Model	Complex	Builds a network model from data in an Excel workbook.
Daily Production	Simple	Runs simulations for each day of production data.
Export Flowline Geometries	Simple	Exports the flowline geometries from a model.
GIS Model Conversion	Complex	Converts junctions into model components after a GIS shape file import.
Network Simulation	Moderate	Uses Excel as a simple interface for network simulations.
PT Profile Simulation	Moderate	Uses Excel as a simple interface for PT Profile simulations.
Slug Catcher Sizing	Moderate	Performs multiple simulations for a slug catcher sizing study.
Well Model Management	Simple	Imports and exports individual well models from a network model.

8.2.1 Build Network Model

The data for a PIPESIM model can come from many different sources. Typically this is available in a database, or in a document such as Excel workbook.

In this case study the model data has been provided in a formatted Excel workbook. The script reads the workbook and automatically builds the PIPESIM model from the data. The user can update the parameterization, add or remove model components, and change the specified parameters in the workbook without needing to edit the script. The final model is opened in the PIPESIM UI for verification and review.

8.2.2 Daily Production

PIPESIM is sometimes used for production allocation and product metering verification. In these situations some of the production data is used as the boundaries in the PIPESIM model, with the remaining data compared to the PIPESIM simulation results.

The case study takes field data and runs the PIPESIM model using these as boundary conditions. The PIPESIM results are provided to compare with the field flow rates. This will highlight discrepancies in meter allocation, model calibration, or differences due to flow assurance issues.

8.2.3 Export Flowline Geometries

The GIS feature in PIPESIM provides elevation data capture from online data sources. After capturing the elevation data the profile is often wanted in other documents such as reports and calculations.

The script exports the geometry for all the flowlines in the model into individual csv (comma separated value) files as well as printing it to the interactive window. The csv files then can be easily opened in Excel, for example, or imported into corporate databases.

8.2.4 GIS Model Conversion

PIPESIM can import GIS shape files to automatically lay out the flowlines and junctions. After import, the junctions can be converted to the equipment that exists in the field, such as well, sources and sinks.

This case study takes a PIPESIM model file created from a shape file import and uses Excel data to do a search and replace on the model components. The components are matched against the latitude and longitude of the junctions.

The script is run from the desktop or through an interactive Python window. While it reads the Excel workbook, it demonstrates that scripts do not need to be embedded in order to interact with Excel.

8.2.5 Network Simulation

Excel can be used as simplified user interface for running PIPESIM network simulations. Pertinent data can be exposed and updated without danger of the user damaging the underlying PIPESIM model. Boundary conditions and equipment data can be updated, simulations performed, and the results reviewed.

This case study uses Excel as an interface for performing network simulations. The boundary conditions and specific equipment data is exposed for user updates. The simulation can be run and results retrieved for analysis.

8.2.6 PT Profile Simulation

Excel can be used as simplified user interface for running PIPESIM PT profile simulations. Pertinent data can be exposed and updated without danger of the user damaging the underlying PIPESIM model. Boundary conditions and equipment data can be updated, simulations performed, and the results reviewed.

This case study uses Excel as an interface for performing PT Profile simulations. The boundary conditions is exposed for user updates, the simulation run and results retrieved for analysis.

8.2.7 Slug Catcher Sizing Study

PIPESIM is often used on projects during the concept and front end engineering design (FEED) phases for equipment sizing.

In this use case, a parametric study is performed on a PIPESIM model to determine the size of a slug catcher at an onshore receiving facility. It varies the flow correlation, Equation of State and liquid / gas ratio for pigging and ramp up scenarios. It is based on the paper presented at Pipeline Simulation Interest Group, PSIG1603.

8.2.8 Well Model Management

A use case for model management is to model wells individually and to import them into a single PIPESIM file for a network simulation. It allows you to maintain individual well models, while being able to easily run a single large network simulation.

This case study provides an Excel interface for managing the well models. It will export all the wells in a PIPESIM model to individual files and import them again on the click of a button. The well names in the model are also listed.

TROUBLESHOOTING

9.1 Python Toolkit fails to install

9.1.1 Symptoms

The installer gives an error of “Installation Failed”

9.1.2 Resolutions

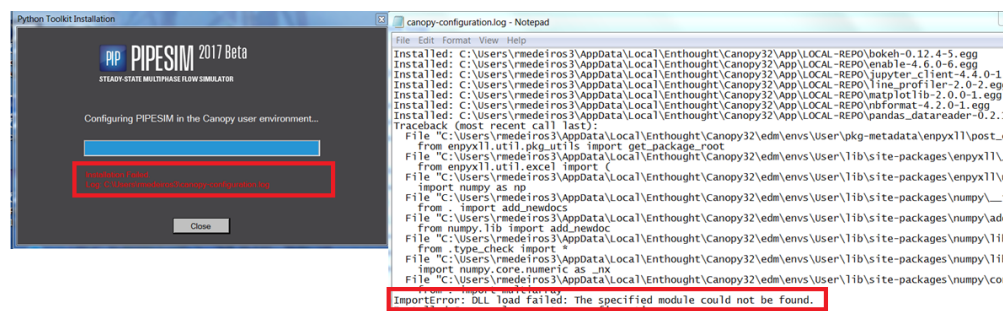
Default “My Documents” folder is not on C: drive

Canopy fails to install properly when the users “Documents” folder has been moved or changed from the default “C:\users*username*” location. This is a known issue that will be addressed in a later release. To install Canopy and the PIPESIM Python Toolkit in this case, follow these steps:

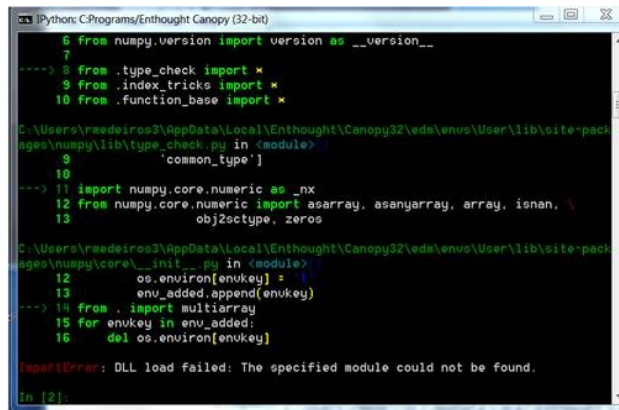
1. Install PIPESIM as normal.
2. Reveal the Windows hidden system files
3. Rename the alias from My Documents by appending “.current”
4. Create a new, temporary folder with the same name
5. Proceed with the installation of the PIPESIM Python Toolkit
6. Move all folders within the new My Documents folder to your current Documents folder
7. Delete the temporary folder.
8. Rename the alias back by deleting the suffix “.current”.

Log file error “ImportError: DLL load failed”

In rare cases the DLLs required for PyXLL are not registered on the computer. The last line of the error log will give the message “ImportError: DLL load failed: The specified module could not be found.”



1. Verify the issue by performing these steps:
 - (a) Open Canopy command prompt
 - (b) Run 'ipython'
 - (c) Run 'import numpy'
 - (d) You should see the error: 'ImportError: DLL load failed: The specified module could not be found.'
 - (e) Close Canopy command prompt window
2. Resolve the issue by following these steps:
 - (a) Copy the folder 'C:\Users\[LOCALUSER]\AppData\Local\Enthought\Canopy32\edm\envs\User\PrivateDLLs' to 'D:'
 - (b) Add to Environment variable 'PATH' the path 'D:\PrivateDLLs'
 - (c) Open a new Canopy command prompt window
 - (d) Run step 1 again and make sure you don't have the error anymore.
 - (e) If the problem was fixed, uninstall and install PIPESIM Python toolkit.



```

Python: C:\Programs\Enthought Canopy (32-bit)
6 from numpy.version import version as __version__
7
----> 8 from .type_check import *
9 from .index_tricks import *
10 from .function_base import *

C:\Users\rmedeiros3\AppData\Local\Enthought\Canopy32\edm\envs\User\lib\site-pack
ages\numpy\lib\type_check.py in <module>
9     'common_type']
10
--> 11 import numpy.core.numeric as _nx
12 from numpy.core.numeric import asarray, asanyarray, array, isnan,
13     obj2ctype, zeros

C:\Users\rmedeiros3\AppData\Local\Enthought\Canopy32\edm\envs\User\lib\site-pack
ages\numpy\core\_init_.py in <module>
12     os.environ[enukey] = '1'
13     env_added.append(enukey)
--> 14 from . import multiarray
15 for enukey in env_added:
16     del os.environ[enukey]

ImportError: DLL load failed: The specified module could not be found.

In [2]:

```

9.2 PIPESIM Tab does not appear in Excel

9.2.1 Symptoms

There is no PIPESIM tab in the ribbon after opening Excel.

9.2.2 Resolutions

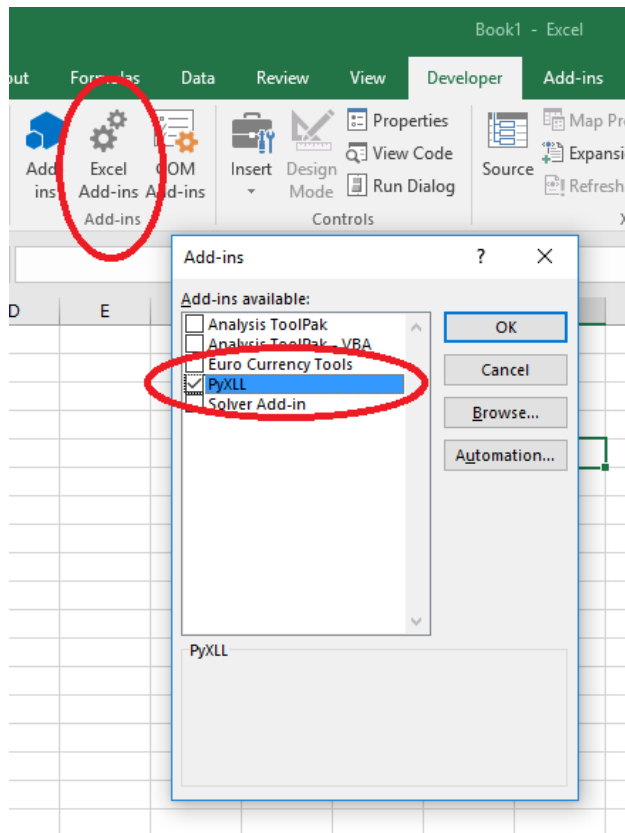
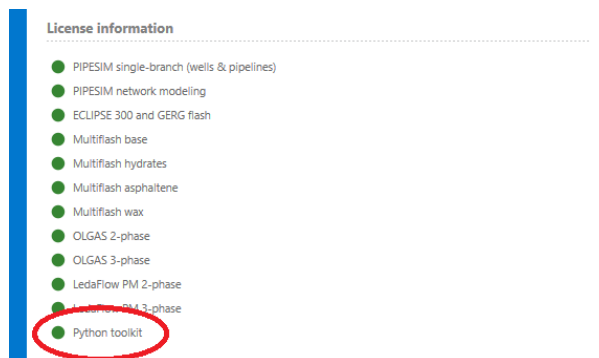
Verify PIPESIM Python Toolkit license

The PIPESIM tab in Excel will appear when there PIPESIM Python Toolkit license available. Open the PIPESIM user interface and on the main page check the Python Toolkit license status. It should be green.

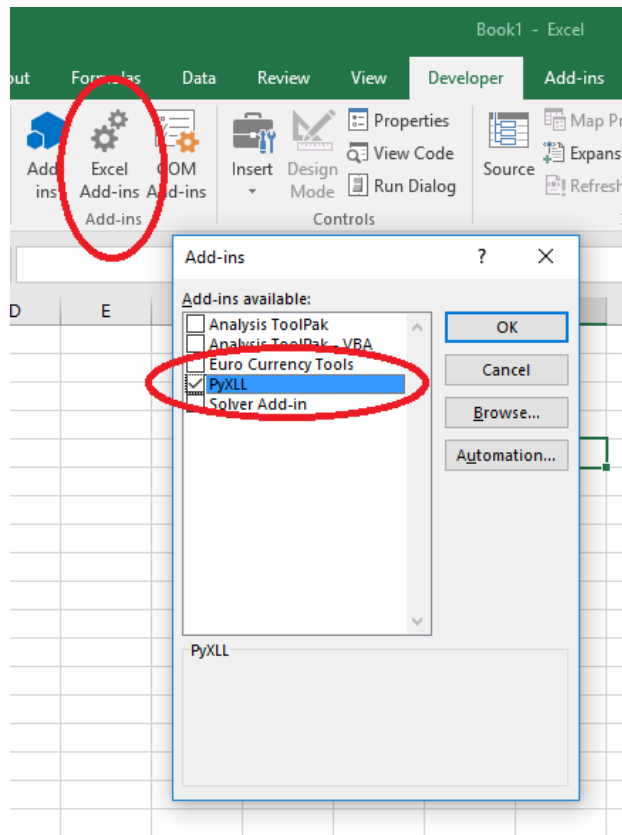
Verify PyXLL has been installed properly

In a rare cases the third party Python module PyXLL fails to register with Excel. Open Excel, select the Developer tab, and then click "Excel Add-ins". The Add-ins pop-up should show the PyXLL addin available and selected.

If PyXLL is not listed then follow the these steps to enable it:

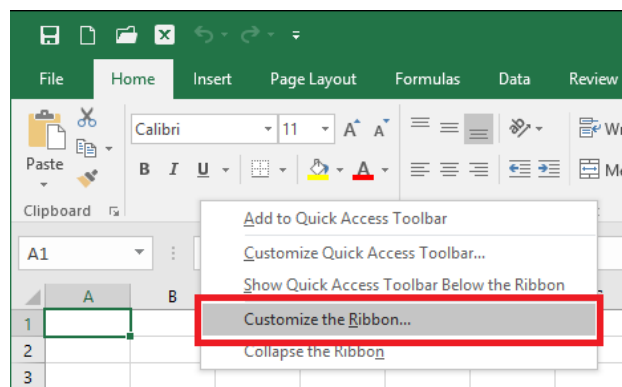


1. On the Add-ins pop-up, click Browse...
2. Browse to the folder “%LOCALAPPDATA% EnthoughtCanopy32edmenvsUserLibsite-packagespyxll”
3. Select the “pyxll.xll” file and click OK.
4. PyXLL will be included in the list of Add-ins and the PIPESIM tab will show up.

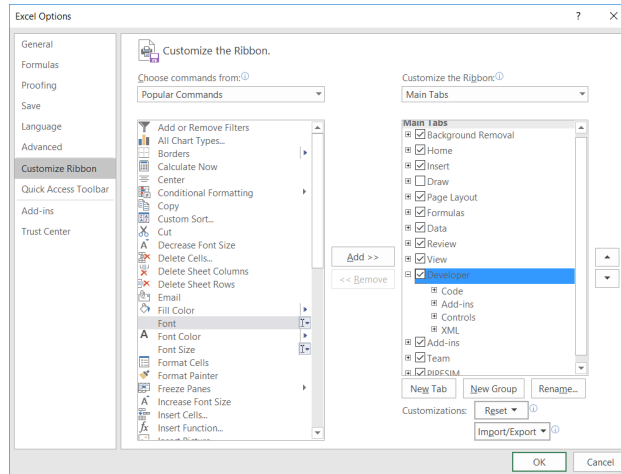


If the Developer tab is not showing on the Excel ribbon, then:

1. Right click on the ribbon and select “Customize Ribbon”.



2. Select “Developer Tab” and click OK.



9.3 Integrated Excel script fails to run

9.3.1 Symptoms

- The assignment of a method using `@entry_point` appears in the Scripts dropdown list but clicking on it gives a cryptic error message.
- Assigning a macro to a button prefixes the method name with the Excel workbook name, e.g. `'MyExcelWorkbook'!mymacro`, and fails to run.
- A function defined using `@xl_func` fails to run and gives a `#NAME?` error in the cell.

9.3.2 Resolutions

Script or Coding Errors

When it is saved PyXLL automatically reloads the script as a module. At this point the Python compiler scans the code in the same way that is done with a Python import statement. Any compilation errors in the code will stop the script from being successfully imported. Typical coding errors that may cause this include:

- Missing import declarations. Verify that all the libraries used in the script are being imported, particularly the decorators `@xl_macro`, `@xl_func`, `@entry_point` and `@log_error`.
- Coding indentation and spelling errors for code sections that are not part of a method or property.

Edit the script in Canopy and click on the “Run” button on the toolbar. This will do the same action and provide error messages as to the problem in the code.

KNOWN ISSUES

10.1 Boundary Simplification Sources

10.1.1 Description

The “Use Surface Conditions” feature specifies the boundary at the wellhead rather than the downhole completion, allowing you to set the well outlet flow or pressure. The methodology in the PIPESIM data model is not yet included by the Python Toolkit and, as a result, this feature is not fully supported at this time.

Changing the “Use Surface Conditions” using the `set_use_surface_conditions()` method will make the corresponding change in the model. The catalog of model components is not updated, however, hence the list of boundary conditions is out of sync with that required. In addition, the catalog does not correctly identify all the boundary conditions when “Use Surface Conditions” is toggled on.

Note that the “Treat As Source” parameter on a junction is similar but the catalog is properly updated.

10.1.2 Workarounds

Ensure that the “Use Surface Conditions” boundary specification are set in the model through the UI before the model is used in the Python Toolkit.

Be aware that when “Use Surface Conditions” is turn on, the well boundary is a source within the well, with a context “*wellname*:Source 1”. The properties for the source can be accessed through the `get/set_values()` method by passing the string context (e.g. “Well:Source 1”) or the component context (e.g. Well=“Well”)

Be aware that for a junction with the “Treat As Source” parameter set to True, the boundary is a source with the context “*junctionname*: **junctionname*_SimpleSource”. The properties for the source can be access through the `get/set_values()` methods by passing the string context (e.g. “J1:J1_SimpleSource”) or the component context (e.g. Junction=“J1”).

This issue will be resolved in the next release of the Python Toolkit.

10.2 Canopy Package Manager does not work

10.2.1 Description

The Python Package Manager is provided as a UI for selecting and updating the Python packages in the Canopy installation. In addition, many packages have dependencies on other packages and the Package Manager verifies that the required package dependencies and versions are compatible.

For the first release of Python Toolkit the Package Manager is disabled, such that it cannot download updates or install new packages. This ensures that the Python Toolkit is not broken by the installation of new packages or updates to existing ones.

10.2.2 Workarounds

You can still use the command line Python pip command for updating or installing new packages. To use pip, open a new DOS Command Window and enter:

```
> pip install packagename
```

This issue will be resolved in the next release of the Python Toolkit.

REFERENCE DOCUMENTATION

11.1 sixgill package

11.1.1 sixgill.pipesim module

The pipesim module maintains the high level classes and methods for the PIPESIM Python Toolkit. This includes aspects such as the Model() class for creating a session with a PIPESIM model as well as the static classes that enumerate the options that are available.

class `sixgill.pipesim.Model(pipesim_session, model_file, unit_system)`

Bases: `object`

The Model() class defines a session for working with a PIPESIM model. It maintains the connection details of the session while the Manta module performs the actual interaction with the PIPESIM model. Users should open a PIPESIM model using the Model.open() which takes care of handling the low level Manta calls.

Parameters

- **pipesim_session** (`manta.server.manager.PipesimManager`) – The pipesim_session object from the Manta module.
- **model_file** (`str`) – The full folder and filename of the pipesim file.
- **unit_system** (`str`) – The engineering units system being used for reading and writing the values, as defined by the Units static class.

Returns The PIPESIM model session object.

Return type `Model`

Examples

```
>>> mymodel=Model.open("C:/ProjectX/Models/PIPESIM/ProjectXModel.pips")
```

is_open

Whether the pipesim model is open (True) or not (False)

api

The low level Manta library API

filename

The model file name and path

static open(*filename: str, units: typing.Union[str, NoneType] = 'PIPESIM_FIELD'*)

Open a PIPESIM model

Opens the PIPESIM model.

Parameters

- **filename** (*str*) – The PIPESIM model path and file name.
- **units** (*str*) – The engineering units that will be used by POET. This is one of the Units enumerations, with the default being FIELD units.

Returns The model object that has been open.

Return type *Model*

Examples

```
>>> model = Model.open("C:/POET Demo/CSN_301_small_network.pips")
>>> model = Model.open("C:/MyProject/FlowNetwork.pips", Units.METRIC)
```

```
static new(filename: str, units: typing.Union[str, NoneType] = 'PIPESIM_FIELD', overwrite: typing.Union[bool, NoneType] = False)
```

Open a new PIPESIM model

Opens a new PIPESIM model.

Parameters

- **filename** (*str*) – The new PIPESIM model path and file name.
- **units** (*str*) – The engineering units that will be used by POET. This is one of the Units enumerations, with the default being FIELD units.
- **overwrite** (*bool*) – If the file already exists, this flag will define if it should be overwritten or not.

Returns The model object that has been open.

Return type *Model*

Examples

```
>>> model = Model.new("C:/POET Demo/new_workspace.pips")
>>> model = Model.new("C:/MyProject/NewWorkspace.pips", Units.METRIC)
```

```
open_ui(filename: typing.Union[str, NoneType] = '')
```

Open a PIPESIM file with PIPESIM UI

Opens the PIPESIM model file in the PIPESIM UI. It supports opening a model directly, and opening the model that is open in the Toolkit. A model that is open in the PIPESIM Python Toolkit will be saved and closed before opening in the UI. This is because changes cannot be made to a model simultaneously in the UI and through the Python Toolkit.

Parameters **filename** (*str*) – The PIPESIM file path and name. This is not required when opening a model that is already open in the Python Toolkit.

Examples

```
>>> # Without first opening the model in Python
>>> Model.open_ui("C:/POET Demo/CSN_301_small network.pips")
>>> # While the model is open in the Python Toolkit
>>> model = Model.open("C:/POET Demo/CSN_301_small network.pips")
>>> model.open_ui()
```

save(filename='')

Save the PIPESIM model

Saves the PIPESIM model. If the filename is not specified then it performs a save; if the filename is specified then it performs a Save As. the suffix “pips” may be omitted.

Parameters **filename** (*str*) – Optional. The full path and name of the file to save.

Examples

```
>>> model.save()
>>> model.save("C:/Temp/MyModel2.pips")
```

close()

Close the PIPESIM model

Closes the PIPESIM model.

Examples

```
>>> model.close()
```

context

The root context of this model.

Returns Returns the model context object that matches the entirety of the model.

Return type AbstractModelContext

find(context: typing.Union[str, NoneType] = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str) → typing.List[]
Find the model components in the context.

Find the specified components in the model.

Parameters

- **context** (*str*) – Optional. The context (identifier) of the item to look for.
- **component** (*str*) – Optional. The model component to look for.
- **model_context** (*AbstractModelContext*) – Optional. The model context object to look for.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns Returns a list of the component contexts (identifiers) of the model components that match the specified criteria.

Return type list

Examples

```
>>> model.find(component=ModelComponents.CHOKE, Name="CK-1")
>>> model.find(Well="Well103")
```

```
find_components(context: typing.Union[str, NoneType] = None, component:
                  typing.Union[str, NoneType] = None, model_context: typ-
                  ing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] =
                  None, **context_keywords: str) → typing.List[]
```

Find the full context of the specified component

Returns the context objects of the items found. This is functionally similar to the `find()` but returns the list of `AbstractModelContext` objects rather than the context strings.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **component** (*str*) – Optional. The model component to look for.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of `component='value'`, where `component` is the model component of the item, and `value` is the value of the model component. Values can also be `ANY` (matches any value) or `NONE` (matches no value).

Returns Returns a list of the contexts for each of the model components that match the specified criteria.

Return type `list[AbstractModelContext]`

Examples

```
>>> model.find_components(component=ModelComponents.CHOKE, context="CK-1")
>>> model.find_components(component=ModelComponents.COMPLETION, Well="Well103")
```

```
filter(context: typing.Union[str, NoneType] = None, component: typing.Union[str, NoneType] =
        None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, None-
        Type] = None, **context_keywords: str)
```

Filter the model, creating a submodel of the items of interest

Returns a Model class containing just the items specified in the context. It effectively allows filtering of the model on specific aspects and then using the usual methods (`get/set/etc.`) on the filtered model.

Parameters

- **context** (*str*) – Optional. The context (identifier) of the item to look for.
- **component** (*str*) – Optional. The model component to look for.
- **model_context** (*AbstractModelContext*) – Optional. The model context object to look for.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of `component='value'`, where `component` is the model component of the item, and `value` is the value of the model component. Values can also be `ANY` (matches any value) or `NONE` (matches no value).

Returns Returns a list of the component contexts (identifiers) of the model components that match the specified criteria.

Return type list

Examples

```
>>> # Filter for all the flowlines
>>> filtered_model = model.filter(Flowline=ALL)
>>> # Filter for all the items in the specified well
>>> filtered_model = model.filter(Well="Well03")
```

```
describe(context: typing.Union[str, NoneType] = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, parameter: typing.Union[str, NoneType] = None, **context_keywords: str)
```

Describe a parameter given a context.

Returns an object with properties for the parameter description such as the units of measurement and unit conversions. The available properties are:

- **units**: The units of measurement, e.g. inches
- **units_symbol**: The symbol(s) for the unit of measurement, e.g. ins
- **si_offset**: The unit conversion offset (bias) from SI
- **si_factor**: The unit conversion factor (gain) from SI

Parameters

- **context** (*str*) – Optional. The context (identifier) of the item to look for.
- **parameter** (*str*) – Required. The parameter that needs to be described.
- **component** (*str*) – Optional. The model component to look for.
- **model_context** (*AbstractModelContext*) – Optional. The model context object to look for.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns The object with the descriptions.

Return type Description

Examples

```
>>> eu = model.describe(context="Well",
                        parameter=Parameters.Well.TESTTRACKTEMPERATURE).units
```

```
get_value(context: typing.Union[str, NoneType] = None, parameter: typing.Union[str, NoneType] = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str) → typing.Any
```

Return a single parameter value from a context in the model.

The function queries the model for the specified item and parameter returning the parameter value. An error is raised if the context matches more than one item.

Parameters

- **context** (*str*) – Optional. The context (identifier) to search for.
- **parameter** (*str*) – Optional. The parameter on the item to return the value of.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – The model context object to search for.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns The value of the specified parameter.

Return type var

Examples

```
>>> val = model.get_value(context="Choke23",
                           parameter=Parameters.Choke.BEANSIZE)
>>> val = model.get_value(Well="Well 23", Choke="CK1",
                           parameter=Parameters.Choke.BEANSIZE)
```

```
set_value(context: typing.Union[str, NoneType] = None, parameter: typing.Union[str, NoneType] = None, value: typing.Any = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str)
```

Set a parameter value on a single context item in the model.

The function queries the model for the specified item and then sets the parameter to the specified value. An error is raised if the context matches more than one item.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **parameter** (*str*) – Required. The parameter on the item to be set.
- **value** (*var*) – Required. The value to set the parameter to.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_value(context="Choke23",
                    parameter=Parameters.Choke.BEANSIZE, value=4.5)
>>> model.set_value(Well="Well 23", Choke="CK1",
                    parameter=Parameters.Choke.BEANSIZE, value=5)
```

```
set_all_value(context: typing.Union[str, NoneType] = None, parameter: typing.Union[str, NoneType] = None, value: typing.Any = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str)
```

Set a parameter value for all items in the context.

The function queries the model for the specified items and then sets the parameter to the specified value. It sets the parameter for all items within the context.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **parameter** (*str*) – Required. The parameter on the item(s) to set.
- **value** (*var*) – Required. The value to set the parameter to.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_all_value(Source=ALL,
                        parameter=Parameters.Source.ASSOCIATEDBLACKOILFLUID,
                        value = 'BK111')
```

```
get_values(context: typing.Union[str, NoneType] = None, parameters: typing.List[] = [], component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, show_units: bool = False, **context_keywords: str) → typing.Dict[KT, VT]
```

Returns a dictionary with the parameters and values from a context.

The function queries the model for the specified items and parameters returning a dictionary. The key of the dictionary is context string, and the value of the dictionary is a sub dictionary, which contains parameter as the key and its value

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **parameters** (*list*) – Optional. The parameter on the item to return the value of. If empty or not provided then it returns all the parameters.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns The dictionary containing the parameter values.

Return type Dict

Examples

```
>>> dict = model.get_values(context="Choke23",
                             parameters=[Parameters.Choke.BEANSIZE,
                                           Parameters.Choke.CRITICALPRESSURERATIO])
>>> dict = model.get_values(Well="Well 23", Choke="CK1",
                             parameters=[Parameters.Choke.BEANSIZE,
                                           Parameters.Choke.CRITICALPRESSURERATIO])
```

```
set_values(dict: typing.Dict[KT, VT], context: typing.Union[str, NoneType] = None,
            component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
            **context_keywords: str)
```

Set parameters values as specified by the Python dictionary.

The key of the dictionary is context string, and the value of the dictionary is a sub dictionary, which specify the parameter as key and its value. The dictionary is therefore the same format as the `get_values()` function.

Parameters dict (*Dict*) – Required. The Python dictionary to set the item parameters.

Examples

```
>>> model.set_values(dict=values)
```

```
get_trajectory(context: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
                **context_keywords: str) → pandas.core.frame.DataFrame
```

Get the specified well trajectory.

Get the well trajectory as identified by the specified context. If more than one item is found by the context then it will fail.

Parameters

- **context** (*str*) – Optional. The context string to search for well.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of `component='value'`, where `component` is the model component of the item, and `value` is the value of the model component. Values can also be `ANY` (matches any value) or `NONE` (matches no value).

Returns The trajectory data frame includes following columns: MeasuredDepth Inclination, Azimuth and MaxDogLegSeverity

Return type `pandas.DataFrame`

Examples

```
>>> df = model.get_trajectory(context="Well-1")
>>> dfw = model.get_trajectory(Well="Well 23")
```

```
set_trajectory(context: typing.Union[str, NoneType] = None, value: pandas.core.frame.DataFrame = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
                **context_keywords: str)
```

Set well trajectory to the values specified in the DataFrame.

Sets well trajectory to the values specified in the DataFrame. It will fail if more than one item is found by the context.

Parameters

- **value** (*DataFrame*) – Required. The data frame to set to the well trajectory.
- **context** (*str*) – Optional. The context string to look for the well.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_trajectory(context="Well-1", value = df)
>>> model.set_trajectory(Well="Well 23", value = df)
```

```
get_completion_test_points(context: typing.Union[str, NoneType] = None, model_context:
                           typing.Union[sixgill.core.model_context.AbstractModelContext,
                           NoneType] = None, **context_keywords)
```

Return well completion test points found in the context.

Return well completion test points found in the context. It will fail if more than one item is found by the context.

Parameters

- **context** (*str*) – Optional. The context string of completion
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value). e.g. Well = 'well_name', Completion = 'completion_name'

Returns The completion test points data frame includes following columns: LiquidFlowrate, GasFlowrate, StaticReservoirPressure, BottomHoleFlowingPressure

Return type pandas.DataFrame

Examples

```
>>> df = model.get_completion_test_points(context="Well-1:VertComp")
```

```
set_completion_test_points(context: typing.Union[str, NoneType] = None, value: pan-
                           das.core.frame.DataFrame = None, model_context: typ-
                           ing.Union[sixgill.core.model_context.AbstractModelContext,
                           NoneType] = None, **context_keywords)
```

Set data frame to the completion test points of well in the context.

Set data frame to the completion test points of well in the context. It will fail if more than one item is found by the context.

Parameters

- **value** (*DataFrame*) – Required. The data frame to set to the well completion test points
- **context** (*str*) – Optional. The context string to look for the completion
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_completion_test_points(context="Well-1:VertComp", value = df)
```

```
get_pq_curve(context: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
              **context_keywords) → pandas.core.frame.DataFrame
```

Return PQ curve in the context.

Returns PQ curve in the context. It will fail if more than one item is found in the context.

Parameters

- **context** (*str*) – Optional. The context string to look for source.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns The PQ curve data frame includes the following columns: GasFlowRate, LiquidFlowRate, MassFlowRate, Pressure

Return type pandas.DataFrame

Examples

```
>>> df = model.get_pq_curve(context="Source_1")
>>> dfw = model.get_pq_curve(Source="Source 23")
```

```
set_pq_curve(context: typing.Union[str, NoneType] = None, value: pandas.core.frame.DataFrame = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
              **context_keywords)
```

Set data frame to the PQ curve of source in the context.

Set data frame to the PQ curve of source in the context. It will fail if more than one item is found in the context or the source does not use PQ curve.

Parameters

- **value** (*DataFrame*) – Required. The data frame to set to the source PQ curve
- **context** (*str*) – Optional. The context string to look for the source.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.

- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_pq_curve(context="Source-1", value = df)
>>> model.set_pq_curve(Source="Source 23" value = df)
```

get_geometry (*context*: *typing.Union[str, NoneType]* = *None*, *model_context*: *typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType]* = *None*, ***context_keywords: str*) → *pandas.core.frame.DataFrame*

Return flowline geometry profile in the context.

Returns flowline geometry profile in the context. It will fail if more than one item is found in the context.

Parameters

- **context** (*str*) – Optional. The context string to search for flowline.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns If flowline is in detailed mode, return flowline detailed profile data.

Return type *pandas.DataFrame*

Examples

```
>>> df = model.get_geometry(context="fl-1")
>>> dfw = model.get_geometry(Flowline="fl-2")
```

set_geometry (*context*: *typing.Union[str, NoneType]* = *None*, *value*: *pandas.core.frame.DataFrame* = *None*, *model_context*: *typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType]* = *None*, ***context_keywords*)

Set data frame to the flowline geometry profile in the context.

Set data frame to the flowline geometry profile in the context. It will fail if more than one item is found in the context.

Parameters

- **value** (*DataFrame*) – Required. The data frame to set to the flowline geometry profile
- **context** (*str*) – Optional. The context string to search for flowline.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_geometry(context="fl-1", value = df)
>>> model.set_geometry(Flowline="fl-2", value = df)
```

```
get_geothermal_profile(context: typing.Union[str, NoneType] = None, model_context:
                        typing.Union[sixgill.core.model_context.AbstractModelContext,
                        NoneType] = None, **context_keywords: str) → pan-
                        das.core.frame.DataFrame
```

Return the flowline or well geothermal profile in the context.

Returnz the flowline or well geothermal profile in the context. It will fail if more than one item is found in the context.

Parameters

- **context** (*str*) – Optional. The context string to search for flowline or well.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns Return geothermal data frame. If flowline is simple, or geothermal data use global settings, empty data frame is returned.

Return type pandas.DataFrame

Examples

```
>>> df = model.get_geothermal_profile(context="fl-1")
>>> dfw = model.get_geothermal_profile(Well="well-1")
```

```
set_geothermal_profile(context: typing.Union[str, NoneType] = None, value: pan-
                        das.core.frame.DataFrame = None, model_context: typ-
                        ing.Union[sixgill.core.model_context.AbstractModelContext, None-
                        Type] = None, **context_keywords: str)
```

Set the DataFrame values to the flowline geothermal profile in the context.

Sets the DataFrame values to the flowline geothermal profile in the context. It will fail if more than one item is found in the context.

Parameters

- **value** (*DataFrame*) – Required. The data frame to set to the flowline geothermal profile
- **context** (*str*) – Optional. The context string to search for flowline or well.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.set_geothermal_profile(context="fl-1", value = df)
>>> model.set_geothermal_profile(Flowline="fl-2", value = df)
```

read_catalog(*context=None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str*)
Update the equipment data from the catalog.

Update equipment data for an ESP, PCP, Pump or Compressor from the catalog. The equipment must have Manufacturer and Model set so that the entry in the catalog can be identified. It will fail if more than one item is found in the context.

Parameters

- **context** (*str*) – Optional. The context string to look for pump
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.read_catalog(context="Pump-1")
```

batch_update()

Apply batch updates to the model

Ordinarily changes to the model through set...() are immediately applied. The batch update allows a series of updates to be cached and then updated as group. This is useful where many changes need to be applied, which can be bundled together and applied all at once. This has a performance improvement over individual updates.

Returns

Return type None

Examples

```
>>> with model.batch_update():
    model.set_value("Choke", Parameters.Choke.BEANSIZE, 56)
    model.set_geometry("FL-1", geometry_df)
```

about

Return information about the current PIPESIM version.

Returns the versions of PIPESIM, web API and PIPESIM Python Toolkit that are being used. Called as a function about() it returns a dictionary of the version information. It will also return the individual properties of the ModelAbout() class.

Parameters None –

Returns Information about the PIPESIM version.

Return type dict

Examples

```
>>> version_dict = model.about()
>>> webapi_version = model.about.webapi_version
```

get_elevation(*lat: float, long: float*) → float

Get the elevation of a location

Returns the elevation of the latitude / longitude location.

Parameters

- **lat** (*float*) – Required. The latitude of the location in degree.
- **long** (*float*) – Required. The longitude of the location in degree.

Returns The elevation of the location with specified lat and long in meter Raise ValueError exception if the location is invalid.

Return type float

Examples

```
>>> elev = model.elevation(0.2, 0.5)
```

get_elevations(*locations*)

Get the elevation of a array of location points with latitude longitude

Returns an array of location points with latitude, longitude and elevation.

Parameters **location** (*array[dict{Longitude:value, Latitude:value}, ...]*) – Required. Location array with longitude and latitude points.

Returns **array[dict{Longitude** – The location array with latitude, longitude and elevation values for each point. Raise ValueError exception if the location is invalid.

Return type value, Latitude: value, Elevation: value, ...]

Examples

```
>>> location = model.elevation([{'Latitude':0.2, Longitude:0.5}])
```

export_parameters(*filename: str, with_templates: bool = False*) → str

Export the model parameters to a CSV file.

Exports the model parameters to a comma separated values (CSV) file. Note that this only includes the scalar model parameters and simulation settings; tabulated data such as trajectories and profiles are not included.

Parameters

- **filename** (*str*) – Required. The name of the CSV filename to save the CSV.
- **with_templates** (*bool*) – Optional. Whether to include templates (True) or not (False). The default is False.

Returns The full absolute path to the CSV file saved.

Return type str

Examples

```
>>> file = model.export_parameters(filename="C:/poet/model_details.pips")
```

sim_settings

The simulation settings

fluids

The fluid settings, primarily setting up compositional fluids

tasks

The simulation tasks

export_well (*context: typing.Union[str, NoneType] = None, folder: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords: str*) → typing.List[]

Export the specified well(s) into PIPESIM files

Exports the specified well(s) in to individual PIPESIM files in the specified folder. The filename will match the well name. If the well name is not specified then all the wells in the model are exported. If folder is not specified, then the wells are exported to the current working folder

Parameters

- **context** (*str*) – Optional. The context string to search for well.
- **folder** (*str*) – Optional. The folder location to save the files into.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns The list of filenames with full path of each well exported.

Return type list

Examples

```
>>> files = model.export_well()
>>> files = model.export_well(context="Well 03")
>>> files = model.export_well(Well="Well 1")
```

import_well (*filename, name, fluid_override=False, overwrite_existing=False*)

Import a well into the model.

Imports the well from the file into the model with the specified options.

Parameters

- **filename** (*str*) – Required. The filename to import the well from
- **name** (*str*) – Required. The name that will be given to the well in the model.
- **fluid_override** (*bool*) – Specifies whether the fluid from the well import file overwrites the existing fluid in the model (True) or not (False).
- **overwrite_existing** (*bool*) – Specifies whether the well will overwrite (replace) an existing well in the model with the same name.

Examples

```
>>> model.import_well(filename="C:/temp/mywells/Well01.pips", name="Well02",
                      fluid_override=False, over_write_existing=False)
```

delete(*context: typing.Union[str, NoneType] = None, component: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, parameter: typing.Union[str, NoneType] = None, **context_keywords: str*)
Delete components from the model.

Deletes the specified components from the model

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **component** (*str*) – Optional. The model component to look for.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.delete(context="Sep01")
```

add(*component, name, context: typing.Union[str, NoneType] = None, parameters: typing.Union[dict, NoneType] = {}*)
Adds components to the model.

Create a new items in the model of the specified component type.

Parameters

- **component** (*str*) – Required. The type of model component to add to the model. This is one of the ModelComponents enumerations.
- **name** (*str*) – Required. The name of the item added to the model.
- **context** (*str*) – Optional. The string representation of the context to add the component. For example, when adding downhole object to a well, it is the context string of the well.
- **parameters** (*dict*) – Optional. The parameters to set when creating the item. If not specified then the item is created with the default parameters.

Examples

```
>>> model.add(component=ModelComponents.BLACKOILFLUID, name="BO_01",
              parameters={Parameters.BlackOilFluid.GOR:200})
>>> model.add(component=ModelComponents.CHOKE, name="TSV-01",
              {Parameters.Choke.BEANSIZE:5})
>>> model.add(component=ModelComponents.CHOKE, name="ck", context="Well 1",
              parameters={Parameters.DowholeLocation.TOPMEASUREDDEPTH:500,
                          Parameters.Choke.BEANSIZE=4.5})
```

copy(*context: str, name: str, use_template: typing.Union[bool, NoneType] = False*)

Duplicate an item in the model.

Copy a model component. **Note that it only works for the following components in the current release:**

- Well

Parameters

- **context** (*str*) – The existing model component to be duplicated.
- **name** (*str*) – The name of the duplicate. If this already exists then it will append a suffix to the name.
- **use_template** (*Boolean*) – Optional. Duplicate the item from the the catalog, if it exists.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.copy(Well='Simple vertical', name='Sixgill', use_template=True)
>>> model.copy(Name='Well01', name='Well01', use_template=False)
```

connect(*source, destination, source_port=None*)

Connect the source item to the destination item

Creates a connection from the source item to the destination item. It will fail if the source context or destination context match more than one item.

Parameters

- **source** (*dict or context string*) – The find context for the source item.
- **destination** (*dict or context string*) – The find context for the destination item.
- **source_port** (*str*) – If the source is a TwoPhaseSeparator or ThreePhaseSeparator, this is required to indicate which outlet should be used to make the connection, as provided by the Connection.Separator enumeration.

Examples

```
>>> connect(source={ModelComponent.Well:"Well 3"},
            destination={"Choke":"TCV-03"})
>>> connect(source="Sep-01", destination="ExportGasSink",
            source_port=Connection.Separator.Top)
```

disconnect(*source, destination: typing.Union[str, NoneType] = None, port: typing.Union[str, NoneType] = None*)

Disconnect the source and the destination components.

Removes the connection between the source model component and the destination model component. Alternately, specifying the source and the source port will break the connection from the specified port. The method will fail if the source or destination contexts do not exist or there is no connection between them.

Parameters

- **source** (*dict or str*) – The source model component (starting point for the connection). This can either be a context string or a context dictionary.
- **destination** (*dict or str*) – Optional. The destination model component (ending point of the connection).
- **port** (*str*) – Optional. The port to be disconnected on the source model component. Note that some model components (e.g. Junction, Heat Exchange, Choke) do not have indentifiable inlet and outlet ports hence should need to use the destination argument instead.

Examples

```
>>> disconnect(source={ModelComponent.Well:"Well 3"},
               destination={Choke:"TCV-03"})
>>> disconnect(source="Well 3", destination="TCV-03")
```

```
connections(context: typing.Union[str, NoneType] = None, component:
              typing.Union[str, NoneType] = None, model_context: typ-
              ing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
              **context_keywords: str) → typing.List[]
```

Get the model component connections in the context.

Returns the model component connections in the context. It is provided as a list of dictionaries with the Source, Destination and Source Port for easy reviewng and formatting in a DataFrame. If no context is provided, then the information of all connections are returned.

Parameters

- **context** (*str*) – Optional. The context string to search for Source or Destination.
- **component** (*str*) – Optional. The model component to look for Source or Destination.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns Return a list of dictionaries. Each dictionary contains connection information including Source, Source Port, and Destination

Return type List[Dict]

Examples

```
>>> model.connections(Well=ALL)
```

```
get_connections(context: typing.Union[str, NoneType] = None, component:
                 typing.Union[str, NoneType] = None, model_context: typ-
                 ing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] =
                 None, **context_keywords: str) → typing.Dict[KT, VT]
```

Get the connections for each item in the context.

Returns the connections for each item in the context. It is formatted as a dictionary of context, with each context providing the source, destination and the ports of the context items. It is therefore useful as a

look-up table to find the inlets and outlets of specific model components. If no context is provided, then the information of all connections is returned.

Parameters

- **context** (*str*) – Optional. The context string to search for Source or Destination.
- **component** (*str*) – Optional. The model component to look for Source or Destination.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns Returns a dictionary of dictionaries. Each dictionary contains connection information including Source, Source Port, and Destination

Return type Dict[Dict]

Examples

```
>>> model.get_connections("MBD-101")
```

```
convert (context: typing.Union[str, NoneType] = None, to_component: str = None, model_context:
        typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **con-
        text_keywords: str) → typing.Union[pandas.core.frame.DataFrame, NoneType]
```

Convert an item to the specified type of model component.

Converts a junction to either a WellHead, Source or Sink. If more than one item matches the search context then it will fail.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **to_component** (*str*) – Required. The model component type to change the specified item into. This must be an enumeration of ModelComponents, though only WellHead, Source and Sink are supported at this time.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.convert(component=ModelComponents.Junction, context="JN-01",
                  to_component=ModelComponents.WELL)
```

```
class sixgill.pipesim.ModelAbout (model)
    Bases: object
```

Provides details about the model, web API and Python Toolkit

```
webapi_version
    The PIPESIM web API version
```

filename

The model folder and file name

toolkit_version

The PIPESIM Python Toolkit version

unit_system

The engineering unit system

class sixgill.pipesim.**Task**(*model*)

Bases: object

networksimulation

ptprofilesimulation

class sixgill.pipesim.**AbstractSimulation**

Bases: object

get_state(*simulation_id*)

Returns the state of the simulation

Returns the state from the simulation.

Parameters **simulation_id** (*str*) – Required. The id of the simulation returned from start_simulation call.

Returns The state of the simulation. sim_status indicates whether the simulation is still running. messages contains the simulation messages returned by the simulator.

Return type *SimulationState*

Examples

```
>>> sim_state = model.tasks.networksimulation.get_state(simulation_id)
```

get_messages(*simulation_id*)

Returns the simulation progress message of the simulation

Returns the simulation progress message from the simulation.

Parameters **simulation_id** (*str*) – Required. The id of the simulation returned from start_simulation call.

Returns **messages** – The simulation messages returned by the simulator.

Return type list

Examples

```
>>> messages = model.tasks.networksimulation.get_messages(simulation_id)
```

get_summary(*simulation_id*)

Returns the warning, info and error messages of the simulation

Returns the warning, info and error messages from the simulation.

Parameters **simulation_id** (*str*) – Required. The id of the simulation returned from start_simulation call.

Returns summary – The keys of the dictionary are “Error”, “Info” and “Warning” and their values are a list of corresponding messages as shown in Stingray UI

Return type dictionary

Examples

```
>>> messages = model.tasks.networksimulation.get_summary(simulation_id)
```

get_results (*simulation_id*)

Returns the results from the simulation.

Returns the results from the simulation.

Parameters **simulation_id** (*str*) – Required. The id of the simulation returned from start_simulation call.

Returns The simulation results.

Return type dict

Examples

```
>>> sim_results = model.tasks.networksimulation.get_results(simulation_id)
```

delete_results (*simulation_id*)

Delete a simulation.

Delete a simulation.

Parameters **simulation_id** (*str*) – Required. The id of the simulation returned from start_simulation call.

Returns Whether the simulation is deleted or not.

Return type bool

Examples

```
>>> model.tasks.networksimulation.delete_results(simulation_id)
```

class sixgill.pipesim.**NetworkSimulation**(*model*)

Bases: *sixgill.pipesim.AbstractSimulation*

get_use_surface_conditions (*study: typing.Union[str, NoneType] = None*)

Returns the value of use surface conditions.

Returns the value of use surface conditions for the specified study. If the network simulation task does not exist in the study, a new one will be created.

Parameters **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The value of use surface conditions.

Return type bool

Examples

```
>>> use_surface = model.tasks.networksimulation.
    get_use_surface_conditions()
>>> use_surface = model.tasks.networksimulation.
    get_use_surface_conditions("Study 1")
```

set_use_surface_conditions (*value: bool, study: typing.Union[str, NoneType] = None*)

Sets the value of use surface conditions.

Sets the the value of use surface conditions for the specified study. If the network simulation task does not exist in the study, a new one will be created.

Parameters

- **value** (*bool*) – The value of use surface conditions. If true, set to use surface conditions. If false, set to use reservoir conditions.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Examples

```
>>> model.tasks.networksimulation.
    set_use_surface_conditions(True)
>>> model.tasks.networksimulation.
    set_use_surface_conditions(False, study = "Study 1")
```

get_conditions (*use_surface_condition: typing.Union[bool, NoneType] = None, study: typing.Union[str, NoneType] = None*) → *typing.Dict[KT, VT]*

Returns the study conditions.

Returns the network simulation conditions for the specified study. If the network simulation task does not exist in the study, a new one will be created.

Parameters

- **use_surface_condition** (*bool*) – Optional. get surface conditions or reservoir conditions. If not provided, default is the conditions used in the model.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The study conditions.

Return type dict

Examples

```
>>> studynetsim1 = model.tasks.networksimulation.get_conditions()
>>> studynetsim1 = model.tasks.networksimulation.get_conditions("Study 1")
>>> studynetsim1 = model.tasks.networksimulation.get_conditions(
    use_surface_condition=True, study="Study 1")
```

set_conditions(*boundaries: typing.Union[dict, NoneType] = {}, use_surface_condition: typing.Union[bool, NoneType] = None, study: typing.Union[str, NoneType] = None*)
 Sets the study conditions.

Sets the study conditions for the specified study. If the network simulation task does not exist in the study, a new one will be created.

Parameters

- **use_surface_condition** (*bool*) – Optional. set surface conditions or reservoir conditions. If not provided, default is the conditions used in the model.
- **boundaries** (*dict*) – Optional. The boundary conditions to be set on the study.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns Whether the set succeeded (True) or not (False).

Return type bool

Examples

```
>>> model.tasks.networksimulation.set_conditions(
    boundaries={"Well:VertComp":{"Pressure":NAN, "Temperature":150,
    "FlowRateType":"GasFlowRate", "GasFlowRate":5}})
>>> model.tasks.networksimulation.set_conditions(use_surface_condition=False,
    boundaries={"Well:VertComp":{"Pressure":NAN, "Temperature":150,
    "FlowRateType":"GasFlowRate", "GasFlowRate":5}},
    study = "Study 1")
```

reset_conditions(*study=None*)

Resets the study conditions to be synchronized with the model.

Resets the study conditions to be synchronized with the model for the specified study. If the network simulation task does not exist in the study, a new one will be created.

Parameters **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns Whether the reset succeeded (True) or not (False).

Return type bool

Examples

```
>>> model.tasks.networksimulation.reset_conditions()
>>> model.tasks.networksimulation.reset_conditions(study = "Study 1")
```

generate_engine_files(*folder_path: typing.Union[str, NoneType] = None, study: typing.Union[str, NoneType] = None*) → typing.List[]

Create engine files for the specified study.

Generate the engine file(s) for the requested study. The files are written to the specified folder, or to the current model folder if one is not given.

Parameters

- **folder_path** (*str*) – Optional. The folder location where the generated files are written to. If not specified, the files will be written to current pipesim model file folder.
- **study** (*str*) – Optional. The study name.

Returns The list of engine files generated.

Return type List[str]

Examples

```
>>> engine_files = model.tasks.networksimulation.generate_engine_files()
>>> engine_files = model.tasks.networksimulation.generate_engine_files(
    study='Study 1')
>>> engine_files = model.tasks.networksimulation.generate_engine_files(
    folder_path="D:\Temp", study='Study 1')
```

run(*profile_variables: typing.Union[list, NoneType] = None, system_variables: typing.Union[list, NoneType] = None, boundaries: typing.Union[dict, NoneType] = {}, use_surface_condition: typing.Union[bool, NoneType] = None, study: typing.Union[str, NoneType] = None*)
Runs a network simulation.

Runs a network simulation. Optionally the boundaries, variables may be set at the same time avoiding the need to use `set_conditions()`. If the network simulation task does not exist in the study, a new one will be created. The method waits until the simulation has finished before returning, and then returns the results.

Parameters

- **use_surface_condition** (*bool*) – Optional. use surface conditions or reservoir conditions. If not provided, default is the conditions used in the model.
- **boundaries** (*dict*) – Optional. The boundary conditions to be set on the study.
- **profile_variables** (*list*) – Optional. The list of profile variables to be returned with the simulation profile results. If omitted then all the possible profile variables are returned.
- **system_variables** (*list*) – Optional. The list of system variables to be returned with the simulation system results. If omitted then all the possible system variables are returned.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The simulation results.

Return type dict

Examples

```
>>> sim_results = model.tasks.networksimulation.run(
    use_surface_condition=False,
    boundaries={"Well1.Comp1": {"Pressure": NAN,
                                "LiquidFlowRate": 200,
                                "FlowRateType": "LiquidFlowRate",
                                }
    },
```

```
profile_variables=OutputVariables.Profile.GASFIELD,
system_variables=OutputVariables.System.GASFIELD)
```

start(*profile_variables: typing.Union[list, NoneType] = None, system_variables: typing.Union[list, NoneType] = None, boundaries: typing.Union[dict, NoneType] = {}, use_surface_condition: typing.Union[bool, NoneType] = None, study: typing.Union[str, NoneType] = None*) → str

Starts running a simulation of the specified type, returning the simulation id immediately which can be used later to check the status and get the results. If the network simulation task does not exist in the study, a new one will be created.

Parameters

- **use_surface_condition** (*bool*) – Optional. use surface conditions or reservoir conditions. If not provided, default is the conditions used in the model.
- **parameters** (*dict*) – Optional. The parameters and respective values to set. For PT Profile, it must contain the name of the StartNode (Well or Source)
- **boundaries** (*dict*) – Optional. The boundary conditions to be set on the study.
- **sensitivities** (*dict*) – Optional. For PT Profile simulations, the sensitivities.
- **profile_variables** (*list*) – Optional. The list of profile variables to be returned with the simulation profile results. If omitted then all the possible profile variables are returned.
- **system_variables** (*list*) – Optional. The list of system variables to be returned with the simulation system results. If omitted then all the possible system variables are returned.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The simulation id, which can be used to access simulation status and get the results.

Return type str

Examples

```
>>> sim_id = model.tasks.networksimulation.start(use_surface_condition=False,
        boundaries={"Well1.Compl": {"Pressure": NAN,
        "LiquidFlowRate": 200,
        "FlowRateType": "LiquidFlowRate",
        },
        profile_variables=OutputVariables.Profile.GASFIELD,
        system_variables=OutputVariables.System.GASFIELD)
```

class sixgill.pipesim.PTProfileSimulation(*model*)

Bases: *sixgill.pipesim.AbstractSimulation*

get_conditions(*producer, study: typing.Union[str, NoneType] = None*)

Returns the study conditions.

Returns the PT profile simulation conditions for the specified study. If the PT profile simulation task does not exist in the study, a new one will be created.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The PT profile settings in a dictionary format.

Return type dict

Examples

```
>>> studynetsim1 = model.tasks.ptprofilesimulation.get_conditions("Well 1")
>>> studynetsim1 = model.tasks.ptprofilesimulation.get_conditions("Well 1",
                                                                    "Study 2")
```

set_conditions (*producer*, *parameters*={}, *study*=None)

Sets the study conditions.

Sets the PT profile simulation task conditions for the specified study. If the PT profile simulation task does not exist in the study, a new one will be created.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.
- **parameters** (*dict*) – Required. The conditions for PT profile task to be set on the study.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns Whether the set succeeded (True) or not (False).

Return type bool

Examples

```
>>> model.tasks.ptprofilesimulation.set_conditions("Well 1",
                                                    parameters={"InletPressure":3000, "OutletPressure":150,
                                                                    "FlowRateType":"GasFlowRate", "GasFlowRate":5})
>>> model.tasks.ptprofilesimulation.set_conditions("Well 1",
                                                    parameters={"InletPressure":3000, "OutletPressure":150,
                                                                    "FlowRateType":"GasFlowRate", "GasFlowRate":5},
                                                    study="Study 2")
```

get_calculated_variables (*producer*, *study*=None) → typing.List[]

get the possible calculated variable list.

Get the list of possible calculated variable for the PT profile simulation task. If the PT profile simulation task does not exist in the study, a new one will be created.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.

- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns a list of possible calculated variable names that can be applied to this task.

Return type list

Examples

```
>>> model.tasks.ptprofilesimulation.get_calculated_variables("Well 1")
>>> model.tasks.ptprofilesimulation.get_calculated_variables("Well 1",
    study="Study 2")
```

get_sensitivity_variables(*producer, study=None*) → typing.List[]

Get the available sensitivity variable list.

Get the list of possible sensitivity variables for the PT profile simulation task. If the PT profile simulation task does not exist in the study, a new one will be created.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns a list of sensitivity variables names that can be applied to this task.

Return type list

Examples

```
>>> model.tasks.ptprofilesimulation.get_sensitivity_variables("Well 1")
>>> model.tasks.ptprofilesimulation.get_sensitivity_variables("Well 1",
    study="Study 2")
```

generate_engine_files(*producer, folder_path: typing.Union[str, NoneType] = None, study: typing.Union[str, NoneType] = None*) → typing.List[]

Create engine files for the specified study.

Generate the engine file(s) for the requested study. The files are written to the specified folder, or to the current model folder if one is not given.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.
- **folder_path** (*str*) – Optional. The folder location where the generated files are written to. If not specified, the files will be written to current pipesim model folder.
- **study** (*str*) – Optional. The study name.

Returns The list of engine files generated.

Return type List[str]

Examples

```
>>> engine_files = model.tasks.ptprofilesimulation.generate_engine_files(
    producer="Well 1")
>>> engine_files = model.tasks.ptprofilesimulation.generate_engine_files(
    producer="Well 1", folder_path="D:\Temp")
>>> engine_files = model.tasks.ptprofilesimulation.generate_engine_files(
    producer="Well 1", folder_path="D:\Temp",
    study="Study 1")
```

run(*producer*, *parameters*={}, *profile_variables*=None, *system_variables*=None, *study*=None)

Runs a PT Profile simulation.

Runs a PT Profile simulation. Optionally the parameters and variables may be set at the same time avoiding the need to use `set_conditions()`. If the PT profile simulation task does not exist in the study, a new one will be created. The method waits until the simulation has finished before returning, and then returns the results.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.
- **parameters** (*dict*) – Optional. The parameters and respective values to set.
- **profile_variables** (*list*) – Optional. The list of profile variables to be returned with the simulation profile results. If omitted then all the possible profile variables are returned.
- **system_variables** (*list*) – Optional. The list of system variables to be returned with the simulation system results. If omitted then all the possible system variables are returned.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The simulation results.

Return type dict

Examples

```
>>> sim_results = model.tasks.ptfilesimulation.run( "Well 1",
    parameters={Parameters.PTProfileSimulation.OUTLETPRESSURE:200,
        Parameters.PTProfileSimulation.SENSITIVITYVARIABLE:
        {Parameters.PTProfileSimulation.SensitivityVariable.VARIABLE:
        "VertComp/ReservoirPressure",
        Parameters.PTProfileSimulation.SensitivityVariable.VALUES:
        [2000,3000,4000]}
    },
    profile_variables=OutputVariables.Profile.GASFIELD,
    system_variables=OutputVariables.System.GASFIELD)
```

start(*producer*, *parameters*={}, *profile_variables*=None, *system_variables*=None, *study*=None)

Starts running a PT Profile simulation, returning the simulation id immediately which can be used later to check the status and get the results. If the PT profile simulation task does not exist in the study, a new one will be created.

Parameters

- **producer** (*str*) – Required. The name of the producer for the PT profile simulation task.
- **parameters** (*dict*) – Optional. The parameters and respective values to set.
- **profile_variables** (*list*) – Optional. The list of profile variables to be returned with the simulation profile results. If omitted then all the possible profile variables are returned.
- **system_variables** (*list*) – Optional. The list of system variables to be returned with the simulation system results. If omitted then all the possible system variables are returned.
- **study** (*str*) – Optional. The name of the study. If no name is specified and there is one default study, the default study will be used. If there are multiple studies in the model, study name is required.

Returns The simulation id, which can be used to access simulation status and get the results.

Return type str

Examples

```
>>> sim_id = model.tasks.ptfilesimulation.start( "Well 1",
        parameters={Parameters.PTProfileSimulation.OUTLETPRESSURE:200,
                    Parameters.PTProfileSimulation.SENSITIVITYVARIABLE:
                    {Parameters.PTProfileSimulation.SensitivityVariable.VARIABLE:
                    "VertComp/ReservoirPressure",
                    Parameters.PTProfileSimulation.SensitivityVariable.VALUES:
                    [2000,3000,4000]}
                    }
        profile_variables=OutputVariables.Profile.GASFIELD,
        system_variables=OutputVariables.System.GASFIELD)
```

11.1.2 sixgill.compositional module

class sixgill.compositional.**FluidSettings**(*model*)

Bases: object

Primary class for operating on fluids

fluid_type

The current fluid type set for the model

Returns One of the fluid mode defined in definitions FluidType class

Return type *FluidType*

compositional

The compositional fluid settings

class sixgill.compositional.**CompositionalFluidSettings**(*model*)

Bases: object

pvt_package

The compositional fluid package set for the model

Returns One of the fluid flash package type defined in definitions PVTPackage class

Return type *PVTPackage*

equation_of_state

The compositional fluid equation of state

Returns One of the fluid equation of state defined in definitions EquationOfState class

Return type *EquationOfState*

viscosity_correlation

The compositional fluid viscosity correlation method

Returns One of the fluid viscosity correlation method defined in definitions ViscosityCorrelationMethod class

Return type *ViscosityCorrelationMethod*

volume_shift_correlation

The compositional fluid volume shift correlation method

Returns One of the fluid volume shift correlation method defined in definitions VolumeShiftCorrelationMethod class

Return type VolumeShiftCorrelation

critical_property_correlation

The compositional fluid critical property correlation method

Returns One of the fluid critical property correlation method defined in definitions CriticalPropertyCorrelationMethod class

Return type CriticalPropertyCorrelation

thermal_coefficient_correlation

The compositional fluid thermal coefficient correlation method

Returns One of the fluid thermal coefficient correlation method defined in definitions ThermalCoefficientsCorrelationMethod class

Return type ThermalCoefficientsCorrelation

acf_correlation

The compositional fluid Acf correlation method

Returns One of the fluid Acf correlation method defined in definitions AcfCorrelationMethod class

Return type ThermalCoefficientsCorrelation

salinity_model

The compositional fluid sanity model type

Returns One of the fluid salinity model type defined in definitions SalinityModel class

Return type *SalinityModel*

select_components (*selected_components: list*) → None

Select the components that will be used by the compositional fluid in the model.

Parameters **selected_components** (*list*) – The list of fluid component names based on the fluid flash package.

Returns

Return type None

unselect_components(*unselected_components: list*)

Unselect the components that are no longer used by the compositional fluid in the model.

Parameters **unselected_components** (*list*) – The list of fluid component names based on the fluid flash package.

Returns If a component cannot be unselected, a `ValueError` will be raised to tell the reason.

Return type `None`

add_pseudocomponent(*component_name, parameters={}*) → `None`

Add a user defined fluid component that will be used by the compositional fluid in the model.

Parameters

- **component_name** (*str*) – The name of the new component.
- **parameters** (*dict*) – Optional. The parameters for the user defined component.

Returns If the component cannot be added, a `ValueError` will be raised to tell the reason.

Return type `None`

delete_pseudocomponent(*component_name*)

Delete a user defined fluid component that is no longer used by the compositional fluid in the model.

Parameters **component_name** (*str*) – The name of the user defined component.

Returns If the component cannot be deleted, a `ValueError` will be raised to tell the reason.

Return type `None`

get_composition(*context: typing.Union[str, NoneType] = None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None, **context_keywords*) → `pandas.core.frame.DataFrame`

Return fluid composition in the context.

Returns fluid composition in the context. It will fail if no item or more than one item is found in the context.

Parameters

- **context** (*str*) – Optional. The context string to look for compositional fluid.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of `component='value'`, where `component` is the model component of the item, and `value` is the value of the model component. Values can also be `ANY` (matches any value) or `NONE` (matches no value).

Returns The fluid composition data frame includes the following columns: `Component`, `SpecifiedFraction` (indexed by `Component`)

Return type `pandas.DataFrame`

Examples

```
>>> df = model.fluids.compositional.get_composition(context="C-Fluid")
>>> dfw = model.fluids.compositional.get_composition(
    CompositionalFluid="C-Fluid")
```

```
set_composition(context: typing.Union[str, NoneType] = None, value=None, model_context: typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType] = None,
                **context_keywords)
```

Set data frame to the fluid composition in the context.

Set data frame to the fluid composition in the context. It will fail if more than one item is found in the context.

Parameters

- **value** (*DataFrame* or *dict*) – Required. The data frame or dict to set to the fluid composition. The fluid composition data frame includes the following columns: Component, SpecifiedFraction (indexed by Component). If dict is used, keys are considered as Component and values are considered as SpecifiedFraction.
- **context** (*str*) – Optional. The context string to look for the compositional fluid.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Examples

```
>>> model.fluids.compositional.set_composition(context="C-Fluid", value = df)
>>> model.fluids.compositional.set_composition(CompositionalFluid="C-Fluid",
                                              value = df)
```

11.1.3 sixgill.model_setting module

```
class sixgill.model_setting.ModelSetting(model)
```

Bases: dict

A class to get and set the model and simulation settings

```
get(key, default=None)
```

```
items()
```

```
keys()
```

```
values()
```

```
pop(*args, **kwargs)
```

```
popitem(*args, **kwargs)
```

```
clear(*args, **kwargs)
```

```
update(*args, **kwargs)
```

```
setdefault(*args, **kwargs)
```

```
fromkeys(*args, **kwargs)
```

```
ambient_temperature
```

```
atmospheric_pressure
```

```
wind_speed
```

`soil_type`
`soil_conductivity`
`metocean_data_location`
`erosional_velocity_constant`
`corrosion_model`
`corrosion_efficiency`
`max_report_interval_length`
`print_computation_segment_result`
`max_computation_segment_length`
`network_solver_method`
`network_solver_tolerance`
`network_solver_max_iterations`
`single_branch_keywords`
`network_keywords_top`
`network_keywords_bottom`
`gis_elevation_interval`
`gis_elevation_max_points`
`gis_elevation_data_source`
`use_global_flow_correlation`
`use_global_heat_transfer`
`global_flow_correlation`

Return global flow correlation settings as a dictionary

`global_heat_transfer_option`

Return global heat transfer options as a dictionary

`get_flow_correlations` (*context*: `typing.Union[str, NoneType]` = `None`, *component*: `typing.Union[str, NoneType]` = `None`, *model_context*: `typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType]` = `None`, ***context_keywords*: `str`) → dict

Return flow correlation settings for wells or flowlines that match the given context. If no context specified, return all flow correlation settings for wells and flowlines.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of `component='value'`, where `component` is the model component of the item, and `value` is the value of the model component. Values can also be `ANY` (matches any value) or `NONE` (matches no value).

Returns A dictionary that contains flow correlation settings.

Return type dict

Examples

```
>>> model_settings.get_flow_correlations(context="Well 23")
>>> model_settings.get_flow_correlations()
```

get_heat_transfer_options(*context*: *typing.Union[str, NoneType]* = *None*, *component*: *typing.Union[str, NoneType]* = *None*, *model_context*: *typing.Union[sixgill.core.model_context.AbstractModelContext, NoneType]* = *None*, ***context_keywords*: *str*) → dict

Return heat transfer options for wells or flowlines that match the given context. If no context specified, return all heat transfer options for wells and flowlines.

Parameters

- **context** (*str*) – Optional. The context string to search in.
- **component** (*str*) – Optional. The type of the item.
- **model_context** (*AbstractModelContext*) – Optional. The context to search in.
- **context_keywords** (*str*) – Comma separated list of context search criteria in the form of component='value', where component is the model component of the item, and value is the value of the model component. Values can also be ANY (matches any value) or NONE (matches no value).

Returns A dictionary that contains heat transfer options.

Return type dict

Examples

```
>>> model_settings.get_heat_transfer_options(context="Well 23")
>>> model_settings.get_heat_transfer_options()
```

set_flow_correlations(*flow_correlations*)

Set flow correlation settings for well and flowlines from a dictionary. The key of the dictionary is the name of flowline or well that flow correlation is associated with and the value is a sub dictionary that contains flow correlation settings. If name is "Default", it is considered the global default flow correlation. The keys of sub dictionary are: VerticalMultiphaseSource: Vertical multiphase flow correlation source VerticalMultiphaseCorrelation: Vertical multiphase flow correlation name VerticalMultiphaseHoldupFactor: Vertical multiphase flow correlation holdup factor VerticalMultiphaseFrictionFactor: Vertical multiphase flow correlation friction factor HorizontalMultiphaseSource: Horizontal multiphase flow correlation source HorizontalMultiphaseCorrelation: Horizontal multiphase flow correlation name HorizontalMultiphaseHoldupFactor: Horizontal multiphase flow correlation holdup factor HorizontalMultiphaseFrictionFactor: Horizontal multiphase flow correlation friction factor SwapAngle: Vertical and horizontal swap angle SinglePhaseCorrelation: Single phase flow correlation name SinglePhaseFactor: Single phase flow correlation factor

set_heat_transfer_options(*heat_transfer_options*)

Set heat transfer option settings for well and flowlines from a dictionary. The key of the dictionary is the name of flowline or well that heat transfer option is associated with and the value is a sub dictionary that contains heat transfer options. If name is "Default", it is considered as global heat transfer options. The keys of sub dictionary are: PipeBurialMethod: Pipe burial method InsideFilmCoeffMethod: Inside film coefficient method UValueMultiplier: U value multiplier EnableHydrateSubcoolCalc: boolean flag indicating whether to enable hydrate subcooling calculation

```
class sixgill.model_setting.GlobalFlowCorrelation(get_web_api,          get_flow_corr,
                                                  set_flow_corr)
```

Bases: object

A class to get and set the global flow correlations.

vertical_source

vertical_correlation

vertical_holdup_factor

vertical_friction_factor

horizontal_source

horizontal_correlation

horizontal_holdup_factor

horizontal_friction_factor

```
class sixgill.model_setting.GlobalHeatTransferOption(get_web_api,          get_htc_opt,
                                                       set_htc_opt)
```

Bases: object

A class to get and set the global heat transfer options

pipe_burial_method

inside_film_coeff_method

u_value_multiplier

hydrate_subcooling_calc

11.1.4 sixgill.utilities module

The utilities submodule provides useful methods for working with Excel.

```
sixgill.utilities.active_sheet()  
    Return the currently active Excel worksheet
```

```
sixgill.utilities.range_to_dataframe(excel_range=[], entriesbyrow=False)  
    Convert an excel range which is a list of lists to a panda DataFrame
```

```
sixgill.utilities.range_to_dictionary(excel_range=[], entriesbyrow=False)  
    Convert an excel range which is a list of lists to a dictionary of dictionaries
```

```
sixgill.utilities.current_folder()  
    Return the folder (path) for the workbook
```

```
sixgill.utilities.worksheet_last_row(worksheet=None)  
    Return the last row with data given a specified column
```

```
sixgill.utilities.worksheet_last_column(worksheet=None)  
    Return the last row with data given a specified column
```

```
sixgill.utilities.get_model_session(filename, units='PIPESIM_FIELD')  
    Return the active model session that the workbook is connected to
```

```
sixgill.utilities.convert_tubing_bmd(bmd: typing.List[T])  
    Convert tubing BMD to TMD and length
```

11.1.5 sixgill.definitions module

The definitions sub-module maintains the enumerations and static classes that are available in the PIPESIM Python Toolkit.

```
class sixgill.definitions.Units
    Bases: object

    Available unit of measurement categories

    METRIC = 'PIPESIM_METRIC'

    SI = 'PIPESIM_SI'

    FIELD = 'PIPESIM_FIELD'

class sixgill.definitions.Tasks
    Bases: object

    Available PIPESIM simulation tasks

    NETWORKSIMULATION = 'NetworkSim'

    PTPROFILESIMULATION = 'PTProfileSim'

class sixgill.definitions.Connection
    Bases: object

    SOURCE = 'Source'

    DESTINATION = 'Destination'

    SOURCEPORT = 'Source Port'

    class Separator
        Bases: object

        TOP = 'Top Port'

        BOTTOM = 'Bottom Port'

        BOOT = 'Boot Port'

class sixgill.definitions.SimulationState
    Bases: object

    RUNNING = 'Running'

    COMPLETED = 'Completed'

    FAILED = 'Failed'

class sixgill.definitions.ModelComponents
    Bases: object

    Available model components

    BLACKOILFLUID = 'BlackOilFluid'

    BOUNDARY = 'Boundary'

    CASING = 'Casing'

    CHECKVALVE = 'CheckValve'

    CHOKE = 'Choke'

    COMPLETION = 'Completion'
```


COMPLETIONCONINGPOINT = 'CompletionConingPoint'
COMPLETIONMODEL = 'CompletionModel'
COMPOSITIONALFLUID = 'CompositionalFluid'
COMPRESSOR = 'Compressor'
ENGINEKEYWORDS = 'EngineKeywords'
ESP = 'ESP'
EXPANDER = 'Expander'
FILEBASEDFLUID = 'FileBasedFluid'
FLOWLINE = 'Flowline'
FLUIDCOMPONENT = 'FluidComponent'
GASLIFTINJECTION = 'GasLiftInjection'
GENERIC EQUIPMENT = 'GenericEquipment'
GRAVELPACK = 'GravelPack'
HEATEXCHANGER = 'HeatExchanger'
INJECTOR = 'Injector'
IPRBACKPRESSURE = 'IPRBackPressure'
IPRDARCY = 'IPRDarcy'
IPRFETKOVITCH = 'IPRFetkovitch'
IPRFORCHHEIMER = 'IPRForchheimer'
IPRHORIZONTALPI = 'IPRHorizontalPI'
IPRHYDRAULICFRACTURE = 'IPRHydraulicFracture'
IPRJONES = 'IPRJones'
IPRPIMODEL = 'IPRPIModel'
IPRPSSBABUODEH = 'IPRPSSBabuOdeh'
IPRSSJOSHI = 'IPRSSJoshi'
IPRVOGEL = 'IPRVogel'
JUNCTION = 'Junction'
LINER = 'Liner'
MEASUREMENTPOINT = 'MeasurementPoint'
MFLFLUID = 'MFLFluid'
MULTIPLIERADDER = 'MultiplierAdder'
MULTIPHASEBOOSTER = 'MultiphaseBooster'
GENERICBOOSTER = 'GenericBooster'
ONESUBSEABOOSTER = 'OneSubseaBooster'
OPEN_HOLE = 'OpenHole'
PACKER = 'Packer'

```
PCP = 'PCP'
PUMP = 'Pump'
PVTFLUID = 'PVTFluid'
RODPUMP = 'RodPump'
SINGLEPHASESEPARATOR = 'SinglephaseSeparator'
SINK = 'Sink'
SLIDINGSLEEVE = 'SlidingSleeve'
SOURCE = 'Source'
STUDY = 'Study'
SUBSURFACESAFETYVALVE = 'SubsurfaceSafetyValve'
THREEPHASESEPARATOR = 'ThreePhaseSeparator'
TUBING = 'Tubing'
TUBINGPLUG = 'TubingPlug'
TWOPHASESEPARATOR = 'TwoPhaseSeparator'
WATERTEMPVELOCITYSURVEY = 'WaterTempVelocitySurvey'
WELL = 'Well'
NETWORKSIMULATION = 'NetworkSim'
PTPROFILESIMULATION = 'PTProfileSim'

class sixgill.definitions.Parameters
    Bases: object

    Available parameters by model component

    class Location
        Bases: object

        LATITUDE = 'Latitude'
        LONGITUDE = 'Longitude'
        CLUSTEREDLATITUDE = 'SchematicLatitude'
        CLUSTEREDLONGITUDE = 'SchematicLongitude'
        ELEVATION = 'Elevation'

    class Parameters.SchematicPosition
        Bases: object

        X_COORD = 'X'
        Y_COORD = 'Y'

    class Parameters.DownholeLocation
        Bases: object

        TOPMEASUREDDEPTH = 'TopMeasuredDepth'
        BOTTOMMEASUREDDEPTH = 'BottomMeasuredDepth'

    class Parameters.ModelComponent
        Bases: object
```

```
NAME = 'Name'
ISACTIVE = 'IsActive'

class Parameters.DownholeComponent
    Bases: sixgill.definitions.ModelComponent, sixgill.definitions.DownholeLocation

class Parameters.SurfaceComponent
    Bases: sixgill.definitions.ModelComponent, sixgill.definitions.Location,
    sixgill.definitions.SchematicPosition

class Parameters.DownholeOrSurfaceComponent
    Bases: sixgill.definitions.ModelComponent, sixgill.definitions.DownholeLocation,
    sixgill.definitions.Location, sixgill.definitions.SchematicPosition

class Parameters.AbstractTubingSection
    Bases: sixgill.definitions.DownholeComponent

    ISFLOWPRESENTINSIDE = 'IsFlowPresentInside'
    TOPMEASUREDDEPTH = 'TopMeasuredDepth'
    GRADE = 'Grade'
    DENSITY = 'Density'
    BOREHOLEDIAMETER = 'BoreholeDiameter'
    CEMENTTOP = 'CementTop'
    CEMENTDENSITY = 'CementDensity'
    ANNULUSMATERIALTYPE = 'AnnulusMaterialType'
    ANNULUSMATERIALDENSITY = 'AnnulusMaterialDensity'
    INNERDIAMETER = 'InnerDiameter'
    ROUGHNESS = 'Roughness'
    WALLTHICKNESS = 'WallThickness'
    LENGTH = 'Length'

class Parameters.AssociatedToFluid
    Bases: object

    ASSOCIATEDBLACKOILFLUID = 'AssociatedBlackOilFluid'
    ASSOCIATEDCOMPOSITIONALFLUID = 'AssociatedCompositionalFluid'
    ASSOCIATEDMFLFLUID = 'AssociatedMFLFluid'

class Parameters.CanBeTemplate
    Bases: object

    ISTEMPLATE = 'IsTemplate'
    ISBUILTINTEMPLATE = 'IsBuiltinTemplate'

class Parameters.SharedPumpParameters
    Bases: object

    EFFICIENCY = 'Efficiency'
    POWER = 'Power'
    PRESSURE = 'Pressure'
```

```
    PRESSUREDIFFERENTIAL = 'PressureDifferential'
    PRESSURERATIO = 'PressureRatio'
    USECURVES = 'UseCurves'
class Parameters.SharedPumpCatalogParameters
    Bases: object
    MANUFACTURER = 'Manufacturer'
    MODEL = 'Model'
    BASESPEED = 'BaseSpeed'
    OPERATINGSPEED = 'OperatingSpeed'
    HEADFACTOR = 'HeadFactor'
    SHOWPOWER = 'ShowPower'
class Parameters.FluidLink
    Bases: object
    RESERVOIRPRESSURE = 'ReservoirPressure'
    RESERVOIRTEMPERATURE = 'ReservoirTemperature'
    COMPLETIONTESTTYPE = 'CompletionTestType'
    USETESTDATA = 'UseTestData'
    USECONINGDATA = 'UseConingData'
    CONEDGASSG = 'ConedGasSG'
    USEGASRATIO = 'UseGasRatio'
    USEWATERRATIO = 'UseWaterRatio'
    GOR = 'GOR'
    OGR = 'OGR'
    GLR = 'GLR'
    LGR = 'LGR'
    GWR = 'GWR'
    WGR = 'WGR'
    WATERCUT = 'WaterCut'
    USEFLUIDOVERRIDES = 'UseFluidOverrides'
    OVERRIDESINITIALIZED = 'OverridesInitialized'
class Parameters.Fluid
    Bases: sixgill.definitions.CanBeTemplate, sixgill.definitions.FluidLink
    COMMENT = 'Comment'
    HASOVERRIDE = 'HasOverride'
class Parameters.FileBasedFluid
    Bases: sixgill.definitions.Fluid
    FLUIDFILENAME = 'FluidFileName'
```

```
class Parameters.About
    Bases: object

    TOOLKITVERSION = 'ToolkitVersion'
    WEBAPIVERSION = 'WebAPIVersion'
    MODELFILENAME = 'ModelFilename'
    UNITSYSTEM = 'UnitSystem'

class Parameters.SimulationSetting
    Bases: object

    AMBIENTTEMPERATURE = 'AmbientTemperature'
    ATMOSPHERICPRESSURE = 'AtmosphericPressure'
    WINDSPEED = 'WindSpeed'
    SOILTYPE = 'SoilType'
    SOILCONDUCTIVITY = 'SoilConductivity'
    METOCEANDATALOCATION = 'MetoceanDataLocation'
    EROSIONALVELOCITYCONSTANT = 'ErosionalVelocityConstant'
    CORROSIONMODEL = 'CorrosionModel'
    CORROSIONEFFICIENCY = 'CorrosionEfficiency'
    PIPESEGMENTATIONMAXREPORTINGINTERVAL = 'PipeSegmentationMaxReportingInterval'
    PIPESEGMENTATIONPRINTSEGMENTATIONRESULT = 'PipeSegmentationPrintSegmentationResult'
    PIPESEGMENTATIONMAXSEGMENTLENGTH = 'PipeSegmentationMaxSegmentLength'
    NETWORKSOLVERMETHOD = 'NetworkSolver'
    NETWORKSOLVERTOLERANCE = 'NetworkSolverTolerance'
    NETWORKSOLVERMAXITERATIONS = 'NetworkSolverMaxIterations'
    SINGLEBRANCHKEYWORDS = 'SingleBranchKeywords'
    NETWORKTOPKEYWORDS = 'NetworkTopKeywords'
    NETWORKBOTTOMKEYWORDS = 'NetworkBottomKeywords'
    GISELEVATIONINTERVAL = 'GISElevationInterval'
    GISELEVATIONMAXPOINTS = 'GISElevationMaxPoints'
    GISELEVATIONSOURCETYPE = 'GISElevationSourceType'
    USEGLOBALFLOWCORRELATIONS = 'UseGlobalFlowCorrelations'
    USEGLOBALHEATTRANSFER = 'UseGlobalHeatTransfer'

class Parameters.FlowCorrelation
    Bases: object

    SWAPANGLE = 'SwapAngle'
    OVERRIDEGLOBAL = 'OverrideGlobal'

    class SinglePhase
        Bases: object
```

```
CORRELATION = 'SinglePhaseCorrelation'
FACTOR = 'SinglePhaseFactor'

class Parameters.FlowCorrelation.Multiphase
    Bases: object

    class Vertical
        Bases: object

        SOURCE = 'VerticalMultiphaseSource'

        CORRELATION = 'VerticalMultiphaseCorrelation'

        HOLDUPFACTOR = 'VerticalMultiphaseHoldupFactor'

        FRICTIONFACTOR = 'VerticalMultiphaseFrictionFactor'

    class Parameters.FlowCorrelation.Multiphase.Horizontal
        Bases: object

        SOURCE = 'HorizontalMultiphaseSource'

        CORRELATION = 'HorizontalMultiphaseCorrelation'

        HOLDUPFACTOR = 'HorizontalMultiphaseHoldupFactor'

        FRICTIONFACTOR = 'HorizontalMultiphaseFrictionFactor'

class Parameters.HeatTransferOption
    Bases: object

    PIPEBURIALMETHOD = 'PipeBurialMethod'

    INSIDEFILMCOEFFMETHOD = 'InsideFilmCoeffMethod'

    UVALUEMULTIPLIER = 'UValueMultiplier'

    ENABLEHYDRATESUBCOOLCALC = 'EnableHydrateSubcoolCalc'

    OVERRIDEGLOBAL = 'OverrideGlobal'

class Parameters.BlackOilFluid
    Bases: sixgill.definitions.Fluid

    API = 'API'

    FRACTIONCO2 = 'CO2Fraction'

    FRACTIONCO = 'COFraction'

    GASSPECIFICGRAVITY = 'GasSpecificGravity'

    GLR = 'GLR'

    GOR = 'GOR'

    USEGASRATIO = 'UseGasRatio'

    USEWATERRATIO = 'UseWaterRatio'

    GWR = 'GWR'

    FRACTIONH2 = 'H2Fraction'

    FRACTIONH2S = 'H2SFraction'

    FRACTIONN2 = 'N2Fraction'

    WATERCUT = 'WaterCut'
```

```
WATERSPECIFICGRAVITY = 'WaterSpecificGravity'
DEADOILDENSITY = 'DeadOilDensity'
USEDEADOILDENSITY = 'UseDeadOilDensity'
DEADOILVISCOSITYCORR = 'DeadOilViscosityCorr'
DEADOILTEMPERATURE1 = 'DeadOilTemperature1'
DEADOILTEMPERATURE2 = 'DeadOilTemperature2'
DEADOILVISCOSITY1 = 'DeadOilViscosity1'
DEADOILVISCOSITY2 = 'DeadOilViscosity2'
LIVEOILVISCOSITYCORR = 'LiveOilViscosityCorr'
LIQUIDVISCOSITYCALC = 'LiquidViscosityCalc'
USEBRAUNERULLMANEQUATION = 'UseBraunerUllmanEquation'
USERWATERCUTCUTOFF = 'UserWatercutCutOff'
UNDERSATURATEDOILVISCOSITYPARA = 'UndersaturatedOilViscosityParA'
UNDERSATURATEDOILVISCOSITYPARB = 'UndersaturatedOilViscosityParB'
UNDERSATURATEDOILVISCOSITYCORR = 'UndersaturatedOilViscosityCorr'
VANDUSERK1 = 'VandUserk1'
VANDUSERK2 = 'VandUserk2'
RICHARDSONKOIW = 'Richardsonkoiw'
RICHARDSONKWIO = 'Richardsonkwio'
OGR = 'OGR'
LGR = 'LGR'
WGR = 'WGR'
ISTEMPLATE = 'IsTemplate'
ISBUILTINTEMPLATE = 'IsBuiltinTemplate'
class SinglePointCalibration
    Bases: object
    SOLUTIONGAS = 'SolutionGas'
    OILFVFCORRELATION = 'OilFVFCorrelation'
    LIVEOILVISCCORRELATION = 'LiveOilViscCorrelation'
    GASCOMPRESSCORRELATION = 'GasCompressCorrelation'
    GASVISCCORRELATION = 'GasViscCorrelation'
    ABOVEBBPType = 'AboveBBPType'
    BELOWBBPType = 'BelowBBPType'
    BUBBLEPOINTSATGAS_VALUE = 'BubblepointSatGas_Value'
    BUBBLEPOINTSATGAS_PRESSURE = 'BubblepointSatGas_Pressure'
    BUBBLEPOINTSATGAS_TEMPERATURE = 'BubblepointSatGas_Temperature'
```

```
ABOVEBBPDENSITY_VALUE = 'AboveBBPDensity_Value'
ABOVEBBPDENSITY_PRESSURE = 'AboveBBPDensity_Pressure'
ABOVEBBPDENSITY_TEMPERATURE = 'AboveBBPDensity_Temperature'
ABOVEBBPOFVF_VALUE = 'AboveBBPOFVF_Value'
ABOVEBBPOFVF_PRESSURE = 'AboveBBPOFVF_Pressure'
ABOVEBBPOFVF_TEMPERATURE = 'AboveBBPOFVF_Temperature'
BELOWBBPDENSITY_VALUE = 'BelowBBPDensity_Value'
BELOWBBPDENSITY_PRESSURE = 'BelowBBPDensity_Pressure'
BELOWBBPDENSITY_TEMPERATURE = 'BelowBBPDensity_Temperature'
BELOWBBPOFVF_VALUE = 'BelowBBPOFVF_Value'
BELOWBBPOFVF_PRESSURE = 'BelowBBPOFVF_Pressure'
BELOWBBPOFVF_TEMPERATURE = 'BelowBBPOFVF_Temperature'
BELOWBBPLIVEOILVISCOSITY_VALUE = 'BelowBBPLiveOilViscosity_Value'
BELOWBBPLIVEOILVISCOSITY_PRESSURE = 'BelowBBPLiveOilViscosity_Pressure'
BELOWBBPLIVEOILVISCOSITY_TEMPERATURE = 'BelowBBPLiveOilViscosity_Temperature'
BELOWBBPGASVISCOSITY_VALUE = 'BelowBBPGasViscosity_Value'
BELOWBBPGASVISCOSITY_PRESSURE = 'BelowBBPGasViscosity_Pressure'
BELOWBBPGASVISCOSITY_TEMPERATURE = 'BelowBBPGasViscosity_Temperature'
BELOWBBPGASZ_VALUE = 'BelowBBPGasZ_Value'
BELOWBBPGASZ_PRESSURE = 'BelowBBPGasZ_Pressure'
BELOWBBPGASZ_TEMPERATURE = 'BelowBBPGasZ_Temperature'

class Parameters.BlackOilFluid.ThermalData
    Bases: object

    GASHEATCAPACITY = 'GasHeatCapacity'
    OILHEATCAPACITY = 'OilHeatCapacity'
    WATERHEATCAPACITY = 'WaterHeatCapacity'
    GASCONDUCTIVITY = 'GasConductivity'
    OILCONDUCTIVITY = 'OilConductivity'
    WATERCONDUCTIVITY = 'WaterConductivity'
    ENTHALPYCALCMETHOD = 'EnthalpyCalcMethod'
    LATENTHEATOFVAPORIZATION = 'LatentHeatOfVaporization'

class Parameters.Boundary
    Bases: object

    USEPQCURVE = 'UsePQCurve'
    USEGASRATIO = 'UseGasRatio'
    USEWATERRATIO = 'UseWaterRatio'
    ISSURFACECONDITION = 'IsSurfaceCondition'
```



```
TEMPERATURE = 'Temperature'
PRESSURE = 'Pressure'
GASFLOWRATE = 'GasFlowRate'
LIQUIDFLOWRATE = 'LiquidFlowRate'
MASSFLOWRATE = 'MassFlowRate'
GOR = 'GOR'
OGR = 'OGR'
GLR = 'GLR'
LGR = 'LGR'
WATERCUT = 'WaterCut'
GWR = 'GWR'
WGR = 'WGR'
FLOWRATETYPE = 'FlowRateType'
COMPLETIONNAME = 'CompletionName'
ISBOTTOMCOMPLETION = 'IsBottomCompletion'

class Parameters.Casing
    Bases: sixgill.definitions.AbstractTubingSection

class Parameters.CheckValve
    Bases: sixgill.definitions.SurfaceComponent
    CHECKVALVESETTING = 'CheckValveSetting'

class Parameters.Choke
    Bases: sixgill.definitions.DownholeOrSurfaceComponent
    ADJUSTSUBCRITICALCORRELATION = 'AdjustSubcriticalCorrelation'
    BEANSIZE = 'BeanSize'
    BOTHPHASESFLOWCOEFFICIENT = 'BothPhasesFlowCoefficient'
    CALCULATECRITICALPRESSURERATIO = 'CalculateCriticalPressureRatio'
    GASEXPANSIONFACTOR = 'GasExpansionFactor'
    CRITICALCORRELATION = 'CriticalCorrelation'
    CRITICALPRESSURERATIO = 'CriticalPressureRatio'
    DISCHARGECOEFFICIENT = 'DischargeCoefficient'
    GASPHASEFLOWCOEFFICIENT = 'GasPhaseFlowCoefficient'
    HEATCAPACITYRATIO = 'HeatCapacityRatio'
    USEFLOWRATEFORCRITICALFLOW = 'UseFlowrateForCriticalFlow'
    USEPRESSURERATIOFORCRITICALFLOW = 'UsePressureRatioForCriticalFlow'
    USESONICDOWNSTREAMVELOCITYFORCRITICALFLOW = 'UseSonicDownstreamVelocityForCriticalFlow'
    USESONICUPSTREAMVELOCITYFORCRITICALFLOW = 'UseSonicUpstreamVelocityForCriticalFlow'
    LIQUIDPHASEFLOWCOEFFICIENT = 'LiquidPhaseFlowCoefficient'
```

```
SUBCRITICALCORRELATION = 'SubCriticalCorrelation'
TOLERANCE = 'Tolerance'
UPSTREAMPIPEID = 'UpstreamPipeID'
PRINTDETAILED CALCULATIONS = 'PrintDetailedCalculations'
GASFLOWRATELIMIT = 'GasFlowRateLimit'
LIQUIDFLOWRATELIMIT = 'LiquidFlowRateLimit'
MASSFLOWRATELIMIT = 'MassFlowRateLimit'
WATERFLOWRATELIMIT = 'WaterFlowRateLimit'
OILFLOWRATELIMIT = 'OilFlowRateLimit'
FLOWRATELIMITTYPE = 'FlowRateLimitType'

class Parameters.Completion
    Bases:
        sixgill.definitions.DownholeComponent, sixgill.definitions.AssociatedToFluid,
        sixgill.definitions.FluidLink

    DIAMETER = 'Diameter'
    LENGTH = 'Length'
    PHASEANGLE = 'PhaseAngle'
    SHOTDENSITY = 'ShotDensity'
    PENETRATIONDEPTH = 'PenetrationDepth'
    FLUIDENTRYTYPE = 'FluidEntryType'
    GEOMETRYPROFILETYPE = 'GeometryProfileType'
    RESERVOIRPRESSURE = 'ReservoirPressure'
    RESERVOIRTEMPERATURE = 'ReservoirTemperature'
    TESTPOINTS = 'CompletionTestPoints'
    IPRMODEL = 'IPRModel'

class Parameters.CompletionConingPoint
    Bases: object

    LIQUIDFLOWRATE = 'LiquidFlowRate'
    GOR = 'GOR'
    WATERCUT = 'WaterCut'

class Parameters.CompositionalFluid
    Bases: sixgill.definitions.Fluid

    COMPONENTFRACTIONTYPE = 'ComponentFractionType'
    USETUNEDFRACTIONS = 'UseTunedFractions'
    SHOWALLCOMPONENTS = 'ShowAllComponents'
    LIQUIDVISCOSITYCALC = 'LiquidViscosityCalc'
    USEBRAUNERULLMANEQUATION = 'UseBraunerUllmanEquation'
```

```
USERWATERCUTCUTOFF = 'UserWatercutCutOff'
VANDUSERK1 = 'VandUserk1'
VANDUSERK2 = 'VandUserk2'
RICHARDSONKOIW = 'Richardsonkoiw'
RICHARDSONKWIO = 'Richardsonkwio'
IONSODIUM = 'SodiumIon'
IONCALCIUM = 'CalciumIon'
IONMAGNESIUM = 'MagnesiumIon'
IONPOTASSIUM = 'PotassiumIon'
IONSTRONTIUM = 'StrontiumIon'
IONBARIUM = 'BariumIon'
IONIRON = 'IronIon'
IONCHLORIDE = 'ChlorideIon'
IONSULPHATE = 'SulphateIon'
IONBICARBONATE = 'BicarbonateIon'
IONBROMIDE = 'BromideIon'
TOTALDISSOLVEDSOLIDS = 'TotalDissolvedSolids'
SALTWATERDENSITYTYPE = 'SaltWaterDensityType'
SALTWATERDENSITY = 'SaltWaterDensity'
SALTWATERSALINITY = 'SaltWaterSalinity'
SALTWATERTEMPERATURE = 'SaltWaterTemperature'
ISTEMPLATE = 'IsTemplate'
ISBUILTINTEMPLATE = 'IsBuiltinTemplate'
COMPOSITION = 'Composition'

class Parameters.Compressor
    Bases:
        sixgill.definitions.SurfaceComponent, sixgill.definitions.SharedPumpParameters,
        sixgill.definitions.SharedPumpCatalogParameters

    ROUTE = 'Route'
    HONOURSTONEWALLLIMIT = 'HonourStonewallLimit'
    ISRECIPROCATING = 'IsReciprocating'

class Parameters.IPRDarcy
    Bases: object

    RATIOType = 'RatioType'
    DAMAGEDZONEPERMRATIO = 'DamagedZonePermRatio'
    COMPACTEDZONEPERMRATIO = 'CompactedZonePermRatio'
    COMPLETIONVERTICALPERMRATIO = 'CompletionVerticalPermRatio'
```

COMPLETIONINTERVALRATIO = 'CompletionIntervalRatio'
DAMAGEDZONESKIN = 'DamagedZoneSkin'
PERFORATIONSKIN = 'PerforationSkin'
COMPACTEDZONESKIN = 'CompactedZoneSkin'
PARTIALPENETRATIONSkin = 'PartialPenetrationSkin'
DEVIATIONSKIN = 'DeviationSkin'
GRAVELPACKSKIN = 'GravelPackSkin'
FRACPACKSKIN = 'FracPackSkin'
CALCULATEMECHANICALSKIN = 'CalculateMechanicalSkin'
CALCULATERATEDEPENDENTSKIN = 'CalculateRateDependentSkin'
CHOKELength = 'ChokeLength'
OBSOLETECOMPACTEDZONEDIAMETER = 'ObsoleteCompactedZoneDiameter'
COMPACTEDZONETHICKNESS = 'CompactedZoneThickness'
COMPACTEDZONEPERMEABILITY = 'CompactedZonePermeability'
COMPLETIONDEVIATION = 'CompletionDeviation'
COMPLETIONVERTICALPERMEABILITY = 'CompletionVerticalPermeability'
COMPLETIONINTERVAL = 'CompletionInterval'
DAMAGEDEPTH = 'DamageDepth'
OBSOLETEDAMAGEDZONEDIAMETER = 'ObsoleteDamagedZoneDiameter'
DAMAGEDZONETHICKNESS = 'DamagedZoneThickness'
DAMAGEDZONEPERMEABILITY = 'DamagedZonePermeability'
DRAINAGERADIUS = 'DrainageRadius'
FRACTURECHOKEPERMEABILITY = 'FractureChokePermeability'
FRACTUREFACEDAMAGEPERMEABILITY = 'FractureFaceDamagePermeability'
FRACTUREHALFLENGTH = 'FractureHalfLength'
FRACTUREPROPPANTPERMEABILITY = 'FractureProppantPermeability'
FRACTUREWIDTH = 'FractureWidth'
ISGASMODEL = 'IsGasModel'
MECHANICALSKIN = 'MechanicalSkin'
PERFSKINMETHOD = 'PerfSkinMethod'
RATEDEPENDENTGASSKIN = 'RateDependentGasSkin'
RATEDEPENDENTLIQUIDSKIN = 'RateDependentLiquidSkin'
RESERVOIRAREA = 'ReservoirArea'
RESERVOIRPERMEABILITY = 'ReservoirPermeability'
RESERVOIRTHICKNESS = 'ReservoirThickness'
SHAPEFACTOR = 'ShapeFactor'

```
USERELATIVEPERMEABILITY = 'UseRelativePermeability'
USECHOKEFRACTURESkin = 'UseChokeFractureSkin'
USEDAMAGEDZONESkin = 'UseDamagedZoneSkin'
USEDRAINAGERADIUS = 'UseDrainageRadius'
USEFRACTUREFACESkin = 'UseFractureFaceSkin'
USEFRACTURESkin = 'UseFractureSkin'
USEGRAVELPACKSkin = 'UseGravelPackSkin'
USEPARTIALPENETRATIONDEVIATIONSkin = 'UsePartialPenetrationDeviationSkin'
USEPERFORATIONSkin = 'UsePerforationSkin'
USEPSEUDOPRESSUREMETHOD = 'UsePseudoPressureMethod'
USEVOGELBELOWBUBBLEPOINT = 'UseVogelBelowBubblepoint'
WELLCOMPLETIONTYPE = 'WellCompletionType'
ISTRANSIENT = 'IsTransient'
COMPRESSIBILITY = 'Compressibility'
POROSITY = 'Porosity'
TIME = 'Time'
WATERSATURATION = 'WaterSaturation'
RELATIVEPERMEABILITYOIL = 'RelativePermeabilityOil'
RELATIVEPERMEABILITYWATER = 'RelativePermeabilityWater'

class Parameters.DistributedCompletionModel
    Bases: object

    RATIOType = 'RatioType'

    DAMAGEDZONEPERMRATIO = 'DamagedZonePermRatio'
    COMPACTEDZONEPERMRATIO = 'CompactedZonePermRatio'
    COMPLETIONVERTICALPERMRATIO = 'CompletionVerticalPermRatio'
    DAMAGEDZONESkin = 'DamagedZoneSkin'
    PERFORATIONSkin = 'PerforationSkin'
    COMPACTEDZONESkin = 'CompactedZoneSkin'
    PARTIALPENETRATIONSkin = 'PartialPenetrationSkin'
    DEVIATIONSkin = 'DeviationSkin'
    GRAVELPACKSkin = 'GravelPackSkin'
    FRACPACKSkin = 'FracPackSkin'
    CALCULATESkin = 'CalculateSkin'
    OBSOLETECOMPACTEDZONEDIAMETER = 'ObsoleteCompactedZoneDiameter'
    COMPACTEDZONETHICKNESS = 'CompactedZoneThickness'
    COMPACTEDZONEPERMEABILITY = 'CompactedZonePermeability'
```

```
COMPLETIONLENGTH = 'CompletionLength'
COMPLETIONMODELTYPE = 'CompletionModelType'
OBSOLETEDEDAMAGEDZONEDIAMETER = 'ObsoleteDamagedZoneDiameter'
DAMAGEDZONETHICKNESS = 'DamagedZoneThickness'
DAMAGEDZONEPERMEABILITY = 'DamagedZonePermeability'
FLUIDOFVF = 'FluidOFVF'
FLUIDVISCOSITY = 'FluidViscosity'
GASPI = 'GasPI'
GASZ = 'GasZ'
GRAVELPACKPERMEABILITY = 'GravelPackPermeability'
GRAVELPACKTUNNEL = 'GravelPackTunnel'
ISGASMODEL = 'IsGasModel'
LIQUIDPI = 'LiquidPI'
LOCATIONECCEN = 'LocationEccen'
LOCATIONXPOS = 'LocationXPos'
LOCATIONYPOS = 'LocationYPos'
LOCATIONZPOS = 'LocationZPos'
PERFORATIONDIAMETER = 'PerforationDiameter'
PERFORATIONLENGTH = 'PerforationLength'
PERFORATIONSHOTDENSITY = 'PerforationShotDensity'
PERMEABILITYX = 'PermeabilityX'
PERMEABILITYY = 'PermeabilityY'
PERMEABILITYZ = 'PermeabilityZ'
RESERVOIREXTENT = 'ReservoirExtent'
RESERVOIRTHICKNESS = 'ReservoirThickness'
RESERVOIRXDIM = 'ReservoirXDim'
RESERVOIRYDIM = 'ReservoirYDim'
SPECIFIEDSKIN = 'SpecifiedSkin'
USEDISTRIBUTEDPI = 'UseDistributedPI'
WELLLENGTH = 'WellLength'
WELLCOMPLETIONTYPE = 'WellCompletionType'

class Parameters.IPRSSBabuOdeh
    Bases: sixgill.definitions.DistributedCompletionModel

class Parameters.IPRSSJoshi
    Bases: sixgill.definitions.DistributedCompletionModel

class Parameters.IPRHorizontalPI
    Bases: sixgill.definitions.DistributedCompletionModel
```

```
class Parameters.EngineKeywords
    Bases: sixgill.definitions.DownholeOrSurfaceComponent

    KEYWORDS = 'Keywords'

class Parameters.ESP
    Bases: sixgill.definitions.DownholeComponent

    SERIES = 'Series'

    USESTAGEBYSTAGECALC = 'UseStageByStageCalc'

    BASEFREQUENCY = 'BaseFrequency'

    CASINGID = 'CasingID'

    OPERATINGPRODUCTIONRATE = 'OperatingProductionRate'

    OPERATINGFREQUENCY = 'OperatingFrequency'

    DIAMETER = 'Diameter'

    GASSEPARATOREFFICIENCY = 'GasSeparatorEfficiency'

    HASGASSEPARATOR = 'HasGasSeparator'

    HEADFACTOR = 'HeadFactor'

    MANUFACTURER = 'Manufacturer'

    MAXFLOWRATE = 'MaxFlowRate'

    MINFLOWRATE = 'MinFlowRate'

    MODEL = 'Model'

    NUMBERSTAGES = 'NumberStages'

    USEVISCOSITYCORRECTION = 'UseViscosityCorrection'

    POWER = 'Power'

    USEPOWER = 'UsePower'

    ESPSPEEDFACTOR = 'ESPSpeedFactor'

    SLIPFACTOR = 'SlipFactor'

    FLOWRATEFACTOR = 'FlowrateFactor'

    POWERFACTOR = 'PowerFactor'

class Parameters.Expander
    Bases:

        sixgill.definitions.SurfaceComponent, sixgill.definitions.SharedPumpParameters

    ROUTE = 'Route'

class Parameters.FetkovitchEquationModel
    Bases: object

    ABSOLUTEOPENFLOWPOTENTIAL = 'AbsoluteOpenFlowPotential'

    NEXPONENT = 'NExponent'

class Parameters.Flowline
    Bases: sixgill.definitions.SurfaceComponent
```

```
SHOWASRISER = 'ShowAsRiser'
DETAILEDMODEL = 'DetailedModel'
USEGISDATA = 'UseGISData'
UNDULATIONRATE = 'UndulationRate'
ENTERSURVEYLENGTH = 'EnterSurveyLength'
MEASUREDDISTANCE = 'MeasuredDistance'
HORIZONTALDISTANCE = 'HorizontalDistance'
ELEVATIONDIFFERENCE = 'ElevationDifference'
SEABEDDEPTH = 'SeabedDepth'
PLATFORMHEIGHT = 'PlatformHeight'
AMBIENTTEMPERATURE = 'AmbientTemperature'
AMBIENTAIRTEMPERATURE = 'AmbientAirTemperature'
AMBIENTWATERTEMPERATURE = 'AmbientWaterTemperature'
AMBIENTTEMPERATUREOPTIONS = 'AmbientTemperatureOptions'
AMBIENTFLUIDTYPE = 'AmbientFluidType'
WATERINTERPOLATIONMETHOD = 'WaterInterpolationMethod'
DEVIATIONSURVEYCALC = 'DeviationSurveyCalc'
ISFLIPPED = 'IsFlipped'
GEOTHERMALSURVEYINDEX = 'GeothermalSurveyIndex'
USEGLOBALSETTINGS = 'UseGlobalSettings'
ISDOWNCOMER = 'IsDowncomer'
USEENVIRONMENTALDATA = 'UseEnvironmentalData'
SURFACETEMPERATURE = 'SurfaceTemperature'
DEPTHATSTART = 'DepthAtStart'
USEDEPTH = 'UseDepth'
INNERDIAMETER = 'InnerDiameter'
LENGTH = 'Length'
ROUGHNESS = 'Roughness'
WALLTHICKNESS = 'WallThickness'
CATALOGNAME = 'CatalogName'
CATALOGLABEL = 'CatalogLabel'
ENTEROD = 'EnterOD'
GEOMETRYPROFILE = 'GeometryProfile'
GEOTHERMALPROFILE = 'FlowlineGeothermalProfile'
class HeatTransfer
    Bases: object
```



```
GROUNDCONDUCTIVITY = 'GroundConductivity'
INCLUDEINSIDEFILMCOEFF = 'IncludeInsideFilmCoeff'
ISUCALCULATED = 'IsUCalculated'
PIPEBURIALDEPTH = 'PipeBurialDepth'
PIPECONDUCTIVITY = 'PipeConductivity'
SURROUNDINGFLUIDVELOCITY = 'SurroundingFluidVelocity'
USETHERMALSURVEY = 'UseThermalSurvey'
UCOEFFUSER = 'UCoeffUser'
UCOEFFUSERAIR = 'UCoeffUserAir'
UCOEFFUSERWATER = 'UCoeffUserWater'
UTYPE = 'UType'
UTYPEAIR = 'UTypeAir'
UTYPEWATER = 'UTypeWater'

class Parameters.FluidComponent
    Bases: sixgill.definitions.ModelComponent
    MOLECULARWEIGHT = 'MolecularWeight'
    TCRIT = 'Tcrit'
    TBOIL = 'Tboil'
    TMELT = 'Tmelt'
    PCRIT = 'Pcrit'
    VCRIT = 'Vcrit'
    ZCRIT = 'Zcrit'
    VCRITVISCOSITY = 'VcritViscosity'
    ACF = 'Acf'
    PARACHOR = 'Parachor'
    OMEGAA = 'OmegaA'
    OMEGAB = 'OmegaB'
    REFERENCEDENSITY = 'ReferenceDensity'
    EOSVOLUMESHIFT = 'EoSVolumeShift'
    SPECIFICGRAVITY = 'SpecificGravity'
    WATSONKFACTOR = 'WatsonKFactor'
    HEATVAPOURISATION = 'HeatVapourisation'
    CALORIFICVALUE = 'CalorificValue'
    THERMEXPCOEFF = 'ThermExpCoeff'
    HYDROCARBONFORM = 'HydrocarbonForm'
    ISUSERDEFINED = 'IsUserDefined'
```

```
THERMALCOEFFICIENT0 = 'ThermalCoefficient0'
THERMALCOEFFICIENT1 = 'ThermalCoefficient1'
THERMALCOEFFICIENT2 = 'ThermalCoefficient2'
THERMALCOEFFICIENT3 = 'ThermalCoefficient3'
THERMALCOEFFICIENT4 = 'ThermalCoefficient4'
THERMALCOEFFICIENT5 = 'ThermalCoefficient5'
THERMALCOEFFICIENT6 = 'ThermalCoefficient6'

class Parameters.FluidComponentFraction
    Bases: object

    SPECIFIEDFRACTION = 'SpecifiedFraction'
    TUNEDFRACTION = 'TunedFraction'
    COMPONENT = 'Component'

class Parameters.IPRForchheimer
    Bases: object

    COEFFICIENTA = 'CoefficientA'
    COEFFICIENTF = 'CoefficientF'

class Parameters.GasLiftInjection
    Bases: sixgill.definitions.DownholeComponent

    GASLIFTTYPE = 'GasLiftType'
    GASRATE = 'GasRate'
    GLR = 'GLR'
    GLRINCREASE = 'GLRIncrease'
    VALVEDIAMETER = 'ValveDiameter'
    MANUFACTURER = 'Manufacturer'
    SERIES = 'Series'
    VALVETYPE = 'ValveType'
    VALVESIZE = 'ValveSize'
    PORTSIZE = 'PortSize'
    PORTAREA = 'PortArea'
    BELLOWAREA = 'BellowArea'
    DISCHARGECEFFICIENT = 'DischargeCoefficient'
    DISCHARGETOFULLYOPEN = 'DischargeToFullyOpen'
    PTRO = 'Ptr0'
    VALVECHOKE = 'ValveChoke'
    SPRINGPRESSURE = 'SpringPressure'

class Parameters.GenericEquipment
    Bases: sixgill.definitions.SurfaceComponent
```

```
DISCHARGEPRESSURE = 'DischargePressure'
DISCHARGETEMPERATURE = 'DischargeTemperature'
DUTY = 'Duty'
PRESSUREDROP = 'PressureDrop'
PRESSURERATIO = 'PressureRatio'
PRESSURESPEC = 'PressureSpec'
ROUTE = 'Route'
TEMPERATUREDIFFERENTIAL = 'TemperatureDifferential'
TEMPERATURESPEC = 'TemperatureSpec'

class Parameters.Pump
    Bases:
        sixgill.definitions.SurfaceComponent, sixgill.definitions.SharedPumpParameters,
        sixgill.definitions.SharedPumpCatalogParameters

    ROUTE = 'Route'
    DESIGNPRODUCTIONRATE = 'DesignProductionRate'
    MAXFLOWRATE = 'MaxFlowRate'
    MINFLOWRATE = 'MinFlowRate'
    NUMBERSTAGES = 'NumberStages'
    USEVISCOSITYCORRECTION = 'UseViscosityCorrection'
    BASESTAGES = 'BaseStages'

class Parameters.GeographicSurvey
    Bases: object

    LATITUDE = 'Latitude'
    LONGITUDE = 'Longitude'
    ISVERTEX = 'IsVertex'

class Parameters.GravelPack
    Bases: object

    GRAVELCASINGID = 'GravelCasingId'
    GRAVELPERMEABILITY = 'GravelPermeability'
    GRAVELSCREENSIZE = 'GravelScreenSize'
    GRAVELTUNNELLENGTH = 'GravelTunnelLength'

class Parameters.HeatExchanger
    Bases: object

    DISCHARGEPRESSURE = 'DischargePressure'
    DISCHARGETEMPERATURE = 'DischargeTemperature'
    DUTY = 'Duty'
    PRESSUREDROP = 'PressureDrop'
    PRESSURESPEC = 'PressureSpec'
```

```
TEMPERATUREDIFFERENTIAL = 'TemperatureDifferential'
TEMPERATURESPEC = 'TemperatureSpec'

class Parameters.Injector
    Bases:
        sixgill.definitions.SurfaceComponent, sixgill.definitions.AssociatedToFluid

    GASFLOWRATE = 'GasFlowRate'
    INJECTEDFLUID = 'InjectedFluid'
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    MASSFLOWRATE = 'MassFlowRate'
    TEMPERATURE = 'Temperature'
    FLOWRATETYPE = 'FlowrateType'
    USEGASRATIO = 'UseGasRatio'
    USEWATERRATIO = 'UseWaterRatio'
    GOR = 'GOR'
    OGR = 'OGR'
    GLR = 'GLR'
    LGR = 'LGR'
    GWR = 'GWR'
    WGR = 'WGR'
    WATERCUT = 'WaterCut'
    USEFLUIDOVERRIDES = 'UseFluidOverrides'
    OVERRIDESINITIALIZED = 'OverridesInitialized'

class Parameters.IPRBackPressure
    Bases: object
    CONSTANTC = 'ConstantC'
    SLOPEN = 'SlopeN'

class Parameters.IPRHydraulicFracture
    Bases: object
    FRACTUREHALFLENGTH = 'FractureHalfLength'
    FRACTUREPERMEABILITY = 'FracturePermeability'
    FRACTUREWIDTH = 'FractureWidth'
    ISGASMODEL = 'IsGasModel'
    POROSITY = 'Porosity'
    RESERVOIRPERMEABILITY = 'ReservoirPermeability'
    RESERVOIRRADIUS = 'ReservoirRadius'
    RESERVOIRTHICKNESS = 'ReservoirThickness'
```

```
TIME = 'Time'
TOTALCOMPRESSIBILITY = 'TotalCompressibility'
USETRANSIENTMODEL = 'UseTransientModel'
USEVOGELBELOWBUBBLEPOINT = 'UseVogelBelowBubblepoint'

class Parameters.IPRPIModel
    Bases: object
    GASPI = 'GasPI'
    ISGASMODEL = 'IsGasModel'
    LIQUIDPI = 'LiquidPI'
    USEVOGELBELOWBUBBLEPOINT = 'UseVogelBelowBubblepoint'

class Parameters.IPRJones
    Bases: object
    GASCOEFFICIENTA = 'GasCoefficientA'
    GASCOEFFICIENTB = 'GasCoefficientB'
    ISGASMODEL = 'IsGasModel'
    LIQUIDCOEFFICIENTA = 'LiquidCoefficientA'
    LIQUIDCOEFFICIENTB = 'LiquidCoefficientB'

class Parameters.Source
    Bases:
        sixgill.definitions.SurfaceComponent, sixgill.definitions.AssociatedToFluid

    USEPQCURVE = 'UsePQCurve'
    TEMPERATURE = 'Temperature'
    SELECTEDRATETYPE = 'SelectedRateType'
    GASFLOWRATE = 'GasFlowRate'
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    MASSFLOWRATE = 'MassFlowRate'
    PRESSURE = 'Pressure'
    USEGASRATIO = 'UseGasRatio'
    USEWATERRATIO = 'UseWaterRatio'
    GOR = 'GOR'
    OGR = 'OGR'
    GLR = 'GLR'
    LGR = 'LGR'
    GWR = 'GWR'
    WGR = 'WGR'
    WATERCUT = 'WaterCut'
    USEFLUIDOVERRIDES = 'UseFluidOverrides'
```

```
    OVERRIDESINITIALIZED = 'OverridesInitialized'
    PQCURVE = 'PQCurve'

class Parameters.Junction
    Bases: sixgill.definitions.Source

    TREATASSOURCE = 'TreatAsSource'

class Parameters.Liner
    Bases: sixgill.definitions.AbstractTubingSection

class Parameters.MeasurementPoint
    Bases: object

    CONNECTEDTOSTARTEQUIPMENT = 'ConnectedToStartEquipment'

class Parameters.MFLFluid
    Bases: sixgill.definitions.FileBasedFluid

class Parameters.MultiphaseBooster
    Bases: sixgill.definitions.SurfaceComponent

    POWER = 'Power'

    DISCHARGEPRESSURE = 'Pressure'

    PRESSUREDIFFERENTIAL = 'PressureDifferential'

    PRESSURERATIO = 'PressureRatio'

    TYPE = 'BoosterType'

class Parameters.GenericMultiphaseBooster
    Bases: sixgill.definitions.MultiphaseBooster

    PUMPEFFICIENCY = 'Efficiency'

    COMPRESSOREFFICIENCY = 'BoosterEfficiency'

class Parameters.OneSubseaMultiphaseBooster
    Bases: sixgill.definitions.MultiphaseBooster

    PUMPMODEL = 'PumpModel'

    TUNINGFACTOR = 'TuningFactor'

    SPEEDLIMIT = 'SpeedLimit'

    NUMBEROFBOOSTERSINPARALLEL = 'NumberOfBoostersInParallel'

    RECIRCULATIONFLOWRATE = 'RecirculationFlowRate'

class Parameters.MultiplierAdder
    Bases: sixgill.definitions.SurfaceComponent

    ADDER = 'Adder'

    SCALER = 'Scaler'

    USEASMULTIPLIER = 'UseAsMultiplier'

    ADDERRATETYPE = 'AdderRateType'

    ADDEDLIQUID = 'AddedLiquid'

    ADDEDGAS = 'AddedGas'

    ADDEDMASS = 'AddedMass'
```

```
class Parameters.OpenHole
    Bases: sixgill.definitions.AbstractTubingSection

class Parameters.Packer
    Bases: sixgill.definitions.DownholeComponent

class Parameters.PCP
    Bases: sixgill.definitions.DownholeComponent

    BASESPEED = 'BaseSpeed'
    CASINGID = 'CasingID'
    OPERATINGPRODUCTIONRATE = 'OperatingProductionRate'
    OPERATINGSPEED = 'OperatingSpeed'
    DIAMETER = 'Diameter'
    GASSEPARATOREFFICIENCY = 'GasSeparatorEfficiency'
    HASGASSEPARATOR = 'HasGasSeparator'
    HEADFACTOR = 'HeadFactor'
    MANUFACTURER = 'Manufacturer'
    NOMINALFLOWRATE = 'NominalFlowRate'
    MODEL = 'Model'
    USEVISCOSITYCORRECTION = 'UseViscosityCorrection'
    POWER = 'Power'
    USEPOWER = 'UsePower'
    SPEEDFACTOR = 'SpeedFactor'
    SLIPFACTOR = 'SlipFactor'
    FLOWRATEFACTOR = 'FlowrateFactor'
    POWERFACTOR = 'PowerFactor'
    ISTOPDRIVE = 'IsTopDrive'
    RODDIAMETER = 'RodDiameter'

class Parameters.PVTFluid
    Bases: sixgill.definitions.FileBasedFluid

class Parameters.RodPump
    Bases: sixgill.definitions.DownholeComponent

    GASSEPARATOREFFICIENCY = 'GasSeparatorEfficiency'
    HASGASSEPARATOR = 'HasGasSeparator'
    HASGASRECOMBINEDATWELLHEAD = 'HasGasRecombinedAtWellHead'
    NOMINALFLOWRATE = 'NominalFlowRate'
    USEVISCOSITYCORRECTION = 'UseViscosityCorrection'
    MAXDP = 'MaxDP'
    MAXPOWER = 'MaxPower'
    RODDIAMETER = 'RodDiameter'
```

```
class Parameters.SinglePhaseSeparator
    Bases: sixgill.definitions.DownholeComponent
    EFFICIENCY = 'Efficiency'
    SEPARATEDFLUID = 'SeparatedFluid'

class Parameters.Sink
    Bases: sixgill.definitions.SurfaceComponent
    GASFLOWRATE = 'GasFlowRate'
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    MASSFLOWRATE = 'MassFlowRate'
    PRESSURE = 'Pressure'
    FLOWRATETYPE = 'FlowrateType'

class Parameters.Slidingsleeve
    Bases: sixgill.definitions.DownholeComponent
    ISOPEN = 'IsOpen'

class Parameters.SubsurfaceSafetyValve
    Bases: sixgill.definitions.DownholeComponent
    BEANID = 'BeanID'

class Parameters.Study
    Bases: object
    NAME = 'Name'
    DESCRIPTION = 'Description'

class Parameters.AbstractTask
    Bases: object
    TASKNAME = 'Name'
    DESCRIPTION = 'Description'
    STUDY = 'Study'
    USEPHASERATIO = 'UsePhaseRatio'
    FLOWRATECONDITION = 'FlowRateCondition'
    FLOWRATECONDITIONTEMPERATURE = 'FlowRateConditionTemperature'
    FLOWRATECONDITIONPRESSURE = 'FlowRateConditionPressure'

class Parameters.NetworkSimulation
    Bases: sixgill.definitions.AbstractTask
    USESURFACEBOUNDARYCONDITIONS = 'UseSurfaceBoundaryConditions'

class Parameters.PTPProfileSimulation
    Bases: sixgill.definitions.AbstractTask
    PRODUCER = 'Producer'
    BRANCHTERMINATOR = 'BranchTerminator'
    CALCULATEDVARIABLE = 'CalculationVariableType'
    CUSTOMVARIABLE = 'CalculationVariable'
```



```
FLOWRATETYPE = 'FlowRateType'
GASFLOWRATE = 'GasFlowRate'
LIQUIDFLOWRATE = 'LiquidFlowRate'
MASSFLOWRATE = 'MassFlowRate'
INLETPRESSURE = 'InletPressure'
OUTLETPRESSURE = 'OutletPressure'
SENSITIVITYVARIABLE = 'SensitivityVariable'

class CustomVariable
    Bases: object

    COMPONENT = 'Component'
    VARIABLE = 'Variable'
    MINVALUE = 'MinValue'
    MAXVALUE = 'MaxValue'
    ISDIRECTPROPORTIONALITY = 'IsDirectProportionality'

class Parameters.PTProfileSimulation.SensitivityVariable
    Bases: object

    COMPONENT = 'Component'
    VARIABLE = 'Variable'
    VALUES = 'Values'

class Parameters.ThreePhaseSeparator
    Bases: sixgill.definitions.SurfaceComponent

    GASOILEFFICIENCY = 'GasOilEfficiency'
    WATEROILEFFICIENCY = 'WaterOilEfficiency'
    PRESSURE = 'Pressure'
    PRODUCTFLUID = 'ProductFluid'

class Parameters.Tubing
    Bases: sixgill.definitions.AbstractTubingSection

class Parameters.TwoPhaseSeparator
    Bases: sixgill.definitions.SurfaceComponent

    EFFICIENCY = 'Efficiency'
    PRESSURE = 'Pressure'
    PRODUCTIONSTREAM = 'ProductionStream'
    DISCARDEDSTREAM = 'DiscardedStream'

class Parameters.IPRVogel
    Bases: object

    ABSOLUTEOPENFLOWPOTENTIAL = 'AbsoluteOpenFlowPotential'
    VOGELCOEFFICIENT = 'VogelCoefficient'
```

```
class Parameters.WaterTempVelocitySurvey
    Bases: object

    ISGLOBAL = 'IsGlobal'

class Parameters.Well
    Bases:

        sixgill.definitions.SurfaceComponent, sixgill.definitions.AssociatedToFluid,
        sixgill.definitions.CanBeTemplate

    GASFLOWRATE = 'GasFlowRate'
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    MASSFLOWRATE = 'MassFlowRate'
    FLOWRATETYPE = 'FlowRateType'
    ISINJECTION = 'IsInjection'
    CHECKVALVESETTING = 'CheckValveSetting'
    WELLHEADDEPTH = 'WellheadDepth'
    AMBIENTTEMPERATURE = 'AmbientTemperature'
    USEGASLIFTVALVE = 'UseGasliftValve'
    ALHANATICHECK = 'AlhanatiCheck'
    SURFACEGASTEMPERATURE = 'SurfaceGasTemperature'
    SURFACEGASPRESSURE = 'SurfaceGasPressure'
    TARGETINJECTIONRATE = 'TargetInjectionRate'
    MINVALVEINJECTIONDP = 'MinValveInjectionDP'
    GASLIFTINPUTOPTION = 'GasLiftInputOption'
    GASSPECIFICGRAVITY = 'GasSpecificGravity'
    TUNINGFACTOR = 'TuningFactor'
    TESTRACKTEMPERATURE = 'TestRackTemperature'
    NITROGENCORRECTIONTYPE = 'NitrogenCorrectionType'
    TRAJECTORY = 'Trajectory'
    GEOTHERMALPROFILE = 'BoreholeGeothermalProfile'

class HeatTransfer
    Bases: object

    UVALUEINPUTOPTION = 'HeatTransferMethod'
    UCOEFF = 'UCoeff'
    USEWELLHEADAMBIENTTEMPERATURE = 'UseWellHeadAmbientTemperature'
    ISGEOSURVEYDEPTHTVD = 'IsGeoSurveyDepthTVD'

class Parameters.Well.DeviationSurvey
    Bases: object

    SURVEYTYPE = 'SurveyType'
    RELATIVEDEPTHOPTION = 'DeviationRelativeDepth'
```

```
DEPENDENTPARAMETER = 'TrajectoryDependantParameter'
CALCULATEUSINGTANGENTIALMETHOD = 'CalculateUsingTangentialMethod'

class Parameters.FlowlineGeometry
    Bases: object

    MEASUREDDISTANCE = 'MeasuredDistance'
    HORIZONTALDISTANCE = 'HorizontalDistance'
    ELEVATION = 'Elevation'
    LATITUDE = 'Latitude'
    LONGITUDE = 'Longitude'
    ISVERTEX = 'IsVertex'

class Parameters.GeothermalSurvey
    Bases: object

    MEASUREDDISTANCE = 'MeasuredDistance'
    HORIZONTALDISTANCE = 'HorizontalDistance'
    TEMPERATURE = 'Temperature'
    CURRENTVELOCITY = 'CurrentVelocity'
    UCOEFF = 'UCoeff'

class Parameters.WellTrajectory
    Bases: object

    MEASUREDDEPTH = 'MeasuredDepth'
    TRUEVERTICALDEPTH = 'TrueVerticalDepth'
    INCLINATION = 'Inclination'
    AZIMUTH = 'Azimuth'
    MAXDOGLEGESEVERITY = 'MaxDogLegSeverity'

class Parameters.CompletionTestPoint
    Bases: object

    LIQUIDFLOWRATE = 'LiquidFlowRate'
    GASFLOWRATE = 'GasFlowRate'
    STATICRESERVOIRPRESSURE = 'StaticReservoirPressure'
    BOTTOMHOLEFLOWINGPRESSURE = 'BottomHoleFlowingPressure'

class Parameters.PQCurve
    Bases: object

    GASFLOWRATE = 'GasFlowRate'
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    MASSFLOWRATE = 'MassFlowRate'
    PRESSURE = 'Pressure'
```

```
class sixgill.definitions.Constants
    Bases: object

    Enumerated constants used by PIPESIM domain model

    class WellImportFluidConflict
        Bases: object

        CREATENEW = 'CreateNew'
        USEEXISTING = 'UseExisting'
        DONTIMPORT = 'DontImport'

    class Constants.SoilType
        Bases: object

        PEATDRY = 'PeatDry'
        PEATWET = 'PeatWet'
        PEATICY = 'PeatIcy'
        LOAM = 'Loam'
        SANDYDRY = 'SandyDry'
        SANDYMOIST = 'SandyMoist'
        SANDYSOAKED = 'SandySoaked'
        CLAYDRY = 'ClayDry'
        CLAYMOIST = 'ClayMoist'
        CLAYWET = 'ClayWet'
        CLAYFROZEN = 'ClayFrozen'
        GRAVEL = 'Gravel'
        GRAVELSANDY = 'GravelSandy'
        LIMESTONE = 'Limestone'
        SANDSTONE = 'Sandstone'
        ICE = 'Ice'
        ICEN40C = 'IceN40C'
        SNOWLOOSE = 'SnowLoose'
        SNOWHARD = 'SnowHard'
        USERDEFINED = 'UserDefined'

    class Constants.MetocceanDataLocation
        Bases: object

        GULFOFMEXICO = 'GOM'
        NORTHSEA = 'NSea'
        WESTAFRICA = 'WAfrica'
        WESTAUSTRALIA = 'WAustralia'
        OFFSHOREBRAZIL = 'OffshoreBrazil'
```

```
    USERDEFINED = 'UserDefined'

class Constants.NetworkSolverMethod
    Bases: object

    AUTOMATIC = 'Automatic'

    STANDARD = 'Standard'

    ADVANCED = 'Advanced'

class Constants.GisElevationDataSource
    Bases: object

    ASTER = 'astergdem'

    ESRI = 'Esri'

    SRTM = 'srtm3'

class Constants.OilFVFCorrelation
    Bases: object

    STANDING = 'Standing'

    VASQUEZANDBEGGS = 'VasquezAndBeggs'

    KARTOATMODJO = 'Kartoatmodjo'

class Constants.LiveOilViscCorrelation
    Bases: object

    BEGGSANDROBINSON = 'BeggsAndRobinson'

    CHEWANDCONNALY = 'ChewandConnaly'

    KARTOATMODJO = 'Kartoatmodjo'

    KHAN = 'Khan'

    DEGGETTO = 'DeGhetto'

    HOSSAIN = 'Hossain'

    ELSHARKAWY = 'Elsharkawy'

    PETROSKYFARSHAD = 'PetroskyFarshad'

class Constants.GasCompressCorrelation
    Bases: object

    STANDING = 'Standing'

    HALLANDYARBOROUGH = 'HallAndYarborough'

    ROBINSONETAL = 'Robinsonetal'

class Constants.GasViscCorrelation
    Bases: object

    LEEETAL = 'Leeetal'

class Constants.CharacterizationSystem
    Bases: object

    PVTI = 'PVTi'

    DBR = 'DBR'
```

```
MULTIFLASH = 'Multiflash'

class Constants.BICsCorrelation
    Bases: object
    PVTI = 'PVTi'
    OILANDGAS1 = 'OilAndGas1'
    OILANDGAS2 = 'OilAndGas2'
    OILANDGAS3 = 'OilAndGas3'
    OILANDGAS4 = 'OilAndGas4'

class Constants.IPRModels
    Bases: object
    IPRBACKPRESSURE = 'IPRBackPressure'
    IPRDARCY = 'IPRDarcy'
    IPRFETKOVITCH = 'IPRFetkovitch'
    IPRFORCHHEIMER = 'IPRForchheimer'
    IPRHORIZONTALPI = 'IPRHorizontalPI'
    IPRHYDRAULICFRACTURE = 'IPRHydraulicFracture'
    IPRJONES = 'IPRJones'
    IPRPIMODEL = 'IPRPIModel'
    IPRSSBABUODEH = 'IPRPSSBabuOdeh'
    IPRSSJOSHI = 'IPRSSJoshi'
    IPRVOGEL = 'IPRVogel'

class Constants.ErosionModels
    Bases: object
    API14E = 'API14e'

class Constants.CorrosionModels
    Bases: object
    DEWAARD1995 = 'deWaard1995'
    NONE = 'None'

class Constants.CompositionalFluidFlash
    Bases: object
    class PhysicalPropertiesMethod
        Bases: object
        INTERPOLATE = 'Interpolate'
        HYBRID = 'Hybrid'
        RIGOROUS = 'Rigorous'
    class Constants.CompositionalFluidFlash.TemperatureEnthalpyMethod
        Bases: object
        INTERPOLATE = 'Interpolate'
```

```
HYBRID = 'Hybrid'
RIGOROUS = 'Rigorous'

class Constants.CompositionalFluidFlash.SingleComponentSystem
    Bases: object
    YES = 'Yes'
    NO = 'No'
    AUTOMATIC = 'Automatic'

class Constants.MultiphaseFlowCorrelationSource
    Bases: object
    BAKER_JARDINE = 'BJA'
    TULSA = 'Tulsa'
    OLGAS = 'OLGAS'
    TUFFPUNIFIED = 'TUFFP Unified'
    LEDAFLOWPM = 'LedaFlow PM'
    NEOTEC = 'Neotec'

class Constants.MultiphaseFlowCorrelation
    Bases: object
    class BakerJardine
        Bases: object
        ANSARI = 'Ansari'
        BEGGSBRILLORIGINAL = 'Beggs & Brill Original'
        BEGGSBRILLTAITELDUKLERMAP = 'Beggs & Brill Taitel Dukler map'
        BEGGSBRILLREVISED = 'Beggs & Brill Revised'
        BEGGSBRILLREVISEDTAITELDUKLERMAP = 'Beggs & Brill Revised Taitel Dukler map'
        BAKERJARDINEREVISED = 'Baker Jardine Revised'
        DUKLERAGAFLANIGAN = 'Dukler AGA & Flanigan'
        DUKLERAGAFLANIGAN_EATONHOLDUP = 'Dukler AGA & Flanigan (Eaton Holdup)'
        MUKHERJEEBRILL = 'Mukherjee & Brill'
        NOSLIPASSUMPTION = 'No Slip Assumption'
        OLIEMANS = 'Oliemans'
        XIAO = 'Xiao'
        BEGGSBRILL = 'Beggs & Brill'
        DUKLER = 'Dukler'
        DUNSROS = 'Duns & Ros'
        GOVIERAZIZFOGARASI = 'Govier Aziz & Fogarasi'
        GRAY_MODIFIED = 'Gray (modified)'
        GRAY_ORIGINAL = 'Gray (original)'
```

```
HAGEDORNBROWN = 'Hagedorn & Brown'
HAGEDORNBROWNDUNSROSMAP = 'Hagedorn & Brown Duns & Ros map'
ORKISZEWSKI = 'Orkiszewski'

class Constants.MultiphaseFlowCorrelation.OLGAS
    Bases: object

    OLGASV731_3PHASE = 'OLGAS v. 7.3.1 3-Phase'
    OLGASV731_2PHASE = 'OLGAS v. 7.3.1 2-Phase'
    OLGASV72_3PHASE = 'OLGAS v. 7.2 3-Phase'
    OLGASV72_2PHASE = 'OLGAS v. 7.2 2-Phase'
    OLGASV627_3PHASE = 'OLGAS v. 6.2.7 3-Phase'
    OLGASV627_2PHASE = 'OLGAS v. 6.2.7 2-Phase'

class Constants.MultiphaseFlowCorrelation.TUFFPUnified
    Bases: object

    TUFFPV20111_3PHASE_DEFAULT = 'TUFFP v. 2011.1 3-Phase (default)'
    TUFFPV20111_3PHASE_EMULSIONOVERRIDE = 'TUFFP v. 2011.1 3-Phase (emulsion override)'
    TUFFPV20111_2PHASE = 'TUFFP v. 2011.1 2-Phase'

class Constants.MultiphaseFlowCorrelation.LedaFlowPM
    Bases: object

    LEDAFLOWV14_3PHASE = 'LedaFlow v. 1.4 3-Phase'
    LEDAFLOWV14_2PHASE = 'LedaFlow v. 1.4 2-Phase'
    LEDAFLOWV12_3PHASE = 'LedaFlow v. 1.2 3-Phase'
    LEDAFLOWV12_2PHASE = 'LedaFlow v. 1.2 2-Phase'

class Constants.MultiphaseFlowCorrelation.Neotec
    Bases: object

    GREGORY = 'Gregory'
    AZIZGOVIERFOGARASI = 'Aziz Govier Fogarasi'
    EATONOLIEMANS = 'Eaton Oliemans'
    HUGHMARKDUKLER = 'Hughmark Dukler'
    XIAOMODFILM = 'Xiao Mod. Film'
    GOMEZ = 'Gomez'
    GOMEZENHANCED = 'Gomez Enhanced'

class Constants.MultiphaseFlowCorrelation.TulsaLegacy
    Bases: object

    BEGGSBRILL = 'Beggs & Brill'
    DUNSROS = 'Duns & Ros'
    GOVIERAZIZ = 'Govier Aziz'
    HAGEDORNBROWN_REVISIED = 'Hagedorn & Brown (Revised)'
    HAGEDORNBROWN_ORIGINAL = 'Hagedorn & Brown (Original)'
```



```
MUKHERJEEBRILL = 'Mukherjee & Brill'
ORKISZEWSKI = 'Orkiszewski'

class Constants.NodalAnalysisLimits
    Bases: object

    LIQUIDFLOWRATE = 'LiquidFlowRate'
    GASFLOWRATE = 'GasFlowRate'
    MASSFLOWRATE = 'MassFlowRate'

class Constants.FlowRateCondition
    Bases: object

    STOCKTANK = 'StockTank'
    INSITU = 'Flowing'

class Constants.TubingPosition
    Bases: object

    POSITIONED = 'Positioned'
    CENTRALIZED = 'Centralized'
    FLUSHJOINT = 'FlushJoint'
    NOTPOSITIONED = 'NotPositioned'

class Constants.PerforationPosition
    Bases: object

    POSITIONED = 'Positioned'
    CENTERED = 'Centered'
    ECCENTERED = 'Eccentered'

class Constants.PerforationGunPhaseAngle
    Bases: object

    ZERO = 'zero'
    P45 = 'p45'
    P60 = 'p60'
    P72 = 'p72'
    P90 = 'p90'
    P99 = 'p99'
    P120 = 'p120'
    P120_P60 = 'p120_p60'
    P135_P45 = 'p135_p45'
    P140_P20 = 'p140_p20'
    P180 = 'p180'
    PM10 = 'pm10'
    PM20 = 'pm20'
    PM45 = 'pm45'
```

```
ZERO_PM45 = 'zero_pm45'
PM60 = 'pm60'
ZERO_PM60 = 'zero_pm60'
PM90 = 'pm90'
P45_PNDLM = 'p45_pndlm'
ZERO_PM35 = 'zero_pm35'
MOEB45 = 'moeb45'
P120_P40 = 'p120_p40'
class Constants.PerforationGunHardware
    Bases: object
    STANDARD = 'Standard'
    OBSOLETE = 'Obsolete'
    NONSTANDARD = 'NonStandard'
    PURE = 'PURE'
    STDPURE = 'STDPURE'
    OBSPURE = 'ObsPURE'
    NONSTANDARDPURE = 'NonStandardPURE'
    MYSTERY = 'Mystery'
    UNKNOWN = 'Unknown'
class Constants.PenetrationModelOptions
    Bases: object
    ROCKONLY = 'RockOnly'
    CONCRETEONLY = 'ConcreteOnly'
    ROCKORCONCRETE = 'RockOrConcrete'
class Constants.PenetrationModelType
    Bases: object
    ROCK = 'Rock'
    CONCRETE = 'Concrete'
class Constants.PerforationGunAPITestEdition
    Bases: object
    ESTIMATED_API = 'Estimated_API'
    UNOFFICIAL_API = 'Unofficial_API'
    UNOFFICIAL_C_33M = 'Unofficial_C_33M'
    UNOFFICIAL_19B_1ST_ED = 'Unofficial_19B_1st_Ed'
    RP43_4THED = 'RP43_4thEd'
    RP43_5THED = 'RP43_5thEd'
    RP43_C_33M = 'RP43_C_33M'
```

```
A19B_1STED = 'A19B_1stEd'
CUSTOM_VALUES = 'Custom_Values'
BASED_CUSTOM_VALUES = 'Based_Custom_Values'
BASED_A19B_1STED = 'Based_A19B_1stEd'
BASED_RP43_C_33M = 'Based_RP43_C_33M'
BASED_RP43_5THED = 'Based_RP43_5thEd'
BASED_RP43_4THED = 'Based_RP43_4thEd'
BASED_UNOFFICIAL_19B_1ST_ED = 'Based_Unofficial_19B_1st_Ed'
BASED_UNOFFICIAL_C_33M = 'Based_Unofficial_C_33M'
BASED_UNOFFICIAL_API = 'Based_Unofficial_API'

class Constants.DeviationSurveyType
    Bases: object

    VERTICAL = 'VerticalDeviation'
    TWODIMENSIONAL = 'TwoDimensional'
    THREEDIMENSIONAL = 'ThreeDimensional'

class Constants.ReferenceDepthOption
    Bases: object

    OriginalRKB = 'OriginalRKB'
    RKB = 'RKB'
    GL = 'GL'
    MSL = 'MSL'
    THF = 'THF'

class Constants.UValueInputOption
    Bases: object

    InputSingleUValue = 'InputSingleUValue'
    InputMultipleUValues = 'InputMultipleUValues'

class Constants.PipeHeatTransfer
    Bases: object

    INSULATED = 'Insulated'
    COATED = 'Coated'
    BAREINAIR = 'BareInAir'
    BAREINWATER = 'BareInWater'
    USERSUPPLIED = 'UserSupplied'

class Constants.PipeBurialMethod
    Bases: object

    METHOD2009 = 'Method2009'
    METHOD2000 = 'Method2000'
    METHOD1983 = 'Method1983'
```

```
class Constants.InsideFilmCoeffMethod
    Bases: object

    KREITH = 'Kreith'
    KAMINSKY = 'Kaminsky'

class Constants.FlowRateType
    Bases: object

    LIQUIDFLOWRATE = 'LiquidFlowRate'
    GASFLOWRATE = 'GasFlowRate'
    MASSFLOWRATE = 'MassFlowRate'

class Constants.RangeType
    Bases: object

    RANGEINCREMENT = 'rangeIncrement'
    RANGEDECREMENT = 'rangeDecrement'
    RANGESTEP = 'rangeStep'
    RANGEMULTIPLY = 'rangeMultiply'

class Constants.SinglePhaseFlowCorrelation
    Bases: object

    MOODY = 'Moody'
    AGA = 'AGA'
    PANHANDLEA = "Panhandle 'A'"
    PANHANDLEB = "Panhandle 'B'"
    HAZEN_WILLIAMS = 'Hazen - Williams'
    WEYMOUTH = 'Weymouth'
    CULLENDER_SMITH = 'Cullender - Smith'

class Constants.SensitivityType
    Bases: object

    UNDEFINED = 'Undefined'
    FLUID = 'Fluid'
    SYSTEMDATA = 'SystemData'
    GASLIFTDATA = 'GasliftData'

class Constants.SystemVarType
    Bases: object

    UNDEFINED = 'varUndefined'
    INLETPRESSURE = 'varInletPressure'
    OUTLETPRESSURE = 'varOutletPressure'
    LIQFLOWRATE = 'varLiqFlowrate'
    GASFLOWRATE = 'varGasFlowrate'
    MASSFLOWRATE = 'varMassFlowrate'
```

```
CUSTOM = 'varCustom'

class Constants.DeviationRelativeDepth
    Bases: object

    ORIGINALRKB = 'OriginalRKB'

    RKB = 'RKB'

    GL = 'GL'

    MSL = 'MSL'

    THF = 'THF'

class Constants.TrajectoryDependentParameter
    Bases: object

    MD = 'MD'

    TVD = 'TVD'

    ANGLE = 'Angle'

class Constants.SurveyType
    Bases: object

    VERTICALDEVIATION = 'VerticalDeviation'

    TWODIMENSIONAL = 'TwoDimensional'

    THREEDIMENSIONAL = 'ThreeDimensional'

class Constants.ZoneMaterial
    Bases: object

    UNKNOWN = 'unknown'

    SHALE = 'shale'

    SAND = 'sand'

    WATER = 'water'

class Constants.GISWMSLayerPropertyFormat
    Bases: object

    PNG = 'PNG'

    JPEG = 'JPEG'

class Constants.ElevationSource
    Bases: object

    SRTM3 = 'srtm3'

    ASTERGDEM = 'astergdem'

    ESRI = 'Esri'

class Constants.GasRatioOption
    Bases: object

    GLR = 'GLR'

    GOR = 'GOR'

    LGR = 'LGR'
```

```
OGR = 'OGR'

class Constants.WaterRatioOption
    Bases: object

    WATERCUT = 'WaterCut'

    GWR = 'GWR'

    WGR = 'WGR'

class Constants.DeadOilViscosityCorrelation
    Bases: object

    BEGGSANDROBINSON = 'BeggsAndRobinson'

    GLASO = 'Glaso'

    KARTOATMODJO = 'Kartoatmodjo'

    DEGGETTO = 'DeGhetto'

    HOSSAIN = 'Hossain'

    ELSHARKAWY = 'Elsharkawy'

    PETROSKYFARSHAD = 'PetroskyFarshad'

    USER2POINT = 'User2Point'

    USERTABLE = 'UserTable'

class Constants.LiveOilViscosityCorrelation
    Bases: object

    BEGGSANDROBINSON = 'BeggsAndRobinson'

    CHEWANDCONNALY = 'ChewandConnaly'

    KARTOATMODJO = 'Kartoatmodjo'

    KHAN = 'Khan'

    DEGGETTO = 'DeGhetto'

    HOSSAIN = 'Hossain'

    ELSHARKAWY = 'Elsharkawy'

    PETROSKYFARSHAD = 'PetroskyFarshad'

class Constants.EnthalpyCalcMethod
    Bases: object

    METHOD1983 = 'Method1983'

    METHOD2009 = 'Method2009'

class Constants.UndersaturatedOilViscosityCorrelation
    Bases: object

    NONE = 'None'

    VASQUEZANDBEGGS = 'VasquezAndBeggs'

    KOUZEL = 'Kouzel'

    KARTOATMODJO = 'Kartoatmodjo'

    KHAN = 'Khan'
```

```
DEGHETTO = 'DeGhetto'
HOSSAIN = 'Hossain'
ELSHARKAWY = 'Elsharkawy'
BERGMANANDSUTTON = 'BergmanAndSutton'
PETROSKYFARSHAD = 'PetroskyFarshad'

class Constants.CriticalFlowCorrelation
    Bases: object
    MECHANISTIC = 'Mechanistic'
    GILBERT = 'Gilbert'
    ROS = 'Ros'
    ACHONG = 'Achong'
    BAXENDELL = 'Baxendell'
    ASHFORD = 'Ashford'
    POETBECK = 'Poetbeck'
    OMANA = 'Omana'
    PILEHVARI = 'Pilehvari'

class Constants.SubCriticalFlowCorrelation
    Bases: object
    MECHANISTIC = 'Mechanistic'
    API14B = 'API14b'
    ASHFORD = 'Ashford'

class Constants.ComponentFractionType
    Bases: object
    MOLE = 'Mole'
    MASS = 'Mass'

class Constants.LiquidViscosityCalculationMethod
    Bases: object
    CONTINUOUSPHASE = 'ContinuousPhase'
    OILWATERVOLUMERATIO = 'OilWaterVolumeRatio'
    ORIGINALWOELFLIN = 'OriginalWoelflin'
    WOELFLINLOOSE = 'WoelflinLoose'
    WOELFLINMEDIUM = 'WoelflinMedium'
    WOELFLINTIGHT = 'WoelflinTight'
    BRINKMAN = 'Brinkman'
    VANDVAND = 'VandVand'
    VANDBARNEAMIZRAHI = 'VandBarneaMizrahi'
    VANDUSER = 'VandUser'
```

```
RICHARDSON = 'Richardson'
LEVITONLEIGHTON = 'LevitonLeighton'
USERTABLE = 'UserTable'

class Constants.SaltWaterDensity
    Bases: object

    DEFAULTDENSITY = 'DefaultDensity'
    DENSITY = 'Density'
    SALINITY = 'Salinity'

class Constants.FluidFlashType
    Bases: object

    PRESSURETEMPERATURE = 'PressureTemperature'
    PRESSUREENTHALPYMASS = 'PressureEnthalpyMass'
    PRESSUREENTROPYMASS = 'PressureEntropyMass'
    PRESSUREENTHALPYMOLAR = 'PressureEnthalpyMolar'
    PRESSUREENTROPYMOLAR = 'PressureEntropyMolar'

class Constants.IPRDarcyRatioType
    Bases: object

    RATIO = 'Ratio'
    ABSOLUTE = 'Absolute'

class Constants.PerforationSkinMethod
    Bases: object

    MCLEOD = 'McLeod'
    KARAKASTARIQ = 'KarakasTariq'

class Constants.WellCompletionType
    Bases: object

    NONE = 'None'
    OPENHOLE = 'OpenHole'
    PERFORATED = 'Perforated'
    GRAVELPACKEDANDPERFORATED = 'GravelPackedAndPerforated'
    OPENHOLEGRAVELPACK = 'OpenHoleGravelPack'
    FRACPACK = 'FracPack'

class Constants.ChokeValveEquation
    Bases: object

    MECHANISTIC = 'Mechanistic'
    APIRP11V2 = 'APIRP11V2'

class Constants.AmbientTemperatureOption
    Bases: object

    INPUTSINGLEVALUE = 'InputSingleValue'
```



```
INPUTMULTIPLEVALUES = 'InputMultipleValues'
USEGLOBALVALUE = 'UseGlobalValue'

class Constants.AmbientFluid
    Bases: object

    AIR = 'Air'
    WATER = 'Water'

class Constants.WaterInterpolationMethod
    Bases: object

    INTERPOLATE = 'Interpolate'
    STEPFUNCTION = 'StepFunction'

class Constants.DeviationSurveyCalculationMethod
    Bases: object

    MEASUREDDISTANCE = 'measuredDistance'
    HORIZONTALDISTANCE = 'horizontalDistance'
    VERTICALDISTANCE = 'verticalDistance'

class Constants.CompletionTestType
    Bases: object

    MULTIPOINT = 'Multipoint'
    ISOCHRONAL = 'Isochronal'

class Constants.GasLiftType
    Bases: object

    INJECTIONGASRATE = 'InjectionGasRate'
    GLR = 'GLR'
    GLRINCREASE = 'GLRIncrease'

class Constants.GaugeType
    Bases: object

    UNDEFINED = 'gaugeUndefined'
    PRESSURE = 'gaugePressure'
    FLOWRATE = 'gaugeFlowRate'
    TEMPERATURE = 'gaugeTemperature'

class Constants.HeatExchangerPressureSpecification
    Bases: object

    DISCHARGEPRESSURE = 'dischargePressure'
    PRESSUREDROP = 'pressureDrop'

class Constants.HeatExchangerTemperatureSpecification
    Bases: object

    DELTATEMPERATURE = 'deltaTemperature'
    DISCHARGETEMPERATURE = 'dischargeTemperature'
    DUTY = 'duty'
```

```
class Constants.OilCalibrationType
    Bases: object

    DENSITY = 'Density'
    OFVF = 'OFVF'

class Constants.NetworkDiagramLayerProperty
    Bases: object

    DEFAULT = 'Default'
    LINKLAYER = 'LinkLayer'
    EQUIPMENT = 'Equipment'
    JUNCTION = 'Junction'
    SINKSOURCE = 'SinkSource'
    FOLDERS = 'Folders'
    GAUGES = 'Gauges'
    PROPERTYPANELS = 'PropertyPanels'
    CUSTOM = 'Custom'

class Constants.SensitivityVariableType
    Bases: object

    NONE = 'varNone'
    INFLOW = 'varInflow'
    OUTFLOW = 'varOutflow'
    XAXIS = 'varXAxis'
    SENSITIVITY = 'varSensitivity'

class Constants.CalculatedVariable
    Bases: object

    INLETPRESSURE = 'calcInletPressure'
    OUTLETPRESSURE = 'calcOutletPressure'
    FLOWRATE = 'calcFlowrate'
    CUSTOM = 'calcCustom'

class Constants.Fluid
    Bases: object

    LIQUID = 'liquid'
    GAS = 'gas'
    WATER = 'water'

class Constants.FluidType
    Bases: object

    BLACKOIL = 'fluidBlackOil'
    COMPOSITIONAL = 'fluidCompositional'
    PVT = 'fluidPVT'
```

```
MFL = 'fluidMFL'
STEAM = 'fluidSteam'

class Constants.PVTPackage
    Bases: object

    E300 = 'E300'
    GERG = 'GERG'
    MULTIFLASH = 'MULTIFLASH'

class Constants.MultiphaseBoosterType
    Bases: object

    GENERIC = 'GenericBooster'
    ONESUBSEA = 'OneSubseaBooster'

class Constants.OneSubseaBoosterModel
    Bases: object

    HX310_250_180 = 'FRAMO Helico-Axial 310-250/180'
    HX310_400_180 = 'FRAMO Helico-Axial 310-400/180'
    HX310_500_45 = 'FRAMO Helico-Axial 310-500/45'
    HX310_500_180 = 'FRAMO Helico-Axial 310-500/180'
    HX310_600_120 = 'FRAMO Helico-Axial 310-600/120'
    HX310_700_45 = 'FRAMO Helico-Axial 310-700/45'
    HX310_800_120 = 'FRAMO Helico-Axial 310-800/120'
    HX310_900_45 = 'FRAMO Helico-Axial 310-900/45'
    HX310_1100_45 = 'FRAMO Helico-Axial 310-1100/45'
    HX310_1100_120 = 'FRAMO Helico-Axial 310-1100/120'
    HX360_1200_38 = 'FRAMO Helico-Axial 360-1200/38'
    HX360_1500_38 = 'FRAMO Helico-Axial 360-1500/38'
    HX360_1800_38 = 'FRAMO Helico-Axial 360-1800/38'

class Constants.SeparatorProductFluid
    Bases: object

    LIQUID = 'liquid'
    GAS = 'gas'
    WATER = 'water'
    OIL = 'oil'
    GASOIL = 'gasOil'
    GASWATER = 'gasWater'

class Constants.TubingSectionType
    Bases: object

    CASING = 'Casing'
    LINER = 'Liner'
```

```
OPENHOLE = 'OpenHole'
TUBING = 'Tubing'
class Constants.AnnulusMaterial
    Bases: object
    CEMENT = 'Cement'
    MUD = 'Mud'
    BRINE = 'Brine'
    WATER = 'Water'
    GAS = 'Gas'
    OIL = 'Oil'
    DIESEL = 'Diesel'
    ACID = 'Acid'
class Constants.UserEquipmentType
    Bases: object
    SURFACE = 'Surface'
    ARTIFICIALLIFT = 'ArtificialLift'
    DOWNHOLE = 'Downhole'
class Constants.CheckValveSetting
    Bases: object
    NONE = 'None'
    BLOCKFORWARD = 'BlockForward'
    BLOCKREVERSE = 'BlockReverse'
    BLOCKBOTH = 'Off'
class Constants.GasLiftInputOption
    Bases: object
    USEGASSPECIFICGRAVITY = 'UseGasSpecificGravity'
    ASSOCIATEEXISTINGFLUID = 'AssociateExistingFluid'
class Constants.NitrogenCorrectionMethod
    Bases: object
    DAKSUTTON = 'DakSutton'
    WRINKLEREADES1989 = 'WrinklerEades1989'
class Constants.GenericEquipmentPressureSpecification
    Bases: object
    DISCHARGEPRESSURE = 'dischargePressure'
    DELTAPRESSURE = 'deltaPressure'
    PRESSURERATIO = 'pressureRatio'
class Constants.GenericEquipmentThermodynamics
    Bases: object
```

```
    DEFAULT = 'defaultRoute'
    ISENTHALPIC = 'isenthalpicRoute'
    ISENTROPIC = 'isentropicRoute'
    ISOTHERMAL = 'isothermalRoute'

class Constants.CompressorThermodynamics
    Bases: object

    ADIABATIC = 'routeAdiabatic'
    POLYTROPIC = 'routePolytropic'
    MOLLIER = 'routeMollier'

class Constants.ViscosityCorrection
    Bases: object

    NONE = 'None'
    CENTRILIFT = 'Centrilift'
    REDA = 'Reda'
    TURZO = 'Turzo'
    USER = 'User'

class Constants.ProfilePlotOption
    Bases: object

    DEFAULT = 'Default'
    ELEVATIONVSPRESSURE = 'ElevationVsPressure'
    ELEVATIONVSTEMPERATURE = 'ElevationVsTemperature'
    PRESSUREVSTOTAL_DISTANCE = 'PressureVsTotal_Distance'
    TEMPERATUREVSTOTAL_DISTANCE = 'TemperatureVsTotal_Distance'

class Constants.PumpThermodynamics
    Bases: object

    ADIABATIC = 'Adiabatic'
    MOLLIER = 'Mollier'
    ISOTHERMAL = 'Isothermal'

class Constants.CompletionFluidEntry
    Bases: object

    SINGLEPOINT = 'SinglePoint'
    DISTRIBUTED = 'Distributed'

class Constants.Orientation
    Bases: object

    VERTICAL = 'Vertical'
    HORIZONTAL = 'Horizontal'

class Constants.BoreholeFluid
    Bases: object
```

```
MUD = 'Mud'
BRINE = 'Brine'
WATER = 'Water'
GAS = 'Gas'
OIL = 'Oil'
DIESEL = 'Diesel'
ACID = 'Acid'
class Constants.F FormationFluid
    Bases: object
    OIL = 'Oil'
    WATER = 'Water'
    GAS = 'Gas'
class Constants.R RockType
    Bases: object
    SANDSTONE = 'Sandstone'
    LIMESTONE = 'Limestone'
    DOLOMITE = 'Dolomite'
    SHALE = 'Shale'
    COAL = 'Coal'
class Constants.P PerforationFlowRate
    Bases: object
    LIQUIDFLOWRATE = 'LiquidFlowRate'
    GASFLOWRATE = 'GasFlowRate'
class Constants.S SensitivityMethod
    Bases: object
    PERMUTED = 'Permuted'
    STEPWITHVARIABLE1 = 'StepWithVariable1'
    STEPWITHXAXIS = 'StepWithXAxis'
class Constants.V VFPTablesOperationTable
    Bases: object
    ECLIPSE = 'TableEclipse'
    PORES = 'TablePores'
    VIP = 'TableVIP'
    COMP4 = 'TableComp4'
    MORES = 'TableMoRes'
class Constants.G GasLift
    Bases: object
    Gas Lift Design constants
```

```
class DiagnosticsOperationDiagnostics
    Bases: object

    FIXEDPRESSURE = 'FixedPressure'
    FIXEDINJECTION = 'FixedInjection'
    FIXEDBOTH = 'FixedBoth'

class Constants.GasLift.DiagnosticsOperationThrottling
    Bases: object

    AUTO = 'Auto'
    ON = 'On'
    OFF = 'Off'

class Constants.GasLift.PressureGradient
    Bases: object

    FRICTIONELEVATION = 'FrictionElevation'
    STATIC = 'Static'

class Constants.GasLift.SolutionPoint
    Bases: object

    LIQUIDPRODRATE = 'LiquidProdRate'
    RESERVOIRPRESSURE = 'ReservoirPressure'
    INJECTIONRATE = 'InjectionRate'
    INJECTIONPRESSURE = 'InjectionPressure'

class Constants.GasLift.DesignSpacing
    Bases: object

    NEWSPACING = 'NewSpacing'
    CURRENTSPACING = 'CurrentSpacing'
    OPTIMUMDEPTH = 'OptimumDepth'
    VALVEPORTDEPTH = 'ValvePortDepth'

class Constants.GasLift.ResponseSolution
    Bases: object

    LIQUIDPRODRATE = 'LiquidProdRate'
    RESERVOIRPRESSURE = 'ReservoirPressure'
    INJECTIONRATE = 'InjectionRate'
    INJECTIONPRESSURE = 'InjectionPressure'

class Constants.GasLift.TransferFactor
    Bases: object

    PINJPPROD = 'PinjPprod'
    PPROD = 'Pprod'

class Constants.GasLift.TransferGradient
    Bases: object

    SURFACEOFFSET = 'SurfaceOffset'
```

```
SURFACEOFFSETDP = 'SurfaceOffsetDP'

class Constants.GasLift.DesignMethod
    Bases: object

    IPOSURFACECLOSE = 'IpoSurfaceClose'

    IPOPTMINMAX = 'IpoPtMinMax'

    PPODESIGN = 'PpoDesign'

class Constants.GasLift.UnloadingTemperature
    Bases: object

    INJECTION = 'Injection'

    PRODUCTION = 'Production'

    AMBIENT = 'Ambient'

    UNLOADING = 'Unloading'

class Constants.GasLift.ProductionPressureCurve
    Bases: object

    MODEL = 'Model'

class Constants.GasLift.DesignSolutionPoint
    Bases: object

    LIQUIDPRODRATE = 'LiquidProdRate'

    RESERVOIRPRESSURE = 'ReservoirPressure'

    INJECTIONRATE = 'InjectionRate'

    INJECTIONPRESSURE = 'InjectionPressure'

class Constants.GasLift.TopValveLocation
    Bases: object

    LIQUIDTOSURFACE = 'LiquidToSurface'

    LIQUIDLEVEL = 'LiquidLevel'

    CALCLIQUIDLEVEL = 'CalcLiquidLevel'

class Constants.GasLift.OperatingValves
    Bases: object

    ORIFICE = 'Orifice'

    IPO = 'IPO'

    PPO = 'PPO'

class Constants.GasLift.DesignTreatTransferGradient
    Bases: object

    OPENINGPRESSURE = 'OpeningPressure'

    CLOSINGPRESSURE = 'ClosingPressure'

class Constants.GasLift.DesignTransferGradientDP
    Bases: object

    INJECTIONPRESSURE = 'InjectionPressure'

    PRODUCTIONPRESSURE = 'ProductionPressure'
```



```
class Constants.BlackOilCalibrationSolutionGas
    Bases: object
    LASATER = 'Lasater'
    STANDING = 'Standing'
    VASQUEZANDBEGGS = 'VasquezAndBeggs'
    GLASO = 'Glaso'
    KARTOATMODJO = 'Kartoatmodjo'
    DEGHETTOETAL = 'DeGhettoEtAl'
    PETROSKYFARSHAD = 'PetroskyFarshad'

class Constants.MultiflashComponent
    Bases: object
    ONE_2_DIETHYLBENZENE = '1,2-Diethylbenzene'
    TWO_2_DIMETHYLPROPANE = '2,2-Dimethylpropane'
    THREE_METHYL_HEXANE = '3-methyl hexane'
    AMMONIA = 'Ammonia'
    ARGON = 'Argon'
    BENZENE = 'Benzene'
    BUTANE = 'Butane'
    CARBON_DIOXIDE = 'Carbon Dioxide'
    CARBON_MONOXIDE = 'Carbon Monoxide'
    CUMENE = 'Cumene'
    CYCLOHEXANE = 'Cyclohexane'
    CYCLOPENTANE = 'Cyclopentane'
    DECANE = 'Decane'
    DIETHYLENE_GLYCOL = 'Diethylene Glycol'
    DOCOSANE = 'Docosane'
    DODECANE = 'Dodecane'
    DOTRIACONTANE = 'Dotriacontane'
    EICOSANE = 'Eicosane'
    ETHANE = 'Ethane'
    ETHANOL = 'Ethanol'
    ETHYLBENZENE = 'Ethylbenzene'
    ETHYLCYCLOHEXANE = 'Ethylcyclohexane'
    ETHYLENE = 'Ethylene'
    ETHYLENE_GLYCOL = 'Ethylene Glycol'
    HELIUM = 'Helium'
    HENEICOSANE = 'Heneicosane'
```

HEPTACOSANE = 'Heptacosane'
HEPTADECANE = 'Heptadecane'
HEPTANE = 'Heptane'
HEXACOSANE = 'Hexacosane'
HEXADECANE = 'Hexadecane'
HEXANE = 'Hexane'
HEXATRIACONTANE = 'Hexatriacontane'
HYDROGEN = 'Hydrogen'
HYDROGEN_SULPHIDE = 'Hydrogen Sulphide'
ISOBUTANE = 'Isobutane'
ISOPENTANE = 'Isopentane'
METHANE = 'Methane'
METHANOL = 'Methanol'
METHYLCYCLOHEXANE = 'Methylcyclohexane'
METHYLCYCLOPENTANE = 'Methylcyclopentane'
M_XYLENE = 'M-Xylene'
NAPHTHALENE = 'Naphthalene'
NITROGEN = 'Nitrogen'
NONACOSANE = 'Nonacosane'
NONADECANE = 'Nonadecane'
NONANE = 'Nonane'
OCTACOSANE = 'Octacosane'
OCTADECANE = 'Octadecane'
OCTANE = 'Octane'
OXYGEN = 'Oxygen'
O_XYLENE = 'O-Xylene'
PENTACOSANE = 'Pentacosane'
PENTADECANE = 'Pentadecane'
PENTANE = 'Pentane'
PROPANE = 'Propane'
P_XYLENE = 'P-Xylene'
SALT_COMPONENT = 'Salt Component'
TETRACOSANE = 'Tetracosane'
TETRADECANE = 'Tetradecane'
TOLUENE = 'Toluene'
TRIACONTANE = 'Triacontane'

```
TRICOSANE = 'Tricosane'
TRIDECANE = 'Tridecane'
TRIETHYLENE_GLYCOL = 'Triethylene Glycol'
UNDECANE = 'Undecane'
WATER = 'Water'

class Constants.E300Component
    Bases: object
    BEN = 'BEN'
    C1 = 'C1'
    C10 = 'C10'
    C11 = 'C11'
    C12 = 'C12'
    C13 = 'C13'
    C14 = 'C14'
    C15 = 'C15'
    C16 = 'C16'
    C17 = 'C17'
    C18 = 'C18'
    C19 = 'C19'
    C2 = 'C2'
    C20 = 'C20'
    C21 = 'C21'
    C22 = 'C22'
    C23 = 'C23'
    C24 = 'C24'
    C25 = 'C25'
    C26 = 'C26'
    C27 = 'C27'
    C28 = 'C28'
    C29 = 'C29'
    C3 = 'C3'
    C30 = 'C30'
    C31 = 'C31'
    C32 = 'C32'
    C33 = 'C33'
    C34 = 'C34'
```

```
C35 = 'C35'
C36 = 'C36'
C37 = 'C37'
C38 = 'C38'
C39 = 'C39'
C4 = 'C4'
C40 = 'C40'
C41 = 'C41'
C42 = 'C42'
C43 = 'C43'
C44 = 'C44'
C45 = 'C45'
C5 = 'C5'
C6 = 'C6'
C7 = 'C7'
C8 = 'C8'
C9 = 'C9'
CO = 'CO'
CO2 = 'CO2'
CYCLOC6 = 'CycloC6'
ETHYLBENZENE = 'Ethylbenzene'
H2 = 'H2'
H2O = 'H2O'
H2S = 'H2S'
IC4 = 'IC4'
IC5 = 'IC5'
MCYCLOC5 = 'MecycloC5'
MCYCLOC6 = 'MecycloC6'
MPXYLENE = 'MPXylene'
N2 = 'N2'
NC4 = 'NC4'
NC5 = 'NC5'
OXYLENE = 'OXylene'
TOL = 'TOL'
```

```
class Constants.GERGComponent
    Bases: object
```

```
ARGON = 'Argon'
CARBON_DIOXIDE = 'Carbon dioxide'
CARBON_MONOXIDE = 'Carbon monoxide'
ETHANE = 'Ethane'
HELIUM = 'Helium'
HYDROGEN = 'Hydrogen'
HYDROGEN_SULPHIDE = 'Hydrogen sulphide'
ISOBUTANE = 'Isobutane'
ISOPENTANE = 'Isopentane'
METHANE = 'Methane'
N_BUTANE = 'n-Butane'
N_DECANE = 'n-Decane'
N_HEPTANE = 'n-Heptane'
N_HEXANE = 'n-Hexane'
NITROGEN = 'Nitrogen'
N_NONANE = 'n-Nonane'
N_OCTANE = 'n-Octane'
N_PENTANE = 'n-Pentane'
OXYGEN = 'Oxygen'
PROPANE = 'Propane'
WATER = 'Water'

class Constants.EmulsionViscosityMethod
    Bases: object

    Set to viscosity of the continuous phase
    CONTINUOUSPHASE = 'ContinuousPhase'
    OILWATERVOLUMERATIO = 'OilWaterVolumeRatio'
    ORIGINALWOELFLIN = 'OriginalWoelflin'
    WOELFLINLOOSE = 'WoelflinLoose'
    WOELFLINMEDIUM = 'WoelflinMedium'
    WOELFLINTIGHT = 'WoelflinTight'
    BRINKMAN = 'Brinkman'
    VANDVAND = 'VandVand'
    VANDBARNEAMIZRAHI = 'VandBarneaMizrahi'
    VANDUSER = 'VandUser'
    RICHARDSON = 'Richardson'
    LEVITONLEIGHTON = 'LevitonLeighton'
```

```
USERTABLE = 'UserTable'

class Constants.EquationOfState
    Bases: object

    class Multiflash
        Bases: object

        PENGROBINSON2PARAMETER = 'PengRobinson2Parameter'
        PENGROBINSON3PARAMETER = 'PengRobinson3Parameter'
        REDLICKKWONG = 'RedlichKwong'
        SOAVEREDLICKKWONG2PARAMETER = 'SoaveRedlichKwong2Parameter'
        SOAVEREDLICKKWONG3PARAMETER = 'SoaveRedlichKwong3Parameter'
        SOAVEREDLICKKWONG3PARAMETERNRTL MIXINGRULE =
            'SoaveRedlichKwong3ParameterNRTLMixingRule'
        BENEDICTWEBBRUBINSTARLING = 'BenedictWebbRubinStarling'
        CUBICPLUSASSOCIATION = 'CubicPlusAssociation'
        CORRESPONDINGSTATES = 'CorrespondingStates'

    class Constants.EquationOfState.GERG
        Bases: object

        GERG_REFPROP = 'GERG_REFPROP'

    class Constants.EquationOfState.E300
        Bases: object

        PENGROBINSON3PARAMETERCORRECTED = 'PengRobinson3ParameterCorrected'
        PENGROBINSON2PARAMETERCORRECTED = 'PengRobinson2ParameterCorrected'
        SOAVEREDLICKKWONG3PARAMETER = 'SoaveRedlichKwong3Parameter'
        SOAVEREDLICKKWONG2PARAMETER = 'SoaveRedlichKwong2Parameter'

class Constants.ViscosityCorrelationMethod
    Bases: object

    LOHRENZBRAYCLARK = 'LohrenzBrayClark'
    PEDERSEN = 'Pedersen'
    AASBERGPETERSEN = 'AasbergPetersen'
    NIST = 'NIST'
    PEDERSENTWU = 'PedersenTwu'
    SUPERTRAPP = 'SuperTRAPP'

class Constants.VolumeShiftCorrelationMethod
    Bases: object

    SORIEDE = 'Soriede'
    PENELOUXDBR = 'PenelouxDBR'
    MULTIFLASH = 'Multiflash'
```

```
class Constants.CriticalPropertyCorrelationMethod
    Bases: object

    KESLERLEE = 'KeslerLee'

    CAVETT = 'Cavett'

    RIAZIDaubert = 'RiaziDaubert'

    WINN = 'Winn'

    PEDERSEN = 'Pedersen'

    SINGLECARBONNUMBER = 'SingleCarbonNumber'

class Constants.ThermalCoefficientsCorrelationMethod
    Bases: object

    KESLERLEE = 'KeslerLee'

    KESLERLEEMODIFIED = 'KeslerLeeModified'

    MULTIFLASH = 'Multiflash'

class Constants.AcfCorrelationMethod
    Bases: object

    KESLERLEE = 'KeslerLee'

    EDMISTER = 'Edmister'

    THOMASSEN = 'Thomassen'

    PEDERSEN = 'Pedersen'

    SINGLECARBONNUMBER = 'SingleCarbonNumber'

class Constants.SalinityModel
    Bases: object

    NONE = 'None'

    IONANALYSIS = 'IonAnalysis'

    TDS = 'TDS'

class Constants.FluidComponentType
    Bases: object

    HYDROCARBON = 'Hydrocarbon'

    NONHYDROCARBON = 'NonHydrocarbon'

class sixgill.definitions.SystemVariables
    Bases: object

    The System and Equipment Results Variables

    ALHANATI_CRITERION1 = 'AlhanatiCriterion1'

    ALHANATI_CRITERION2 = 'AlhanatiCriterion2'

    ARTIFICIAL_LIFT_QUANTITY = 'ArtificialLiftQuantity'

    BOTTOM_HOLE_PRESSURE = 'BottomHolePressure'

    CASE_NUMBER = 'CaseNumber'

    CASE_STATUS = 'CaseStatus'
```

```
CASING_HEAD_PRESSURE = 'CasingHeadPressure'
CONED_GOR = 'ConedGOR'
ESP_DELTA_PRESSURE = 'ESPDeltaPressure'
ESP_DELTA_TEMPERATURE = 'ESPDeltaTemperature'
ESP_DISCHARGE_PRESSURE = 'ESPDDischargePressure'
ESP_EFFICIENCY = 'ESPEfficiency'
ESP_EFFICIENCY_FACTOR = 'ESPEfficiencyFactor'
ESP_FLOWRATE_FACTOR = 'ESPFlowrateFactor'
ESP_FREQUENCY = 'ESPFrequency'
ESP_HEAD = 'ESPHead'
ESP_HEAD_FACTOR = 'ESPHeadFactor'
ESP_INTAKE_GAS_VOLUME_FRACTION = 'ESPIntakeGasVolumeFraction'
ESP_INTAKE_PRESSURE = 'ESPIntakePressure'
ESP_INTAKE_TOTAL_VOLUMETRIC_FLOWRATE = 'ESPIntakeTotalVolumetricFlowrate'
ESP_NUMBER_OF_STAGES = 'ESPNumberOfStages'
ESP_POWER = 'ESPPower'
ESP_POWER_FACTOR = 'ESPPowerFactor'
ESP_SUCTION_GAS_VOLUME_FRACTION = 'ESPSuctionGasVolumeFraction'
FRICTION_FACTOR_MEAN = 'FrictionFactorMean'
GAS_INJECTION_DEPTH = 'GasInjectionDepth'
GAS_INJECTION_GAS_TEMPERATURE_BEFORE_MIXING = 'GasInjectionGasTemperatureBeforeMixing'
GAS_INJECTION_PRODUCTION_TEMPERATURE_DOWNSTREAM =
    'GasInjectionProductionTemperatureDownstream'
GAS_INJECTION_PRODUCTION_TEMPERATURE_UPSTREAM = 'GasInjectionProductionTemperatureUpstream'
GAS_INJECTION_VALVE_DELTA_TEMPERATURE = 'GasInjectionValveDeltaTemperature'
GAS_INJECTION_VALVE_DELTA_PRESSURE = 'GasInjectionValveDeltaPressure'
GAS_LIFT_INJECTION_CASING_HEAD_TEMPERATURE = 'GasLiftInjectionCasingHeadTemperature'
GAS_LIFT_INJECTION_PORT_DIAMETER = 'GasLiftInjectionPortDiameter'
GAS_LIFT_INJECTION_CASING_PRESSURE = 'GasLiftInjectionCasingPressure'
GAS_LIFT_INJECTION_PRESSURE = 'GasLiftInjectionPressure'
GAS_LIFT_INJECTION_RATIO = 'GasLiftInjectionRatio'
GAS_LIFT_INJECTION_SOURCE_PRESSURE = 'GasLiftInjectionSourcePressure'
GAS_LIFT_INJECTION_CASING_TEMPERATURE = 'GasLiftInjectionCasingTemperature'
GAS_LIFT_INJECTION_TEMPERATURE = 'GasLiftInjectionTemperature'
HEAT_TRANSFER_COEFFICIENT = 'HeatTransferCoefficient'
HEEL_GOR_STOCKTANK = 'HeelGORStockTank'
```



```
HEEL_MASS_FLOWRATE_FLUID = 'HeelMassFlowrateFluid'
HEEL_PRESSURE = 'HeelPressure'
HEEL_RESERVOIR_DRAWDOWN = 'HeelReservoirDrawdown'
HEEL_TEMPERATURE = 'HeelTemperature'
HEEL_VELOCITY_FLUID = 'HeelVelocityFluid'
HEEL_VOLUME_FLOWRATE_FLUID_INSITU = 'HeelVolumeFlowrateFluidInSitu'
HEEL_VOLUME_FLOWRATE_FLUID_STOCKTANK = 'HeelVolumeFlowrateFluidStockTank'
HEEL_VOLUME_FLOWRATE_GAS_STOCKTANK = 'HeelVolumeFlowrateGasStockTank'
HEEL_VOLUME_FLOWRATE_LIQUID_INSITU = 'HeelVolumeFlowrateLiquidInSitu'
HEEL_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'HeelVolumeFlowrateLiquidStockTank'
HEEL_VOLUME_FLOWRATE_OIL_STOCKTANK = 'HeelVolumeFlowrateOilStockTank'
HEEL_WATER_CUT_STOCKTANK = 'HeelWaterCutStockTank'
INLET_EROSIONAL_VELOCITY_RATIO = 'InletErosionalVelocityRatio'
INLET_GOR_STOCKTANK = 'InletGORStockTank'
INLET_MASS_FLOWRATE_FLUID = 'InletMassFlowrateFluid'
INLET_MASS_FRACTION_GAS = 'InletMassFractionGas'
INLET_VELOCITY_FLUID = 'InletVelocityFluid'
INLET_VELOCITY_GAS = 'InletVelocityGas'
INLET_VELOCITY_LIQUID = 'InletVelocityLiquid'
INLET_VOLUME_FLOWRATE_FLUID_INSITU = 'InletVolumeFlowrateFluidInSitu'
INLET_VOLUME_FLOWRATE_FLUID_STOCKTANK = 'InletVolumeFlowrateFluidStockTank'
INLET_VOLUME_FLOWRATE_GAS_INSITU = 'InletVolumeFlowrateGas'
INLET_VOLUME_FLOWRATE_GAS_STOCKTANK = 'InletVolumeFlowrateGasStockTank'
INLET_VOLUME_FLOWRATE_LIQUID_INSITU = 'InletVolumeFlowrateLiquid'
INLET_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'InletVolumeFlowrateLiquidStockTank'
INLET_VOLUME_FLOWRATE_OIL_STOCKTANK = 'InletVolumeFlowrateOilStockTank'
INLET_VOLUME_FRACTION_LIQUID = 'InletVolumeFractionLiquid'
INLET_WATER_CUT_STOCKTANK = 'InletWaterCutStockTank'
INPUT_FLOWRATE = 'InputFlowrate'
INPUT_TUBING_HEAD_PRESSURE = 'InputTubingHeadPressure'
INPUT_WATERCUT = 'InputWatercut'
INPUT_WATER_RATIO = 'InputWaterRatio'
MAXIMUM_CORROSION_RATE = 'MaximumCorrosionRate'
MAXIMUM_EROSIONAL_VELOCITY_RATIO = 'MaximumErosionalVelocityRatio'
MAXIMUM_EROSION_RATE = 'MaximumErosionRate'
MAXIMUM_HEAT_TRANSFER_COEFFICIENT = 'MaximumHeatTransferCoefficient'
```

```
MAXIMUM_HYDRATE_SUB_COOLING_TEMPERATURE_DIFFERENCE =  
    'MaximumHydrateSubCoolingTemperatureDifference'  
MAXIMUM_LIQUID_LOADING_GAS_RATE = 'MaximumLiquidLoadingGasRate'  
MAXIMUM_LIQUID_LOADING_VELOCITY_RATIO = 'MaximumLiquidLoadingVelocityRatio'  
MAXIMUM_VELOCITY_FLUID = 'MaximumVelocityFluid'  
MAXIMUM_VELOCITY_GAS = 'MaximumVelocityGas'  
MAXIMUM_VELOCITY_LIQUID = 'MaximumVelocityLiquid'  
MAXIMUM_WAX_SUB_COOLING_TEMPERATURE_DIFFERENCE = 'MaximumWaxSubCoolingTemperatureDifference'  
NODAL_POINT_BUBBLE_POINT_PRESSURE = 'NodalPointBubblePointPressure'  
NODAL_POINT_ENTHALPY_FLUID = 'NodalPointEnthalpyFluid'  
NODAL_POINT_FRACTION_LIQUID = 'NodalPointFractionLiquid'  
NODAL_POINT_GLR_STOCKTANK = 'NodalPointGLRStockTank'  
NODAL_POINT_LGR_STOCKTANK = 'NodalPointLGRStockTank'  
NODAL_POINT_MASS_FLOWRATE_FLUID = 'NodalPointMassFlowrateFluid'  
NODAL_POINT_PRESSURE = 'NodalPointPressure'  
NODAL_POINT_TEMPERATURE = 'NodalPointTemperature'  
NODAL_POINT_VELOCITY_FLUID = 'NodalPointVelocityFluid'  
NODAL_POINT_VOLUME_FLOWRATE_FLUID_INSITU = 'NodalPointVolumeFlowrateFluidInSitu'  
NODAL_POINT_VOLUME_FLOWRATE_FLUID_STOCKTANK = 'NodalPointVolumeFlowrateFluidStockTank'  
NODAL_POINT_VOLUME_FLOWRATE_GAS_INSITU = 'NodalPointVolumeFlowrateGasInSitu'  
NODAL_POINT_VOLUME_FLOWRATE_GAS_STOCKTANK = 'NodalPointVolumeFlowrateGasStockTank'  
NODAL_POINT_VOLUME_FLOWRATE_LIQUID_INSITU = 'NodalPointVolumeFlowrateLiquidInSitu'  
NODAL_POINT_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'NodalPointVolumeFlowrateLiquidStockTank'  
NODAL_POINT_VOLUME_FLOWRATE_OIL_STOCKTANK = 'NodalPointVolumeFlowrateOilStockTank'  
NODAL_POINT_WATER_CUT_INSITU = 'NodalPointWaterCutInSitu'  
NODAL_POINT_WATER_CUT_STOCKTANK = 'NodalPointWaterCutStockTank'  
OUTLET_EROSIONAL_VELOCITY = 'OutletErosionalVelocity'  
OUTLET_EROSIONAL_VELOCITY_RATIO = 'OutletErosionalVelocityRatio'  
OUTLET_FRACTION_CO = 'OutletFractionCO'  
OUTLET_FRACTION_CO2 = 'OutletFractionCO2'  
OUTLET_FRACTION_H2S = 'OutletFractionH2S'  
OUTLET_FRACTION_N2 = 'OutletFractionN2'  
OUTLET_GLR_STOCKTANK = 'OutletGLRStockTank'  
OUTLET_GOR_STOCKTANK = 'OutletGORStockTank'  
OUTLET_LGR_STOCKTANK = 'OutletLGRStockTank'  
OUTLET_MASS_FLOWRATE_FLUID = 'OutletMassFlowrateFluid'
```

OUTLET_MASS_FLOWRATE_GAS_STOCKTANK = 'OutletMassFlowrateGasStockTank'
OUTLET_MASS_FLOWRATE_OIL_STOCKTANK = 'OutletMassFlowrateOilStockTank'
OUTLET_MASS_FLOWRATE_WATER_STOCKTANK = 'OutletMassFlowrateWaterStockTank'
OUTLET_MASS_FRACTION_GAS = 'OutletMassFractionGas'
OUTLET_SPECIFIC_GRAVITY_GAS = 'OutletSpecificGravityGas'
OUTLET_VELOCITY_FLUID = 'OutletVelocityFluid'
OUTLET_VELOCITY_GAS = 'OutletVelocityGas'
OUTLET_VELOCITY_LIQUID = 'OutletVelocityLiquid'
OUTLET_VOLUME_FLOWRATE_FLUID_INSITU = 'OutletVolumeFlowrateFluidInSitu'
OUTLET_VOLUME_FLOWRATE_FLUID_STOCKTANK = 'OutletVolumeFlowrateFluidStockTank'
OUTLET_VOLUME_FLOWRATE_GAS_INSITU = 'OutletVolumeFlowrateGas'
OUTLET_VOLUME_FLOWRATE_GAS_STOCKTANK = 'OutletVolumeFlowrateGasStockTank'
OUTLET_VOLUME_FLOWRATE_LIQUID_INSITU = 'OutletVolumeFlowrateLiquid'
OUTLET_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'OutletVolumeFlowrateLiquidStockTank'
OUTLET_VOLUME_FLOWRATE_OIL_STOCKTANK = 'OutletVolumeFlowrateOilStockTank'
OUTLET_VOLUME_FLOWRATE_WATER_STOCKTANK = 'OutletVolumeFlowrateWaterStockTank'
OUTLET_VOLUME_FRACTION_LIQUID = 'OutletVolumeFractionLiquid'
OUTLET_WATER_CUT_STOCKTANK = 'OutletWaterCutStockTank'
PCP_DISCHARGE_PRESSURE = 'PCPDischargePressure'
PCP EFFICIENCY = 'PCPEfficiency'
PCP_INTAKE_PRESSURE = 'PCPIntakePressure'
PCP_INTAKE_VOLUMETRIC_FLOWRATE_FLUID = 'PCPIntakeVolumetricFlowrateFluid'
PCP_POWER = 'PCPPower'
PCP_SPEED = 'PCPSpeed'
PIPE_OUTSIDE_DIAMETER = 'PipeOutsideDiameter'
PRESSURE_DROP_TOTAL_ACCELERATION = 'PressureDropTotalAcceleration'
PRESSURE_DROP_TOTAL_COMPLETION = 'PressureDropTotalCompletion'
RECORD_COUNT = 'RecordCount'
RESISTANCE_ELEVATIONAL = 'ResistanceElevational'
RESISTANCE_FRICTIONAL = 'ResistanceFrictional'
REYNOLDS_NUMBER_MEAN = 'ReynoldsNumberMean'
ROD_PUMP_DELTA_PRESSURE = 'RodPumpDeltaPressure'
ROD_PUMP_DISCHARGE_PRESSURE = 'RodPumpDischargePressure'
ROD_PUMP EFFICIENCY = 'RodPumpEfficiency'
ROD_PUMP_INTAKE_VOLUME_FLOWRATE_FREE_GAS = 'RodPumpIntakeVolumeFlowrateFreeGas'
ROD_PUMP_INTAKE_VOLUME_FRACTION_GAS = 'RodPumpIntakeVolumeFractionGas'

```
ROD_PUMP_INTAKE_VOLUME_FLOWRATE_LIQUID = 'RodPumpIntakeVolumeFlowrateLiquid'
ROD_PUMP_INTAKE_PRESSURE = 'RodPumpIntakePressure'
ROD_PUMP_INTAKE_VOLUMETRIC_FLOWRATE_FLUID = 'RodPumpIntakeVolumetricFlowrateFluid'
ROD_PUMP_POWER = 'RodPumpPower'
SEVERE_SLUGGING_INDICATOR = 'SevereSluggingIndicator'
SLUG_FREQUENCY1IN1000 = 'SlugFrequency1In1000'
SLUG_FREQUENCY_MEAN = 'SlugFrequencyMean'
SLUG_LENGTH1IN1000 = 'SlugLength1In1000'
SLUG_LENGTH_MEAN = 'SlugLengthMean'
SLUG_VOLUME1IN1000 = 'SlugVolume1In1000'
SLUG_VOLUME_MEAN = 'SlugVolumeMean'
SPECIFIC_GRAVITY_PRODUCED_GAS = 'SpecificGravityProducedGas'
SPHERE_GENERATED_LIQUID_VOLUME = 'SphereGeneratedLiquidVolume'
SPHERE_GENERATED_LIQUID_VOLUME_DUMPING_TIME = 'SphereGeneratedLiquidVolumeDumpingTime'
SPHERE_TRANSIT_TIME = 'SphereTransitTime'
SURFACE_INJECTION_CHOKE_DIAMETER = 'SurfaceInjectionChokeDiameter'
SYSTEM_INLET_PRESSURE = 'SystemInletPressure'
SYSTEM_OUTLET_PRESSURE = 'SystemOutletPressure'
SYSTEM_OUTLET_TEMPERATURE = 'SystemOutletTemperature'
SYSTEM_PRESSURE_LOSS = 'SystemPressureLoss'
SYSTEM_TEMPERATURE_DIFFERENCE = 'SystemTemperatureDifference'
TIME_HOURS = 'Time_Hours'
TOE_PRESSURE = 'ToePressure'
TOE_RESERVOIR_DRAWDOWN = 'ToeReservoirDrawdown'
TOE_TEMPERATURE = 'ToeTemperature'
TOE_VELOCITY_FLUID = 'ToeVelocityFluid'
TOTAL_INJECTION_GAS = 'TotalInjectionGas'
TOTAL_LIQUID_HOLDUP = 'TotalLiquidHoldup'
TOTAL_OIL_HOLDUP = 'TotalOilHoldup'
TOTAL_PRESSURE_DROP_ELEVATIONAL = 'TotalPressureDropElevational'
TOTAL_PRESSURE_DROP_FRICTIONAL = 'TotalPressureDropFrictional'
TOTAL_WATER_HOLDUP = 'TotalWaterHoldup'
WELLHEAD_GOR_STOCKTANK = 'WellheadGORStockTank'
WELLHEAD_MASS_FLOWRATE_FLUID = 'WellheadMassFlowrateFluid'
WELLHEAD_PRESSURE = 'WellheadPressure'
WELLHEAD_TEMPERATURE = 'WellheadTemperature'
```

WELLHEAD_TRUE_VERTICAL_DEPTH = 'WellheadTrueVerticalDepth'
WELLHEAD_VOLUME_FLOWRATE_FLUID_INSITU = 'WellheadVolumeFlowrateFluidInSitu'
WELLHEAD_VOLUME_FLOWRATE_GAS_STOCKTANK = 'WellheadVolumeFlowrateGasStockTank'
WELLHEAD_VOLUME_FLOWRATE_LIQUID_INSITU = 'WellheadVolumeFlowrateLiquidInSitu'
WELLHEAD_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'WellheadVolumeFlowrateLiquidStockTank'
WELLHEAD_VOLUME_FLOWRATE_OIL_STOCKTANK = 'WellheadVolumeFlowrateOilStockTank'
WELLHEAD_VOLUME_FLOWRATE_FLUID_STOCKTANK = 'WellheadVolumeFlowrateFluidStockTank'
WELLHEAD_WATER_CUT_STOCKTANK = 'WellheadWaterCutStockTank'
API_GRAVITY_OIL = 'ApiGravityOil'
ARTIFICIAL_LIFT_FLOWRATE_CONVERSION_FACTOR = 'ArtificialLiftFlowrateConversionFactor'
ARTIFICIAL_LIFT_FLOWRATE_TYPE = 'ArtificialLiftFlowrateType'
BEAN_SIZE = 'BeanSize'
BRANCH_DELTA_PRESSURE = 'BranchDeltaPressure'
CHOKE_DELTA_PRESSURE = 'ChokeDeltaPressure'
COILED_TUBING_INJECTION_PRESSURE = 'CoiledTubingInjectionPressure'
COMPLETION_VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'CompletionVolumeFlowrateLiquidStockTank'
COMPRESSOR_EFFICIENCY = 'CompressorEfficiency'
COMPRESSOR_POWER = 'CompressorPower'
CONTAMINANTS_CO2 = 'ContaminantsCO2'
CONTAMINANTS_H2S = 'ContaminantsH2S'
CONTAMINANTS_N2 = 'ContaminantsN2'
CRITICAL_PRESSURE_RATIO = 'CriticalPressureRatio'
DELTA_ENTHALPY = 'DeltaEnthalpy'
DELTA_ENTROPY = 'DeltaEntropy'
DELTA_PRESSURE = 'DeltaPressure'
DELTA_TEMPERATURE = 'DeltaTemperature'
DENSITY_FLUID_MEAN = 'DensityFluidMean'
DENSITY_GAS_STOCKTANK = 'DensityGasStockTank'
DENSITY_OIL_STOCKTANK = 'DensityOilStockTank'
DENSITY_WATER_STOCKTANK = 'DensityWaterStockTank'
DIAMETER_RATIO = 'DiameterRatio'
DISCHARGE_COEFFICIENT = 'DischargeCoefficient'
DISCHARGE_PRESSURE = 'DischargePressure'
DOWNSTREAM_DENSITY = 'DownstreamDensity'
DOWNSTREAM_HEAT_CAPACITY_RATIO = 'DownstreamHeatCapacityRatio'
DOWNSTREAM_MACH_NUMBER = 'DownstreamMachNumber'

DOWNSTREAM_PRESSURE = 'DownstreamPressure'
DOWNSTREAM_SONIC_VELOCITY = 'DownstreamSonicVelocity'
DOWNSTREAM_VELOCITY = 'DownstreamVelocity'
DOWNSTREAM_VELOCITY_RATIO_CRITICAL = 'DownstreamVelocityRatioCritical'
DUTY = 'Duty'
EFFICIENCY = 'Efficiency'
ELEVATION_DELTA_PRESSURE = 'ElevationDeltaPressure'
ELEVATION_DIFFERENCE = 'ElevationDifference'
ENTHALPY = 'Enthalpy'
EQUIPMENT_TYPE = 'EquipmentType'
EROSIONAL_VELOCITY = 'ErosionalVelocity'
EROSIONAL_VELOCITY_MAXIMUM = 'ErosionalVelocityMaximum'
EROSIONAL_VELOCITY_RATIO = 'ErosionalVelocityRatio'
EROSIONAL_VELOCITY_RATIO_MAXIMUM = 'ErosionalVelocityRatioMaximum'
EXPANSION_FACTOR_GAS = 'ExpansionFactorGas'
EXPANSION_FACTOR_GAS_CRITICAL = 'ExpansionFactorGasCritical'
FLOW_COEFFICIENT_FLUID = 'FlowCoefficientFluid'
FLOW_COEFFICIENT_GAS = 'FlowCoefficientGas'
FLOW_COEFFICIENT_LIQUID = 'FlowCoefficientLiquid'
FLOW_FACTOR = 'FlowFactor'
FLOWRATE = 'Flowrate'
FLOWRATE_BEYOND_CURVE_MAX_RATE = 'FlowrateBeyondCurveMaxRate'
FLOWRATE_DERIVATIVE_OF_OUTLET_PRESSURE = 'FlowrateDerivativeOfOutletPressure'
FLOWRATE_RATIO_CRITICAL = 'FlowrateRatioCritical'
FLOWRATE_TYPE = 'FlowrateType'
FLUID_HANDLE_STRING = 'FluidHandleString'
FRICTION_DELTA_PRESSURE = 'FrictionDeltaPressure'
GLR_INSITU = 'GLRInSitu'
GLR_STOCKTANK = 'GLRStockTank'
GOR_STOCKTANK = 'GORStockTank'
GWR_STOCKTANK = 'GWRStockTank'
HEAD = 'Head'
COMPRESSOR_HEAD = 'CompressorHead'
HEAT_CAPACITY_RATIO_UPSTREAM = 'HeatCapacityRatioUpstream'
HEEL_DRAWDOWN = 'HeelDrawdown'
INJECTION_FLUID_ENTHALPY = 'InjectionFluidEnthalpy'

```
INJECTION_FLUID_GAS_FLOWRATE_STOCKTANK = 'InjectionFluidGasFlowrateStockTank'
INJECTION_FLUID_LIQUID_FLOWRATE_STOCKTANK = 'InjectionFluidLiquidFlowrateStockTank'
INJECTION_FLUID_MASS_FLOWRATE_STOCKTANK = 'InjectionFluidMassFlowrateStockTank'
INJECTION_FLUID_SPECIFIC_GRAVITY_GAS = 'InjectionFluidSpecificGravityGas'
INJECTION_FLUID_TEMPERATURE = 'InjectionFluidTemperature'
INJECTION_GAS_PRESSURE_AT_CASING_HEAD = 'InjectionGasPressureAtCasingHead'
INJECTION_GAS_PRESSURE_AT_INJECTION_POINT = 'InjectionGasPressureAtInjectionPoint'
INJECTION_GAS_TEMPERATURE_AT_CASING_HEAD = 'InjectionGasTemperatureAtCasingHead'
INJECTION_GAS_TEMPERATURE_AT_INJECTION_POINT = 'InjectionGasTemperatureAtInjectionPoint'
INJECTION_PORT_DELTA_PRESSURE = 'InjectionPortDeltaPressure'
INJECTION_PORT_DELTA_TEMPERATURE = 'InjectionPortDeltaTemperature'
INJECTION_PORT_DEPTH = 'InjectionPortDepth'
INLET_PRESSURE = 'InletPressure'
INLET_PRESSURE_DERIVATIVE_OF_FLOWRATE = 'InletPressureDerivativeOfFlowrate'
INLET_PRESSURE_DERIVATIVE_OF_OUTLET_PRESSURE = 'InletPressureDerivativeOfOutletPressure'
INLET_TEMPERATURE = 'InletTemperature'
INPUT_FLOWRATE_GAS = 'InputFlowrateGas'
INPUT_FLOWRATE_LIQUID = 'InputFlowrateLiquid'
INPUT_FLOWRATE_OIL = 'InputFlowrateOil'
INPUT_MASS_FLOWRATE_FLUID = 'InputMassFlowrateFluid'
INPUT_TEMPERATURE = 'InputTemperature'
IS_CRITICAL_FLOW = 'IsCriticalFlow'
IS_INJECTING_INTO_COMPLETION = 'IsInjectingIntoCompletion'
IS_SUPER_CRITICAL = 'IsSuperCritical'
LIMITED_BY = 'LimitedBy'
LIMIT_RATIO = 'LimitRatio'
LIQUID_FLOWRATE_CRITICAL = 'LiquidFlowrateCritical'
LIQUID_RATE = 'LiquidRate'
MASS_FLOWRATE_CRITICAL = 'MassFlowrateCritical'
MASS_FLOWRATE_FLUID = 'MassFlowrateFluid'
MASS_FLOWRATE_GAS_STOCKTANK = 'MassFlowrateGasStockTank'
MASS_FLOWRATE_OIL_STOCKTANK = 'MassFlowrateOilStockTank'
MASS_FLOWRATE_STOCKTANK = 'MassFlowrateStockTank'
MASS_FLOWRATE_WATER_STOCKTANK = 'MassFlowrateWaterStockTank'
MASS_RATE = 'MassRate'
MASS_RATE_REMOVED = 'MassRateRemoved'
```

MAX_RATIO_CRITICAL = 'MaxRatioCritical'
OGR_STOCKTANK = 'OGRStockTank'
OUTLET_PRESSURE_DERIVATIVE_OF_FLOWRATE = 'OutletPressureDerivativeOfFlowrate'
OUTPUT_FLOWRATE_GAS = 'OutputFlowrateGas'
OUTPUT_FLOWRATE_LIQUID = 'OutputFlowrateLiquid'
OUTPUT_FLOWRATE_OIL = 'OutputFlowrateOil'
OUTPUT_FLOWRATE_WATER = 'OutputFlowrateWater'
OUTPUT_MASS_FLOWRATE_FLUID = 'OutputMassFlowrateFluid'
PCP_TORQUE = 'PcpTorque'
POWER = 'Power'
PRESSURE = 'Pressure'
PRESSURE_DIFFERENCE = 'PressureDifference'
PRESSURE_DIFFERENCE_CRITICAL = 'PressureDifferenceCritical'
PRESSURE_MAXIMUM = 'PressureMaximum'
PRESSURE_RATIO = 'PressureRatio'
PRESSURE_RATIO_CRITICAL = 'PressureRatioCritical'
PRESSURE_REDUCTION_PER_STAGE = 'PressureReductionPerStage'
PUMP_EFFICIENCY = 'PumpEfficiency'
PUMP_POWER = 'PumpPower'
RESERVOIR_TEMPERATURE_AVERAGE = 'ReservoirTemperatureAverage'
ROUTE = 'Route'
SPECIFIC_GRAVITY_GAS_STOCKTANK = 'SpecificGravityGasStockTank'
SPECIFIC_GRAVITY_OIL_INSITU = 'SpecificGravityOilInSitu'
SPECIFIC_GRAVITY_WATER_INSITU = 'SpecificGravityWaterInSitu'
SPEED = 'Speed'
STAGES = 'Stages'
STATIC_DELTA_PRESSURE = 'StaticDeltaPressure'
STATIC_RESERVOIR_PRESSURE = 'StaticReservoirPressure'
STONEWALL_LIMIT_EXCEEDED = 'StonewallLimitExceeded'
SUCTION_INLET_GAS_VOLUME_PERCENT = 'SuctionInletGasVolumePercent'
SUCTION_PRESSURE = 'SuctionPressure'
SUCTION_PRESSURE_DOWNSTREAM_OF_INLET_CHOKE = 'SuctionPressureDownstreamOfInletChoke'
SURFACE_CHOKE_DIAMETER = 'SurfaceChokeDiameter'
SURFACE_INJECTION_PRESSURE = 'SurfaceInjectionPressure'
TEMPERATURE = 'Temperature'
TEMPERATURE_MIXED_FLUID = 'TemperatureMixedFluid'

TEMPERATURE_PRODUCTION_FLUID = 'TemperatureProductionFluid'
TOE_DRAWDOWN = 'ToeDrawdown'
TUBING_PRESSURE = 'TubingPressure'
TYPE = 'Type'
UPSTREAM_DENSITY = 'UpstreamDensity'
UPSTREAM_MACH_NUMBER = 'UpstreamMachNumber'
UPSTREAM_PIPE_ID = 'UpstreamPipeID'
UPSTREAM_PRESSURE = 'UpstreamPressure'
UPSTREAM_SONIC_VELOCITY = 'UpstreamSonicVelocity'
UPSTREAM_VELOCITY = 'UpstreamVelocity'
UPSTREAM_VELOCITY_RATIO_CRITICAL = 'UpstreamVelocityRatioCritical'
VELOCITY_GAS = 'VelocityGas'
VELOCITY_GAS_MAXIMUM = 'VelocityGasMaximum'
VELOCITY_LIQUID = 'VelocityLiquid'
VELOCITY_LIQUID_MAXIMUM = 'VelocityLiquidMaximum'
VELOCITY_MEAN = 'VelocityMean'
VELOCITY_MEAN_MAXIMUM = 'VelocityMeanMaximum'
VISCOSITY_CORRELATION_FACTOR_FOR EFFICIENCY = 'ViscosityCorrelationFactorForEfficiency'
VISCOSITY_CORRELATION_FACTOR_FOR_FLOWRATE = 'ViscosityCorrelationFactorForFlowrate'
VISCOSITY_CORRELATION_FACTOR_FOR_HEAD = 'ViscosityCorrelationFactorForHead'
VISCOSITY_CORRELATION_FACTOR_FOR_POWER = 'ViscosityCorrelationFactorForPower'
VOLUME_FLOWRATE_FLUID = 'VolumeFlowrateFluid'
VOLUME_FLOWRATE_GAS_INSITU = 'VolumeFlowrateGasInSitu'
VOLUME_FLOWRATE_GAS_STOCKTANK = 'VolumeFlowrateGasStockTank'
VOLUME_FLOWRATE_GAS_STANDARD = 'VolumeFlowrateGasStandard'
VOLUME_FLOWRATE_LIQUID_INSITU = 'VolumeFlowrateLiquidInSitu'
VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'VolumeFlowrateLiquidStockTank'
VOLUME_FLOWRATE_OIL_INSITU = 'VolumeFlowrateOilInSitu'
VOLUME_FLOWRATE_OIL_STOCKTANK = 'VolumeFlowrateOilStockTank'
VOLUME_FLOWRATE_WATER_INSITU = 'VolumeFlowrateWaterInSitu'
VOLUME_FLOWRATE_WATER_STOCKTANK = 'VolumeFlowrateWaterStockTank'
VOLUME_FRACTION_GAS = 'VolumeFractionGas'
WATER_CUT_INSITU = 'WaterCutInSitu'
WATER_CUT_STOCKTANK = 'WaterCutStockTank'
WATER_OIL_RATIO = 'WaterOilRatio'
WGR_STOCKTANK = 'WGRStockTank'

```
class sixgill.definitions.ProfileVariables
    Bases: object

    Available Results Profile Variables

    AMBIENT_FLUID_COEFFICIENT_OF_EXPANSION = 'AmbientFluidCoefficientOfExpansion'
    AMBIENT_FLUID_CONDUCTIVITY = 'AmbientFluidConductivity'
    AMBIENT_FLUID_DENSITY = 'AmbientFluidDensity'
    AMBIENT_FLUID_HEAT_CAPACITY = 'AmbientFluidHeatCapacity'
    AMBIENT_FLUID_VELOCITY = 'AmbientFluidVelocity'
    AMBIENT_FLUID_VISCOSITY = 'AmbientFluidViscosity'
    AMBIENT_TEMPERATURE_AT_NODE = 'AmbientTemperatureAtNode'
    ANNULUS_INSIDE_DIAMETER = 'AnnulusInsideDiameter'
    ANNULUS_OUTSIDE_DIAMETER = 'AnnulusOutsideDiameter'
    ASPHALTENE_FORMATION_TEMPERATURE = 'AsphalteneFormationTemperature'
    BP_BUBBLE_LENGTH1IN1000 = 'BPBubbleLength1In1000'
    BP_BUBBLE_LENGTH_MEAN = 'BPBubbleLengthMean'
    BP_SLUG_FREQUENCY = 'BPSlugFrequency'
    BP_SLUG_FREQUENCY1IN1000 = 'BPSlugFrequency1In1000'
    BP_SLUG_LENGTH1IN1000 = 'BPSlugLength1In1000'
    BP_SLUG_LENGTH_MEAN = 'BPSlugLengthMean'
    BP_SLUG_LIQUID_HOLDUP = 'BPSlugLiquidHoldup'
    BUBBLE_POINT_PRESSURE_INSITU = 'BubblePointPressureInSitu'
    BURIAL_DEPTH_CIRCUMFERENCE = 'BurialDepthCircumference'
    BURIAL_DEPTH_FACTOR_EXTRA = 'BurialDepthFactorExtra'
    BURIAL_DEPTH_OF_PIPE_CENTERLINE = 'BurialDepthOfPipeCenterline'
    BURIAL_DEPTH_OF_PIPE_TOPMOST_COAT = 'BurialDepthOfPipeTopmostCoat'
    CASING_GAS_PRESSURE = 'CasingGasPressure'
    CASING_GAS_TEMPERATURE = 'CasingGasTemperature'
    COEF_OF_EXPANSION_GAS = 'CoefOfExpansionGas'
    COEF_OF_EXPANSION_LIQUID = 'CoefOfExpansionLiquid'
    COMPRESSIBILITY_OIL_INSITU = 'CompressibilityOilInSitu'
    CONDUCTIVITY_FLUID_INSITU = 'ConductivityFluidInSitu'
    CONDUCTIVITY_GAS_INSITU = 'ConductivityGasInSitu'
    CONDUCTIVITY_LIQUID_INSITU = 'ConductivityLiquidInSitu'
    CONDUCTIVITY_OIL_INSITU = 'ConductivityOilInSitu'
    CONDUCTIVITY_WATER_INSITU = 'ConductivityWaterInSitu'
    CORROSION_RATE = 'CorrosionRate'
```

CROSS_SECTIONAL_AREA_FOR_FLOW = 'CrossSectionalAreaForFlow'
CUMULATIVE_ACCELERATION_PRESSURE_DIFFERENCE = 'CumulativeAccelerationPressureDifference'
CUMULATIVE_ELEVATION_PRESSURE_DIFFERENCE = 'CumulativeElevationPressureDifference'
CUMULATIVE_FRICTION_PRESSURE_DIFFERENCE = 'CumulativeFrictionPressureDifference'
CUMULATIVE_GAS_HOLDUP = 'CumulativeGasHoldup'
CUMULATIVE_LIQUID_HOLDUP = 'CumulativeLiquidHoldup'
CUMULATIVE_OIL_HOLDUP = 'CumulativeOilHoldup'
CUMULATIVE_PIPELINE_VOLUME = 'CumulativePipelineVolume'
CUMULATIVE_WATER_HOLDUP = 'CumulativeWaterHoldup'
DENSITY_FLUID_INSITU = 'DensityFluidInSitu'
DENSITY_FLUID_NO_SLIP_INSITU = 'DensityFluidNoSlipInSitu'
DENSITY_GAS_INSITU = 'DensityGasInSitu'
DENSITY_GAS_STOCKTANK = 'DensityGasStockTank'
DENSITY_LIQUID_INSITU = 'DensityLiquidInSitu'
DENSITY_LIQUID_STOCKTANK = 'DensityLiquidStockTank'
DENSITY_OIL_INSITU = 'DensityOilInSitu'
DENSITY_OIL_STOCKTANK = 'DensityOilStockTank'
DENSITY_WATER_INSITU = 'DensityWaterInSitu'
DENSITY_WATER_STOCKTANK = 'DensityWaterStockTank'
DISTRIBUTED_PI_GAS_INSITU = 'DistributedPIGasInSitu'
DISTRIBUTED_PI_LIQUID_INSITU = 'DistributedPILiquidInSitu'
EFFECTIVE_PERMEABILITY = 'EffectivePermeability'
ELEVATION = 'Elevation'
ENTHALPY_FLUID = 'EnthalpyFluid'
ENTROPY_FLUID_INSITU = 'EntropyFluidInSitu'
EQUIVALENT_HYDRAULIC_DIAMETER = 'EquivalentHydraulicDiameter'
EROSIONAL_VELOCITY = 'ErosionalVelocity'
EROSIONAL_VELOCITY_RATIO = 'ErosionalVelocityRatio'
EROSION_RATE = 'ErosionRate'
FLOW_PATTERN_GAS_LIQUID = 'FlowPatternGasLiquid'
FLOW_PATTERN_OIL_WATER = 'FlowPatternOilWater'
FLOWRATE_GAS_INSITU = 'FlowrateGasInSitu'
FLOWRATE_LIQUID_INSITU = 'FlowrateLiquidInSitu'
FORMATION_VOLUME_FACTOR_OIL_INSITU = 'FormationVolumeFactorOilInSitu'
FRICTION_FACTOR_RATIO_TWO_PHASE = 'FrictionFactorRatioTwoPhase'
FRICTION_FACTOR_SINGLE_PHASE = 'FrictionFactorSinglePhase'

```
FROUDE_NUMBER_GAS = 'FroudeNumberGas'
FROUDE_NUMBER_LIQUID = 'FroudeNumberLiquid'
GLR_STOCKTANK = 'GLRStockTank'
GOR_STOCKTANK = 'GORStockTank'
GROUND_CONDUCTIVITY = 'GroundConductivity'
GROUND_HEAT_TRANSFER_COEFFICIENT = 'GroundHeatTransferCoefficient'
GROUND_HEAT_TRANSFER_COEFFICIENT_NO_ADJUSTMENT_BY_AREA_RATIO =
'GroundHeatTransferCoefficientNoAdjustmentByAreaRatio'
GWR_STOCKTANK = 'GWRStockTank'
HEAT_CAPACITY_FLUID_INSITU = 'HeatCapacityFluidInSitu'
HEAT_CAPACITY_GAS_INSITU = 'HeatCapacityGasInSitu'
HEAT_CAPACITY_LIQUID_INSITU = 'HeatCapacityLiquidInSitu'
HEAT_CAPACITY_OIL_INSITU = 'HeatCapacityOilInSitu'
HEAT_CAPACITY_WATER_INSITU = 'HeatCapacityWaterInSitu'
HOLDUP_FRACTION_LIQUID = 'HoldupFractionLiquid'
HOLDUP_LIQUID_WATER = 'HoldupLiquidWater'
HORIZONTAL_DISTANCE = 'HorizontalDistance'
HORIZONTAL_POSITION = 'HorizontalPosition'
HYDRATE_FORMATION_TEMPERATURE = 'HydrateFormationTemperature'
HYDRATE_SUB_COOLING_DELTA_TEMPERATURE = 'HydrateSubCoolingDeltaTemperature'
HYDROSTATIC_HEAD = 'HydrostaticHead'
INLINE_HEATER_MINIMUM_TEMPERATURE = 'InlineHeaterMinimumTemperature'
INLINE_HEATER_POWER_AVAILABLE = 'InlineHeaterPowerAvailable'
INLINE_HEATER_POWER_USED = 'InlineHeaterPowerUsed'
INPUT_HEAT_TRANSFER_COEFFICIENT = 'InputHeatTransferCoefficient'
INSIDE_FILM_FORCED_CONV_NUSSELT_NUMBER = 'InsideFilmForcedConvNusseltNumber'
INSIDE_FILM_GAS_GRASHOF_NUMBER = 'InsideFilmGasGrashofNumber'
INSIDE_FILM_GAS_HEAT_TRANSFER_COEFFICIENT = 'InsideFilmGasHeatTransferCoefficient'
INSIDE_FILM_GAS_NUSSELT_NUMBER = 'InsideFilmGasNusseltNumber'
INSIDE_FILM_GAS_PRANDTL_NUMBER = 'InsideFilmGasPrandtlNumber'
INSIDE_FILM_GAS_REYNOLDS_NUMBER = 'InsideFilmGasReynoldsNumber'
INSIDE_FILM_GRASHOF_NUMBER = 'InsideFilmGrashofNumber'
INSIDE_FILM_GRASHOF_NUMBER_LIQUID = 'InsideFilmGrashofNumberLiquid'
INSIDE_FILM_LIQUID_HEAT_TRANSFER_COEFFICIENT = 'InsideFilmLiquidHeatTransferCoefficient'
INSIDE_FILM_LIQUID_NUSSELT_NUMBER = 'InsideFilmLiquidNusseltNumber'
INSIDE_FILM_LIQUID_PRANDTL_NUMBER = 'InsideFilmLiquidPrandtlNumber'
```

```
INSIDE_FILM_LIQUID_REYNOLDS_NUMBER = 'InsideFilmLiquidReynoldsNumber'
INSIDE_FILM_MEAN_HEAT_TRANSFER_COEFFICIENT = 'InsideFilmMeanHeatTransferCoefficient'
INSIDE_FILM_NATURAL_CONV_NUSSELT_NUMBER = 'InsideFilmNaturalConvNusseltNumber'
INSIDE_FILM_NUSSELT_NUMBER = 'InsideFilmNusseltNumber'
INSIDE_FILM_PRANDTL_NUMBER = 'InsideFilmPrandtlNumber'
INSIDE_FILM_REYNOLDS_NUMBER = 'InsideFilmReynoldsNumber'
INSIDE_FLUID_FILM_HEAT_TRANSFER_COEFFICIENT = 'InsideFluidFilmHeatTransferCoefficient'
JOULE_THOMPSON_COEFFICIENT_INSITU = 'JouleThompsonCoefficientInSitu'
LAYER_STATIC_PRESSURE = 'LayerStaticPressure'
LGR_STOCKTANK = 'LGRStockTank'
LIQUID_LOADING_GAS_RATE = 'LiquidLoadingGasRate'
LIQUID_LOADING_VELOCITY = 'LiquidLoadingVelocity'
LIQUID_LOADING_VELOCITY_RATIO = 'LiquidLoadingVelocityRatio'
LIVE_OIL_SATURATED_VISCOSITY_INSITU = 'LiveOilSaturatedViscosityInSitu'
MACH_NUMBER_IN_FLUID = 'MachNumberInFluid'
MASS_FLOWRATE = 'MassFlowrate'
MASS_FLOWRATE_GAS_INSITU = 'MassFlowrateGasInSitu'
MASS_FLOWRATE_GAS_STOCKTANK = 'MassFlowrateGasStockTank'
MASS_FLOWRATE_LIQUID_INSITU = 'MassFlowrateLiquidInSitu'
MASS_FLOWRATE_LIQUID_STOCKTANK = 'MassFlowrateLiquidStockTank'
MASS_FLOWRATE_OIL_INSITU = 'MassFlowrateOilInSitu'
MASS_FLOWRATE_OIL_STOCKTANK = 'MassFlowrateOilStockTank'
MASS_FLOWRATE_WATER_INSITU = 'MassFlowrateWaterInSitu'
MASS_FLOWRATE_WATER_STOCKTANK = 'MassFlowrateWaterStockTank'
MASS_FRACTION_GAS_INSITU = 'MassFractionGasInSitu'
MASS_FRACTION_LIQUID_INSITU = 'MassFractionLiquidInSitu'
MEAN_COEF_OF_EXPANSION_FLUID = 'MeanCoefOfExpansionFluid'
MEAN_VELOCITY_FLUID = 'MeanVelocityFluid'
MEASURED_DEPTH = 'MeasuredDepth'
MOLAR_DENSITY_STOCKTANK = 'MolarDensityStockTank'
MOLAR_VOLUME_STOCKTANK = 'MolarVolumeStockTank'
MOLECULAR_WEIGHT_FLUID = 'MolecularWeightFluid'
MOLE_FRACTION_CO = 'MoleFractionCO'
MOLE_FRACTION_CO2 = 'MoleFractionCO2'
MOLE_FRACTION_H2 = 'MoleFractionH2'
MOLE_FRACTION_H2S = 'MoleFractionH2S'
```

```
MOLE_FRACTION_N2 = 'MoleFractionN2'
MUDLINE_TEMPERATURE = 'MudlineTemperature'
NODE_NUMBER = 'NodeNumber'
OGR_STOCKTANK = 'OGRStockTank'
OIL_FORMATION_VOLUME_FACTOR = 'OilFormationVolumeFactor'
OUTSIDE_FILM_FORCED_CONVECTION_HEAT_TRANSFER_COEFFICIENT =
'OutsideFilmForcedConvectionHeatTransferCoefficient'
OUTSIDE_FILM_FORCED_CONVECTION_NUSSELT_NUMBER = 'OutsideFilmForcedConvectionNusseltNumber'
OUTSIDE_FILM_FREE_CONVECTION_HEAT_TRANSFER_COEFFICIENT =
'OutsideFilmFreeConvectionHeatTransferCoefficient'
OUTSIDE_FILM_FREE_CONVECTION_NUSSELT_NUMBER = 'OutsideFilmFreeConvectionNusseltNumber'
OUTSIDE_FILM_GRASHOF_NUMBER = 'OutsideFilmGrashofNumber'
OUTSIDE_FILM_HEAT_TRANSFER_COEFFICIENT_FOR_BURIED_PORTION =
'OutsideFilmHeatTransferCoefficientForBuriedPortion'
OUTSIDE_FILM_HEAT_TRANSFER_COEFFICIENT_FOR_EXPOSED_PORTION =
'OutsideFilmHeatTransferCoefficientForExposedPortion'
OUTSIDE_FILM_PRANDTL_NUMBER = 'OutsideFilmPrandtlNumber'
OUTSIDE_FILM_RALEIGH_NUMBER = 'OutsideFilmRaleighNumber'
OUTSIDE_FILM_REYNOLDS_NUMBER = 'OutsideFilmReynoldsNumber'
OUTSIDE_FLUID_FILM_HEAT_TRANSFER_COEFFICIENT = 'OutsideFluidFilmHeatTransferCoefficient'
OVERALL_AMBIENT_HEAT_TRANSFER_COEFFICIENT = 'OverallAmbientHeatTransferCoefficient'
OVERALL_HEAT_TRANSFER_COEFFICIENT = 'OverallHeatTransferCoefficient'
OVERALL_HEAT_TRANSFER_COEFFICIENT_OF_BURIED_PORTION =
'OverallHeatTransferCoefficientOfBuriedPortion'
OVERALL_HEAT_TRANSFER_COEFFICIENT_OF_EXPOSED_PORTION =
'OverallHeatTransferCoefficientOfExposedPortion'
PHASE_INVERSION_WATERCUT_LIQUID_INSITU = 'PhaseInversionWatercutLiquidInsitu'
PIPE_ANGLE_TO_HORIZONTAL = 'PipeAngleToHorizontal'
PIPE_COATINGS_THICKNESS = 'PipeCoatingsThickness'
PIPE_INSIDE_DIAMETER = 'PipeInsideDiameter'
PIPE_INSIDE_TEMPERATURE = 'PipeInsideTemperature'
PIPE_INSIDE_TEMPERATURE_BURIED_PART = 'PipeInsideTemperatureBuriedPart'
PIPE_INSIDE_TEMPERATURE_EXPOSED_PART = 'PipeInsideTemperatureExposedPart'
PIPE_OUTSIDE_DIAMETER = 'PipeOutsideDiameter'
PIPE_OUTSIDE_DIAMETER_WITH_COATINGS = 'PipeOutsideDiameterWithCoatings'
PIPE_OUTSIDE_TEMPERATURE = 'PipeOutsideTemperature'
```

PIPE_OUTSIDE_TEMPERATURE_BURIED_PART = 'PipeOutsideTemperatureBuriedPart'
PIPE_OUTSIDE_TEMPERATURE_EXPOSED_PART = 'PipeOutsideTemperatureExposedPart'
PIPE_WALL_CONDUCTIVITY = 'PipeWallConductivity'
PIPE_WALL_HEAT_TRANSFER_COEFFICIENT = 'PipeWallHeatTransferCoefficient'
PIPE_WALL_ROUGHNESS = 'PipeWallRoughness'
PIPE_WALL_THICKNESS = 'PipeWallThickness'
PRESSURE = 'Pressure'
PRESSURE_DROP_RATIO_ELEVATION = 'PressureDropRatioElevation'
PRESSURE_DROP_RATIO_FRICTION = 'PressureDropRatioFriction'
PRESSURE_DROP_RATIO_TOTAL = 'PressureDropRatioTotal'
PRESSURE_GRADIENT_ACCELERATION = 'PressureGradientAcceleration'
PRESSURE_GRADIENT_ELEVATION = 'PressureGradientElevation'
PRESSURE_GRADIENT_FRICTION = 'PressureGradientFriction'
PRESSURE_GRADIENT_TOTAL = 'PressureGradientTotal'
RELATIVE_PERMEABILITY_OIL = 'RelativePermeabilityOil'
RELATIVE_PERMEABILITY_WATER = 'RelativePermeabilityWater'
RESERVOIR_DRAWDOWN = 'ReservoirDrawdown'
RESERVOIR_PRESSURE = 'ReservoirPressure'
RESERVOIR_SATURATION = 'ReservoirSaturation'
RESERVOIR_SPECIFIC_GAS_INFLOW = 'ReservoirSpecificGasInflow'
RESERVOIR_SPECIFIC_MASS_INFLOW = 'ReservoirSpecificMassInflow'
RESERVOIR_TEMPERATURE = 'ReservoirTemperature'
RESISTANCE_OF_BURIED_PORTION_OVERALL = 'ResistanceOfBuriedPortionOverall'
RESISTANCE_OF_EXPOSED_PORTION_OVERALL = 'ResistanceOfExposedPortionOverall'
RESISTANCE_OF_OUTSIDE_FILM_ON_EXPOSED_PORTION = 'ResistanceOfOutsideFilmOnExposedPortion'
REYNOLDS_NUMBER = 'ReynoldsNumber'
SAND_PRODUCTION_RATE = 'SandProductionRate'
SEGMENT_ENTHALPY_GRADIENT = 'SegmentEnthalpyGradient'
SEGMENT_INFLOW_RATE_GRADIENT = 'SegmentInflowRateGradient'
SEGMENT_LENGTH = 'SegmentLength'
SEGMENT_PRESSURE_GRADIENT = 'SegmentPressureGradient'
SEGMENT_TEMPERATURE_GRADIENT = 'SegmentTemperatureGradient'
SINGLE_PHASE_ELEVATION_PRESSURE_GRADIENT = 'SinglePhaseElevationPressureGradient'
SINGLE_PHASE_FRICTIONAL_PRESSURE_GRADIENT = 'SinglePhaseFrictionalPressureGradient'
SINGLE_PHASE_TOTAL_PRESSURE_GRADIENT = 'SinglePhaseTotalPressureGradient'
SKIN_DUE_TO_COMPACTED_ZONE = 'SkinDueToCompactedZone'

SKIN_DUE_TO_DAMAGED_WELLBORE = 'SkinDueToDamagedWellbore'
SKIN_DUE_TO_GRAVEL_PACK = 'SkinDueToGravelPack'
SKIN_DUE_TO_PERFORATION_GEOMETRY = 'SkinDueToPerforationGeometry'
SKIN_FACTOR_OVERALL = 'SkinFactorOverall'
SKIN_FACTOR_RATE_DEPENDENT = 'SkinFactorRateDependent'
SKIN_TURBULENT_DUE_TO_DAMAGED_WELLBORE = 'SkinTurbulentDueToDamagedWellbore'
SKIN_TURBULENT_DUE_TO_GRAVEL_PACKING = 'SkinTurbulentDueToGravelPacking'
SKIN_TURBULENT_DUE_TO_PERFORATIONS = 'SkinTurbulentDueToPerforations'
SLIP_RATIO_GAS_LIQUID = 'SlipRatioGasLiquid'
SLIP_RATIO_OIL_WATER = 'SlipRatioOilWater'
SLUG_FREQUENCY1IN10 = 'SlugFrequency1In10'
SLUG_FREQUENCY1IN100 = 'SlugFrequency1In100'
SLUG_FREQUENCY1IN1000 = 'SlugFrequency1In1000'
SLUG_FREQUENCY_MEAN = 'SlugFrequencyMean'
SLUG_LENGTH1IN10 = 'SlugLength1In10'
SLUG_LENGTH1IN100 = 'SlugLength1In100'
SLUG_LENGTH1IN1000 = 'SlugLength1In1000'
SLUG_LENGTH_MEAN = 'SlugLengthMean'
SLUG_VOLUME1IN10 = 'SlugVolume1In10'
SLUG_VOLUME1IN100 = 'SlugVolume1In100'
SLUG_VOLUME1IN1000 = 'SlugVolume1In1000'
SLUG_VOLUME_MEAN = 'SlugVolumeMean'
SOLUTION_GAS_IN_OIL_INSITU = 'SolutionGasInOilInSitu'
SOLUTION_GAS_IN_WATER_INSITU = 'SolutionGasInWaterInSitu'
SOLUTION_GAS_POTENTIAL_IN_LIQUID_INSITU = 'SolutionGasPotentialInLiquidInSitu'
SOLUTION_GAS_POTENTIAL_IN_OIL_INSITU = 'SolutionGasPotentialInOilInSitu'
SOLUTION_GAS_POTENTIAL_IN_WATER_INSITU = 'SolutionGasPotentialInWaterInSitu'
SOLUTION_GAS_VOLUME_FLOWRATE_INSITU = 'SolutionGasVolumeFlowrateInSitu'
SONIC_VELOCITY_IN_FLUID = 'SonicVelocityInFluid'
SPECIFIC_GRAVITY_GAS_INSITU = 'SpecificGravityGasInSitu'
SPECIFIC_GRAVITY_GAS_STOCKTANK = 'SpecificGravityGasStockTank'
SPECIFIC_GRAVITY_LIQUID_STOCKTANK = 'SpecificGravityLiquidStockTank'
SPECIFIC_GRAVITY_OIL_STOCKTANK = 'SpecificGravityOilStockTank'
SPECIFIC_GRAVITY_WATER_STOCKTANK = 'SpecificGravityWaterStockTank'
SPECIFIC_HEAT_CAPACITY_RATIO_GAS_INSITU = 'SpecificHeatCapacityRatioGasInSitu'
SPHERE_GENERATED_LIQUID_VOLUME_DUMPING_TIME_TOTAL =

`'SphereGeneratedLiquidVolumeDumpingTimeTotal'`

`SPHERE_GENERATED_LIQUID_VOLUME_FROM_SECTION = 'SphereGeneratedLiquidVolumeFromSection'`

`SPHERE_GENERATED_LIQUID_VOLUME_SO_FAR = 'SphereGeneratedLiquidVolumeSoFar'`

`SPHERE_TRANSIT_TIME_FROM_SECTION = 'SphereTransitTimeFromSection'`

`SUPERFICIAL_VELOCITY_GAS = 'SuperficialVelocityGas'`

`SUPERFICIAL_VELOCITY_LIQUID = 'SuperficialVelocityLiquid'`

`SUPERFICIAL_VELOCITY_OIL = 'SuperficialVelocityOil'`

`SUPERFICIAL_VELOCITY_WATER = 'SuperficialVelocityWater'`

`SURFACE_TENSION_LIQUID_INSITU = 'SurfaceTensionLiquidInSitu'`

`SURFACE_TENSION_OIL_GAS_INSITU = 'SurfaceTensionOilGasInSitu'`

`SURFACE_TENSION_OIL_WATER_INSITU = 'SurfaceTensionOilWaterInSitu'`

`SURFACE_TENSION_WATER_GAS_INSITU = 'SurfaceTensionWaterGasInSitu'`

`TEMPERATURE = 'Temperature'`

`TEMPERATURE_FLUID = 'TemperatureFluid'`

`TEMPERATURE_GRADIENT_ELEVATION = 'TemperatureGradientElevation'`

`TEMPERATURE_GRADIENT_GROUND_AND_AMBIENT = 'TemperatureGradientGroundAndAmbient'`

`TEMPERATURE_GRADIENT_HEAT_TRANSFER = 'TemperatureGradientHeatTransfer'`

`TEMPERATURE_GRADIENT_INSIDE_FILM = 'TemperatureGradientInsideFilm'`

`TEMPERATURE_GRADIENT_JOULE_THOMSON = 'TemperatureGradientJouleThomson'`

`TEMPERATURE_GRADIENT_OVERALL = 'TemperatureGradientOverall'`

`TEMPERATURE_GRADIENT_PIPE_AND_COATINGS = 'TemperatureGradientPipeAndCoatings'`

`TOTAL_DISTANCE = 'TotalDistance'`

`TOTAL_SPHERE_TRANSIT_TIME_SO_FAR = 'TotalSphereTransitTimeSoFar'`

`TRUE_VERTICAL_DEPTH = 'TrueVerticalDepth'`

`VELOCITY_GAS = 'VelocityGas'`

`VELOCITY_LIQUID = 'VelocityLiquid'`

`VELOCITY_OIL = 'VelocityOil'`

`VELOCITY_WATER = 'VelocityWater'`

`VISCOSITY_DEAD_OIL_STOCKTANK = 'ViscosityDeadOilStockTank'`

`VISCOSITY_FLUID_NO_SLIP_INSITU = 'ViscosityFluidNoSlipInSitu'`

`VISCOSITY_FLUID_SLIP_INSITU = 'ViscosityFluidSlipInSitu'`

`VISCOSITY_GAS_INSITU = 'ViscosityGasInSitu'`

`VISCOSITY_LIQUID_INSITU = 'ViscosityLiquidInSitu'`

`VISCOSITY_OIL_INSITU = 'ViscosityOilInSitu'`

`VISCOSITY_WATER_INSITU = 'ViscosityWaterInSitu'`

`VOLUME_FLOWRATE_FLUID_INSITU = 'VolumeFlowrateFluidInSitu'`

```

VOLUME_FLOWRATE_GAS_INSITU = 'VolumeFlowrateGasInSitu'
VOLUME_FLOWRATE_GAS_INSITU_IDEAL = 'VolumeFlowrateGasInsituIdeal'
VOLUME_FLOWRATE_GAS_STOCKTANK = 'VolumeFlowrateGasStockTank'
VOLUME_FLOWRATE_LIQUID_INSITU = 'VolumeFlowrateLiquidInSitu'
VOLUME_FLOWRATE_LIQUID_STOCKTANK = 'VolumeFlowrateLiquidStockTank'
VOLUME_FLOWRATE_OIL_INSITU = 'VolumeFlowrateOilInSitu'
VOLUME_FLOWRATE_OIL_STOCKTANK = 'VolumeFlowrateOilStockTank'
VOLUME_FLOWRATE_WATER_INSITU = 'VolumeFlowrateWaterInSitu'
VOLUME_FLOWRATE_WATER_STOCKTANK = 'VolumeFlowrateWaterStockTank'
VOLUME_FRACTION_GAS_INSITU = 'VolumeFractionGasInSitu'
VOLUME_FRACTION_LIQUID = 'VolumeFractionLiquid'
VOLUME_FRACTION_LIQUID_STOCKTANK = 'VolumeFractionLiquidStockTank'
WATERCUT = 'Watercut'
WATER_CUT_STOCKTANK = 'WaterCutStockTank'
WAX_FORMATION_TEMPERATURE = 'WaxFormationTemperature'
WAX_SUB_COOLING_DELTA_TEMPERATURE = 'WaxSubCoolingDeltaTemperature'
WGR_STOCKTANK = 'WGRStockTank'
Z_FACTOR_GAS_INSITU = 'ZFactorGasInSitu'

```

```
class sixgill.definitions.OutputVariables
```

```
    Bases: object
```

```
    class System
```

```
        Bases: object
```

```

        GAS_FIELD = ['PipeOutsideDiameter', 'SystemPressureLoss', 'SystemInletPressure',
                     'SystemOutletPressure', 'SystemOutletTemperature', 'TotalLiquidHoldup', 'InletVolume-
                     FlowrateLiquid', 'InletVolumeFlowrateGas', 'InletVelocityFluid', 'OutletVolumeFlowrateLiq-
                     uid', 'OutletVolumeFlowrateGas', 'OutletVelocityFluid', 'OutletVolumeFlowrateLiquidStock-
                     Tank', 'OutletGLRStockTank', 'OutletMassFlowrateFluid', 'CaseNumber', 'InletWaterCut-
                     StockTank', 'InputTubingHeadPressure', 'InputFlowrate', 'TotalInjectionGas', 'GasLift-
                     InjectionRatio', 'GasLiftInjectionCasingPressure', 'GasLiftInjectionCasingTemperature',
                     'GasLiftInjectionPortDiameter', 'GasLiftInjectionSourcePressure', 'GasLiftInjectionCasing-
                     HeadTemperature', 'PCPIntakePressure', 'PCPIntakeVolumetricFlowrateFluid', 'PCPEffi-
                     ciency', 'PCPPower', 'PcpTorque', 'PCPSpeed', 'PCPDischargePressure', 'RodPumpIntake-
                     Pressure', 'RodPumpIntakeVolumetricFlowrateFluid', 'RodPumpIntakeVolumeFractionGas',
                     'RodPumpIntakeVolumeFlowrateLiquid', 'RodPumpIntakeVolumeFlowrateFreeGas', 'Rod-
                     PumpDischargePressure', 'RodPumpPower', 'RodPumpDeltaPressure', 'RodPumpEfficiency',
                     'ESPPower', 'ESPHead', 'ESPEfficiency', 'ESPNumberOfStages', 'ESPIntakeGasVol-
                     umeFraction', 'ESPIntakePressure', 'ESPIntakeTotalVolumetricFlowrate', 'ESPSuction-
                     GasVolumeFraction', 'ESPDischargePressure', 'ESPDeltaPressure', 'ESPPower', 'ESPNum-
                     berOfStages', 'ESPHead', 'ESPDeltaTemperature', 'ESPEfficiency', 'ESPEfficiencyFactor',
                     'ESPFlowrateFactor', 'ESPHeadFactor', 'ESPPowerFactor', 'BottomHolePressure', 'Input-
                     Watercut', 'InletVolumeFlowrateLiquidStockTank', 'InletMassFlowrateFluid', 'WellheadPres-
                     sure', 'WellheadVolumeFlowrateLiquidStockTank', 'WellheadVolumeFlowrateLiquidInSitu',
                     'WellheadVolumeFlowrateGasStockTank', 'WellheadMassFlowrateFluid', 'NodalPointVol-
                     umeFlowrateLiquidStockTank', 'NodalPointPressure', 'NodalPointMassFlowrateFluid',

```

‘TotalPressureDropElevational’, ‘TotalPressureDropFrictional’, ‘OutletVolumeFlowrateOilStockTank’, ‘OutletMassFlowrateOilStockTank’, ‘OutletVolumeFlowrateWaterStockTank’, ‘OutletMassFlowrateWaterStockTank’, ‘OutletVolumeFlowrateGasStockTank’, ‘OutletMassFlowrateGasStockTank’, ‘NodalPointVolumeFlowrateGasStockTank’, ‘InletVolumeFlowrateGasStockTank’, ‘SevereSluggingIndicator’, ‘SlugVolumeMean’, ‘SlugLengthMean’, ‘SlugFrequencyMean’, ‘ErosionalVelocityMaximum’, ‘MaximumErosionalVelocityRatio’, ‘ESPFrequency’, ‘SphereGeneratedLiquidVolume’, ‘InletMassFractionGas’, ‘TotalOilHoldup’, ‘TotalWaterHoldup’, ‘SystemTemperatureDifference’, ‘OutletWaterCutStockTank’, ‘OutletGORStockTank’, ‘OutletFractionCO2’, ‘OutletFractionH2S’, ‘OutletFractionN2’, ‘OutletFractionH2S’, ‘OutletFractionCO’, ‘CaseStatus’, ‘MaximumVelocityGas’, ‘InletErosionalVelocityRatio’, ‘OutletErosionalVelocityRatio’, ‘MaximumHydrateSubCoolingTemperatureDifference’, ‘MaximumErosionRate’, ‘MaximumCorrosionRate’, ‘MaximumLiquidLoadingVelocityRatio’, ‘MaximumLiquidLoadingGasRate’, ‘NodalPointBubblePointPressure’]

WELL_PERFORMANCE = [‘PipeOutsideDiameter’, ‘SystemPressureLoss’, ‘SystemInletPressure’, ‘SystemOutletPressure’, ‘SystemOutletTemperature’, ‘TotalLiquidHoldup’, ‘InletVolumeFlowrateLiquid’, ‘InletVolumeFlowrateGas’, ‘InletVelocityFluid’, ‘OutletVolumeFlowrateLiquid’, ‘OutletVolumeFlowrateGas’, ‘OutletVelocityFluid’, ‘OutletVolumeFlowrateLiquidStockTank’, ‘OutletGLRStockTank’, ‘OutletMassFlowrateFluid’, ‘CaseNumber’, ‘InletWaterCutStockTank’, ‘InputTubingHeadPressure’, ‘InputFlowrate’, ‘TotalInjectionGas’, ‘GasLiftInjectionRatio’, ‘GasLiftInjectionCasingPressure’, ‘GasLiftInjectionCasingTemperature’, ‘GasLiftInjectionPortDiameter’, ‘GasLiftInjectionSourcePressure’, ‘GasLiftInjectionCasingHeadTemperature’, ‘PCPIntakePressure’, ‘PCPIntakeVolumetricFlowrateFluid’, ‘PCPEfficiency’, ‘PCPPower’, ‘PcpTorque’, ‘PCPSpeed’, ‘PCPDischargePressure’, ‘RodPumpIntakePressure’, ‘RodPumpIntakeVolumetricFlowrateFluid’, ‘RodPumpIntakeVolumeFractionGas’, ‘RodPumpIntakeVolumeFlowrateLiquid’, ‘RodPumpIntakeVolumeFlowrateFreeGas’, ‘RodPumpDischargePressure’, ‘RodPumpPower’, ‘RodPumpDeltaPressure’, ‘RodPumpEfficiency’, ‘ESPPower’, ‘ESPHead’, ‘ESPEfficiency’, ‘ESPNumberOfStages’, ‘ESPIntakeGasVolumeFraction’, ‘ESPIntakePressure’, ‘ESPIntakeTotalVolumetricFlowrate’, ‘ESPSuctionGasVolumeFraction’, ‘ESPDDischargePressure’, ‘ESPDDeltaPressure’, ‘ESPPower’, ‘ESPNumberOfStages’, ‘ESPHead’, ‘ESPDDeltaTemperature’, ‘ESPEfficiency’, ‘ESPEfficiencyFactor’, ‘ESPFlowrateFactor’, ‘ESPHeadFactor’, ‘ESPPowerFactor’, ‘BottomHolePressure’, ‘InputWatercut’, ‘InletVolumeFlowrateLiquidStockTank’, ‘InletMassFlowrateFluid’, ‘WellheadPressure’, ‘WellheadVolumeFlowrateLiquidStockTank’, ‘WellheadVolumeFlowrateLiquidInSitu’, ‘WellheadVolumeFlowrateGasStockTank’, ‘WellheadMassFlowrateFluid’, ‘NodalPointVolumeFlowrateLiquidStockTank’, ‘NodalPointPressure’, ‘NodalPointVolumeFlowrateLiquidInSitu’, ‘NodalPointVolumeFlowrateGasInSitu’, ‘NodalPointMassFlowrateFluid’, ‘HeelReservoirDrawdown’, ‘TotalPressureDropElevational’, ‘TotalPressureDropFrictional’, ‘OutletVolumeFlowrateOilStockTank’, ‘OutletMassFlowrateOilStockTank’, ‘OutletVolumeFlowrateWaterStockTank’, ‘OutletMassFlowrateWaterStockTank’, ‘OutletVolumeFlowrateGasStockTank’, ‘OutletMassFlowrateGasStockTank’, ‘WellheadVolumeFlowrateOilStockTank’, ‘NodalPointVolumeFlowrateOilStockTank’, ‘WellheadVolumeFlowrateGasStockTank’, ‘NodalPointVolumeFlowrateGasStockTank’, ‘InletVolumeFlowrateGasStockTank’, ‘SevereSluggingIndicator’, ‘SlugVolumeMean’, ‘SlugLengthMean’, ‘SlugFrequencyMean’, ‘ErosionalVelocityMaximum’, ‘MaximumErosionalVelocityRatio’, ‘ESPFrequency’, ‘SphereGeneratedLiquidVolume’, ‘WellheadTemperature’, ‘PressureDropTotalCompletion’, ‘InletMassFractionGas’, ‘SystemTemperatureDifference’, ‘OutletWaterCutStockTank’, ‘OutletGORStockTank’, ‘CaseStatus’, ‘MaximumErosionRate’, ‘MaximumCorrosionRate’, ‘MaximumLiquidLoadingVelocityRatio’, ‘MaximumLiquidLoadingGasRate’, ‘NodalPointBubblePointPressure’]

FLOW_ASSURANCE = [‘PipeOutsideDiameter’, ‘SystemPressureLoss’, ‘SystemInletPressure’, ‘SystemOutletPressure’, ‘SystemOutletTemperature’, ‘TotalLiquidHoldup’, ‘InletVolumeFlowrateLiquid’, ‘InletVolumeFlowrateGas’, ‘InletVelocityFluid’, ‘OutletVolumeFlowrateLiquid’, ‘OutletVolumeFlowrateGas’, ‘OutletVelocityFluid’, ‘HeatTransferCoefficient’, ‘Out-

letVolumeFlowrateLiquidStockTank', 'OutletGLRStockTank', 'OutletMassFlowrateFluid', 'CaseNumber', 'InletWaterCutStockTank', 'InputTubingHeadPressure', 'InputFlowrate', 'TotalInjectionGas', 'GasLiftInjectionRatio', 'GasLiftInjectionCasingPressure', 'GasLiftInjectionCasingTemperature', 'GasLiftInjectionPortDiameter', 'GasLiftInjectionSourcePressure', 'GasLiftInjectionCasingHeadTemperature', 'PCPIntakePressure', 'PCPIntakeVolumetricFlowrateFluid', 'PCPEfficiency', 'PCPPower', 'PcpTorque', 'PCPSpeed', 'PCPDischargePressure', 'RodPumpIntakePressure', 'RodPumpIntakeVolumetricFlowrateFluid', 'RodPumpIntakeVolumeFractionGas', 'RodPumpIntakeVolumeFlowrateLiquid', 'RodPumpIntakeVolumeFlowrateFreeGas', 'RodPumpDischargePressure', 'RodPumpPower', 'RodPumpDeltaPressure', 'RodPumpEfficiency', 'ESPPower', 'ESPHead', 'ESPEfficiency', 'ESPNumberOfStages', 'ESPIntakeGasVolumeFraction', 'ESPIntakePressure', 'ESPIntakeTotalVolumetricFlowrate', 'ESPSuctionGasVolumeFraction', 'ESPDDischargePressure', 'ESPDDeltaPressure', 'ESPPower', 'ESPNumberOfStages', 'ESPHead', 'ESPDDeltaTemperature', 'ESPEfficiency', 'ESPEfficiencyFactor', 'ESPFlowrateFactor', 'ESPHeadFactor', 'ESPPowerFactor', 'BottomHolePressure', 'InputWatercut', 'InletVolumeFlowrateLiquidStockTank', 'InletMassFlowrateFluid', 'WellheadPressure', 'WellheadVolumeFlowrateLiquidStockTank', 'WellheadVolumeFlowrateLiquidInSitu', 'WellheadVolumeFlowrateGasStockTank', 'WellheadMassFlowrateFluid', 'NodalPointVolumeFlowrateLiquidStockTank', 'NodalPointPressure', 'NodalPointMassFlowrateFluid', 'TotalPressureDropElevational', 'TotalPressureDropFrictional', 'OutletVolumeFlowrateOilStockTank', 'OutletMassFlowrateOilStockTank', 'OutletVolumeFlowrateWaterStockTank', 'OutletMassFlowrateWaterStockTank', 'OutletVolumeFlowrateGasStockTank', 'OutletMassFlowrateGasStockTank', 'WellheadVolumeFlowrateOilStockTank', 'NodalPointVolumeFlowrateGasStockTank', 'InletVolumeFlowrateGasStockTank', 'SevereSluggingIndicator', 'SlugVolumeMean', 'SlugLengthMean', 'SlugFrequencyMean', 'SlugVolume1In1000', 'SlugLength1In1000', 'SlugFrequency1In1000', 'ErosionalVelocityMaximum', 'MaximumErosionalVelocityRatio', 'ESPFrequency', 'SphereGeneratedLiquidVolume', 'PressureDropTotalAcceleration', 'InletMassFractionGas', 'TotalOilHoldup', 'TotalWaterHoldup', 'SystemTemperatureDifference', 'MaximumHeatTransferCoefficient', 'OutletWaterCutStockTank', 'OutletGORStockTank', 'CaseStatus', 'MaximumVelocityFluid', 'MaximumVelocityLiquid', 'MaximumVelocityGas', 'MaximumHydrateSubCoolingTemperatureDifference', 'MaximumWaxSubCoolingTemperatureDifference', 'MaximumErosionRate', 'MaximumCorrosionRate', 'MaximumLiquidLoadingVelocityRatio', 'MaximumLiquidLoadingGasRate', 'NodalPointBubblePointPressure']

LARGE_NETWORK = ['PipeOutsideDiameter', 'SystemPressureLoss', 'SystemInletPressure', 'SystemOutletPressure', 'SystemOutletTemperature', 'TotalLiquidHoldup', 'InletVolumeFlowrateLiquid', 'InletVolumeFlowrateGas', 'InletVelocityFluid', 'OutletVolumeFlowrateLiquid', 'OutletVolumeFlowrateGas', 'OutletVelocityFluid', 'OutletVolumeFlowrateLiquidStockTank', 'OutletGLRStockTank', 'OutletMassFlowrateFluid', 'TotalInjectionGas', 'GasLiftInjectionRatio', 'GasLiftInjectionCasingPressure', 'GasLiftInjectionCasingTemperature', 'GasLiftInjectionPortDiameter', 'GasLiftInjectionSourcePressure', 'GasLiftInjectionCasingHeadTemperature', 'PCPIntakePressure', 'PCPIntakeVolumetricFlowrateFluid', 'PCPEfficiency', 'PCPPower', 'PcpTorque', 'PCPSpeed', 'PCPDischargePressure', 'RodPumpIntakePressure', 'RodPumpIntakeVolumetricFlowrateFluid', 'RodPumpIntakeVolumeFractionGas', 'RodPumpIntakeVolumeFlowrateLiquid', 'RodPumpIntakeVolumeFlowrateFreeGas', 'RodPumpDischargePressure', 'RodPumpPower', 'RodPumpDeltaPressure', 'RodPumpEfficiency', 'ESPPower', 'ESPHead', 'ESPEfficiency', 'ESPNumberOfStages', 'ESPIntakeGasVolumeFraction', 'ESPIntakePressure', 'ESPIntakeTotalVolumetricFlowrate', 'ESPSuctionGasVolumeFraction', 'ESPDDischargePressure', 'ESPDDeltaPressure', 'ESPPower', 'ESPNumberOfStages', 'ESPHead', 'ESPDDeltaTemperature', 'ESPEfficiency', 'ESPEfficiencyFactor', 'ESPFlowrateFactor', 'ESPHeadFactor', 'ESPPowerFactor', 'BottomHolePressure', 'InletVolumeFlowrateLiquidStockTank', 'InletMassFlowrateFluid', 'WellheadPressure', 'WellheadVolumeFlowrateLiquidStockTank', 'WellheadVolumeFlowrateLiquidInSitu', 'WellheadVolumeFlowrateGasStockTank', 'NodalPointVolumeFlowrateLiquidStockTank',

‘NodalPointPressure’, ‘NodalPointMassFlowrateFluid’, ‘TotalPressureDropElevational’, ‘TotalPressureDropFrictional’, ‘OutletVolumeFlowrateOilStockTank’, ‘OutletMassFlowrateOilStockTank’, ‘OutletVolumeFlowrateWaterStockTank’, ‘OutletMassFlowrateWaterStockTank’, ‘OutletVolumeFlowrateGasStockTank’, ‘OutletMassFlowrateGasStockTank’, ‘WellheadVolumeFlowrateOilStockTank’, ‘NodalPointVolumeFlowrateGasStockTank’, ‘InletVolumeFlowrateGasStockTank’, ‘SevereSluggingIndicator’, ‘SlugVolumeMean’, ‘SlugLengthMean’, ‘SlugFrequencyMean’, ‘ErosionalVelocityMaximum’, ‘MaximumErosionalVelocityRatio’, ‘ESPFrequency’, ‘SphereGeneratedLiquidVolume’, ‘InletMassFractionGas’, ‘SystemTemperatureDifference’, ‘OutletWaterCutStockTank’, ‘OutletGORStockTank’, ‘MaximumErosionRate’, ‘MaximumCorrosionRate’, ‘MaximumLiquidLoadingVelocityRatio’, ‘MaximumLiquidLoadingGasRate’, ‘NodalPointBubblePointPressure’]

```
class OutputVariables.Profile
```

```
Bases: object
```

```
GAS_FIELD = ['HorizontalDistance', 'TotalDistance', 'Elevation', 'Pressure', 'Temperature', 'HoldupFractionLiquid', 'VelocityLiquid', 'VelocityGas', 'MeanVelocityFluid', 'NodeNumber', 'HydrostaticHead', 'PipeInsideDiameter', 'EnthalpyFluid', 'Watercut', 'VolumeFractionLiquid', 'TrueVerticalDepth', 'MeasuredDepth', 'FlowrateLiquidInSitu', 'FlowrateGasInSitu', 'PressureGradientTotal', 'AmbientTemperatureAtNode', 'PressureGradientFriction', 'PressureGradientElevation', 'PressureGradientAcceleration', 'VolumeFlowrateGasStockTank', 'VolumeFlowrateLiquidStockTank', 'VolumeFlowrateOilStockTank', 'GLRStockTank', 'LGRStockTank', 'MassFlowrate', 'VolumeFlowrateGasInSitu', 'VolumeFlowrateLiquidInSitu', 'VolumeFlowrateOilInSitu', 'VolumeFractionGasInSitu', 'SpecificGravityGasInSitu', 'DensityGasInSitu', 'ViscosityGasInSitu', 'ViscosityLiquidInSitu', 'EntropyFluidInSitu', 'ZFactorGasInSitu', 'SlugVolumeMean', 'SlugLengthMean', 'SlugFrequencyMean', 'ErosionalVelocity', 'ErosionalVelocityRatio', 'FlowPatternGasLiquid', 'FlowPatternOilWater', 'SuperficialVelocityLiquid', 'SuperficialVelocityGas', 'HorizontalPosition', 'ErosionRate', 'CorrosionRate', 'HydrateFormationTemperature', 'HydrateSubCoolingDeltaTemperature', 'LiquidLoadingVelocity', 'LiquidLoadingVelocityRatio', 'LiquidLoadingGasRate']
```

```
WELL_PERFORMANCE = ['HorizontalDistance', 'TotalDistance', 'Elevation', 'Pressure', 'Temperature', 'HoldupFractionLiquid', 'VelocityLiquid', 'VelocityGas', 'MeanVelocityFluid', 'NodeNumber', 'HydrostaticHead', 'PipeInsideDiameter', 'Watercut', 'VolumeFractionLiquid', 'TrueVerticalDepth', 'MeasuredDepth', 'FlowrateLiquidInSitu', 'FlowrateGasInSitu', 'ReservoirDrawdown', 'PressureGradientTotal', 'LayerStaticPressure', 'OilFormationVolumeFactor', 'AmbientTemperatureAtNode', 'SkinTurbulentDueToDamagedWellbore', 'SkinFactorOverall', 'SkinFactorRateDependent', 'SkinDueToDamagedWellbore', 'SkinDueToPerforationGeometry', 'SkinDueToCompactedZone', 'SkinDueToGravelPack', 'SkinTurbulentDueToPerforations', 'SkinTurbulentDueToGravelPacking', 'EffectivePermeability', 'PressureGradientFriction', 'PressureGradientElevation', 'PressureGradientAcceleration', 'VolumeFlowrateGasStockTank', 'VolumeFlowrateLiquidStockTank', 'VolumeFlowrateOilStockTank', 'GLRStockTank', 'LGRStockTank', 'MassFlowrate', 'VolumeFlowrateGasInSitu', 'VolumeFlowrateLiquidInSitu', 'VolumeFlowrateOilInSitu', 'VolumeFlowrateWaterInSitu', 'VolumeFractionGasInSitu', 'BubblePointPressureInSitu', 'DensityLiquidInSitu', 'DensityOilInSitu', 'DensityWaterInSitu', 'ViscosityGasInSitu', 'ViscosityLiquidInSitu', 'ErosionalVelocity', 'ErosionalVelocityRatio', 'HeatCapacityGasInSitu', 'HeatCapacityLiquidInSitu', 'FlowPatternGasLiquid', 'FlowPatternOilWater', 'SuperficialVelocityLiquid', 'SuperficialVelocityGas', 'HorizontalPosition', 'ErosionRate', 'CorrosionRate', 'LiquidLoadingVelocity', 'LiquidLoadingVelocityRatio', 'LiquidLoadingGasRate', 'SpecificHeatCapacityRatioGasInSitu']
```

```
FLOW_ASSURANCE = ['HorizontalDistance', 'TotalDistance', 'Elevation', 'Pressure', 'Temperature', 'HoldupFractionLiquid', 'VelocityLiquid', 'VelocityGas', 'MeanVelocityFluid', 'NodeNumber', 'HydrostaticHead', 'PipeInsideDiameter', 'EnthalpyFluid', 'Watercut', 'VolumeFractionLiquid', 'TrueVerticalDepth', 'MeasuredDepth', 'FlowrateLiquidInSitu', 'FlowrateGasInSitu', 'PressureGradientTotal', 'AmbientTemperatureAtNode', 'Pressure-
```

GradientFriction', 'PressureGradientElevation', 'PressureGradientAcceleration', 'ReynoldsNumber', 'VolumeFlowrateGasStockTank', 'VolumeFlowrateLiquidStockTank', 'VolumeFlowrateOilStockTank', 'GLRStockTank', 'LGRStockTank', 'MassFlowrate', 'VolumeFlowrateGasInSitu', 'VolumeFlowrateLiquidInSitu', 'VolumeFlowrateOilInSitu', 'VolumeFlowrateWaterInSitu', 'VolumeFractionGasInSitu', 'BubblePointPressureInSitu', 'DensityGasInSitu', 'DensityLiquidInSitu', 'DensityOilInSitu', 'DensityWaterInSitu', 'DensityFluidNoSlipInSitu', 'DensityFluidInSitu', 'ViscosityGasInSitu', 'ViscosityLiquidInSitu', 'EntropyFluidInSitu', 'SlugVolumeMean', 'SlugLengthMean', 'SlugFrequencyMean', 'SlugVolume1In1000', 'SlugLength1In1000', 'SlugFrequency1In1000', 'ErosionalVelocity', 'ErosionalVelocityRatio', 'OverallHeatTransferCoefficient', 'InsideFluidFilmHeatTransferCoefficient', 'HeatCapacityGasInSitu', 'HeatCapacityLiquidInSitu', 'FlowPatternGasLiquid', 'FlowPatternOilWater', 'SuperficialVelocityLiquid', 'SuperficialVelocityGas', 'HorizontalPosition', 'ErosionRate', 'CorrosionRate', 'HydrateFormationTemperature', 'HydrateSubCoolingDeltaTemperature', 'WaxFormationTemperature', 'WaxSubCoolingDeltaTemperature', 'AsphalteneFormationTemperature', 'LiquidLoadingVelocity', 'LiquidLoadingVelocityRatio', 'LiquidLoadingGasRate', 'TemperatureGradientElevation', 'TemperatureGradientJouleThomson', 'TemperatureGradientHeatTransfer', 'TemperatureGradientOverall', 'TemperatureGradientInsideFilm', 'TemperatureGradientPipeAndCoatings', 'TemperatureGradientGroundAndAmbient', 'SpecificHeatCapacityRatioGasInSitu']

LARGE_NETWORK = ['HorizontalDistance', 'TotalDistance', 'Elevation', 'Pressure', 'Temperature', 'HoldupFractionLiquid', 'VelocityLiquid', 'VelocityGas', 'MeanVelocityFluid', 'HydrostaticHead', 'PipeInsideDiameter', 'Watercut', 'VolumeFractionLiquid', 'FlowrateLiquidInSitu', 'FlowrateGasInSitu', 'PressureGradientTotal', 'AmbientTemperatureAtNode', 'PressureGradientFriction', 'PressureGradientElevation', 'PressureGradientAcceleration', 'VolumeFlowrateGasStockTank', 'VolumeFlowrateLiquidStockTank', 'VolumeFlowrateOilStockTank', 'GLRStockTank', 'LGRStockTank', 'MassFlowrate', 'VolumeFlowrateGasInSitu', 'VolumeFlowrateLiquidInSitu', 'VolumeFlowrateOilInSitu', 'VolumeFlowrateWaterInSitu', 'VolumeFractionGasInSitu', 'DensityLiquidInSitu', 'DensityOilInSitu', 'DensityWaterInSitu', 'ViscosityGasInSitu', 'ViscosityLiquidInSitu', 'SlugVolumeMean', 'SlugLengthMean', 'SlugFrequencyMean', 'ErosionalVelocity', 'ErosionalVelocityRatio', 'HeatCapacityGasInSitu', 'HeatCapacityLiquidInSitu', 'FlowPatternGasLiquid', 'FlowPatternOilWater', 'HorizontalPosition', 'ErosionRate', 'CorrosionRate', 'LiquidLoadingVelocity', 'LiquidLoadingVelocityRatio', 'LiquidLoadingGasRate', 'SpecificHeatCapacityRatioGasInSitu']

11.2 API Documentation

- [enpyxll](#)
- [pyxll](#)
- [xlwings](#)