Table of Contents

3. Streamz Reference Manual
4. Streamz Tabulation Reference
5. Psm Gui Manual
6. Psm Toolkit Manual

Copyright Notice

Disclaimer

Getting Started

This document gives a brief introduction to the use of **Petrostreamz Pipe-It** (referred to as simply **Pipe-It** henceforth) and is meant to be a guide for understanding and using a project built by Petrostreamz AS for clients.

The Petrostreamz Support department can arrange necessary training courses, short video tutorials or provide on-site support/consultancy services. Please contact support@petrostreamz.com .

Pipe-It Advantages

Petrostreamz Resource Management is a new methodology for asset optimization and exploitation where petroleum streams are optimized over the whole value chain of, and within each, petroleum sector for:

- maximum product value
- better decisions for investments and technical solutions
- cost reductions

Petroleum resources originate as static bodies of oil and gas in underground reservoirs. As these resources are exploited, they flow as petroleum streams into a production system. Streams are mixed and processed, splitting into new streams. Mixing and splitting of streams occurs throughout a labyrinth of wells, surface equipments, pipelines, storage facilities, transportation … with new mixing and products being generated at the final refinery destination. With *Pipe-It* users can manage their petroleum streams and optimize the production of hydrocarbons.

Pipe-It has the ability to keep a detailed, quantitative upstream-to-downstream accounting of the individual components making up petroleum resources. This bridges the work of reservoir engineers, who talks in terms of volumetric rates, to the processing engineers, who talk in terms of component molar rates, and ultimately to managers who talk in terms of currency, revenue, net present value and profit.

With *Pipe-It* software, a petroleum resource stream can be described simply as crude oil and gas, or it can be quantified with the hundreds of molecular constituents (methane, propane, …, benzene, carbon dioxide, …). It also allows for every conceivable description between simple and complex – LPG, NGL, GTL products, crude oil refinery distillation cuts, wax, asphaltenes, and mercaptons. Even value in any currency instead of barrels.

Pipe-It allows a project to be visualized with an intuitive graphical layout design. One visualization can be created to give a clear vision of the project from a top-level management point-of-view, showing how fields are connected and interact at major process facilities and export terminals. Other visualizations detail sub-projects like a subsea well template or a satellite process facility, always maintaining the actual connectivity to other elements in the project. Custom visualizations of the same Project can be made by each individual or team working on a Project.

Pipe-It allows an entire project or elements of a project to be executed (run). This might involve executing external computer programs describing performance of reservoirs, wells, pipelines, process facilities, storage and transportation vessels, and final-destination refining. Pipe-It allows these 3rd-party programs to communicate using Pipe-It's engine.

Not only does *Pipe-It* rely on calculation programs for resource streams, it allows access to measured data provided by online metering and spot testing stored in databases. Sophisticated data management features of the Streamz engine allows measured and calculated data to be compared on a consistent basis in terms of quantity (mass, volume, etc.) and time period (daily, monthly, annual). Once on a common basis comparative stream data can be used as the basis for optimization.

A unique technology called Linkz allows Pipe-It to "connect" to tokens (numbers and strings) in text files irrespective of formats. These tokens are then available to Pipe-It's execution and optimization engines and become part of the overall optimization. This ability together with the possibility of executing any program that can be launched from the OS gives Pipe-It an unprecedented capability and flexibility making it an ideal framework for integrated computational integration. No programs are imposed on the end user (they choose which programs are best suited for their area of expertise), but rather Pipe-It accepts all and only facilitates their integration (connected execution) and data-exchange.

Pipe-It's Optimization Interface allows (through Linkz) any number in any file to be variables, constraints and/or objectives of an optimization. Multiple levels of optimization can be defined and be active in a single top level global optimization. Different solvers and directions of optimization (MAX, MIN) can be specified for each instance of the global or local optimization. A set of robust solvers are provided, which can be extended by means of documented API.

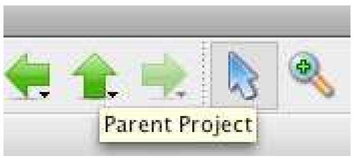
Pipe-It includes a command-line version that can be called by not only other programs (even batch files) but also by itself! This allows Pipe-It to be embedded within any other software that can launch an OS command-line. This also means Pipe-It can recursively call many instances of itself from within a top-level graphical instance. This is the basis of multi-level local and global optimization made possible by Pipe-It.

Help and Documentation

There are several ways to get help in Pipe-It depending on the level of help required.

Tooltips

Hypertext – short description of objects in the user interface. The description usually appears when the cursor is placed over an object.

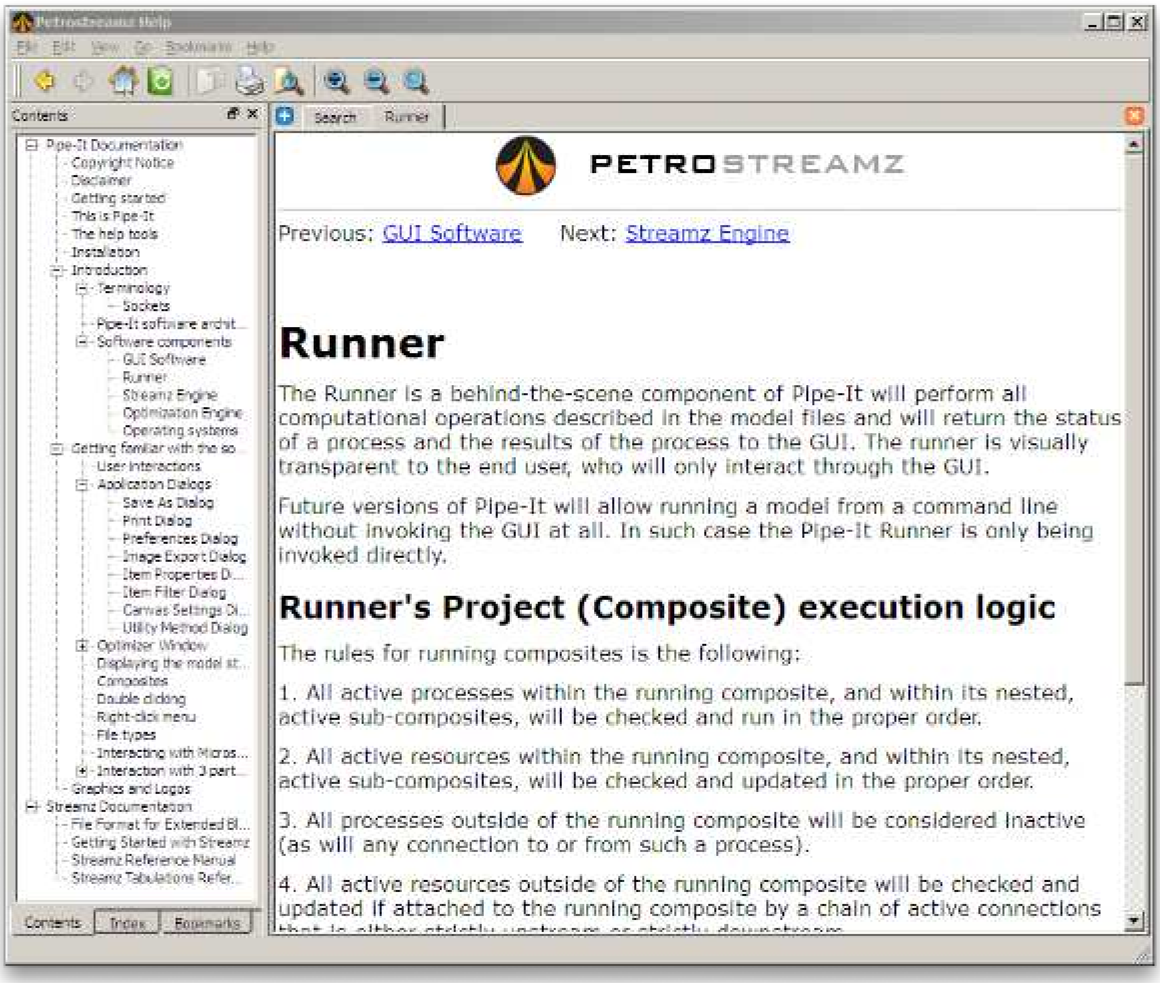


Help buttons

Each Pipe-It dialog normally has a bottom-left button (with a Question marks "?") designed to invoke Petrostreamz Help system page that explains all properties actions and information related to current dialog.

Petrostreamz Help System

When user press Help button or invokes Menu -> Help -> Help, the Table of Contents or appropriate Help page is opened.



- Pipe-It Documentation
  - Copyright Notice
  - Disclaimer
  - Getting started
  - This is Pipe-It
  - The help tools
  - Installation
  - Introduction
    - Terminology
      - Sockets
    - Pipe-It software archit...
    - Software components
      - GUI Software
      - Runner
      - Streamz Engine
      - Optimization Engine
      - Operating systems
  - Getting familiar with the so...
    - User interactions
    - Application Dialogs
      - Save As Dialog
      - Print Dialog
      - Preferences Dialog
      - Image Export Dialog
      - Item Properties D...
      - Item Filter Dialog
      - Canvas Settings Di...
      - Utility Method Dialog
    - Optimizer Window
    - Displaying the model st...
    - Composites
    - Double clicking
    - Right-click menu
    - File types
    - Interacting with Micros...
    - Interaction with 3 part...
  - Graphics and Logos
- Streamz Documentation
  - File Format for Extended Bl...
  - Getting Started with Streamz
  - Streamz Reference Manual
  - Streamz Tabulations Refer...

Previous: GUI Software Next: Streamz Engine

Runner

The Runner is a behind-the-scene component of Pipe-It will perform all computational operations described in the model files and will return the status of a process and the results of the process to the GUI. The runner is visually transparent to the end user, who will only interact through the GUI.

Future versions of Pipe-It will allow running a model from a command line without invoking the GUI at all. In such case the Pipe-It Runner is only being invoked directly.

Runner's Project (Composite) execution logic

The rules for running composites is the following:

1. All active processes within the running composite, and within its nested, active sub-composites, will be checked and run in the proper order.

2. All active resources within the running composite, and within its nested, active sub-composites, will be checked and updated in the proper order.

3. All processes outside of the running composite will be considered inactive (as will any connection to or from such a process).

4. All active resources outside of the running composite will be checked and updated if attached to the running composite by a chain of active connections

The format of the Help is similar to a web browser that shows hypertext content with links and images. Through this help the user has have access to Table of Content that includes Pipe-It and Streamz documentation, panel with list of keywords, bookmarks panel and search system that allows you easily find content that you are interested in.

Support by email

Sending an e-mail - to the Support Department at support@petrostreamz.com is the highest level of help and can be used to resolve intricate problems when you are otherwise stuck.

Installation

Windows Operating System

Pipe-It is distributed as a packed installation executable that is an application installation and configuration service. This binary is an engine for the installation, maintenance, and removal of Petrostreamz software on modern Microsoft

Windows systems.

On other systems it may use other installation procedures. Installation packages are available for Mac OSX systems and for Linux systems (on request).

Quick Installation Guide

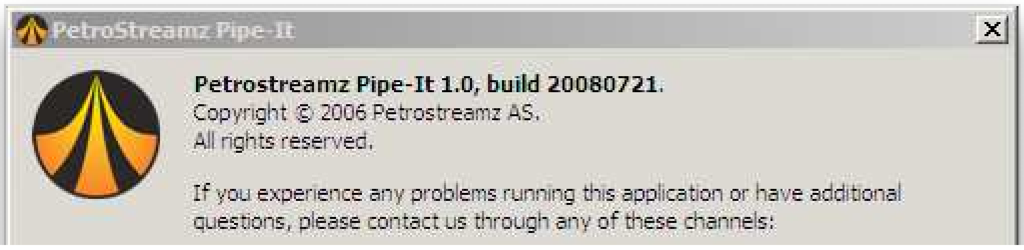
Here is explanation of installation procedure that allows you quickly install Pipe-It application on your desktop system.

File Naming and Versioning

Petrostreamz AS provides installation binaries with following naming conventions: Example:

```
    PipeItSetup-20080721.exe
```

Where binary gets suffix that indicating date of build. This allows you to keep track of build number and have up-to-dated versions. You can easily compare this build ID with the indication in About dialog:



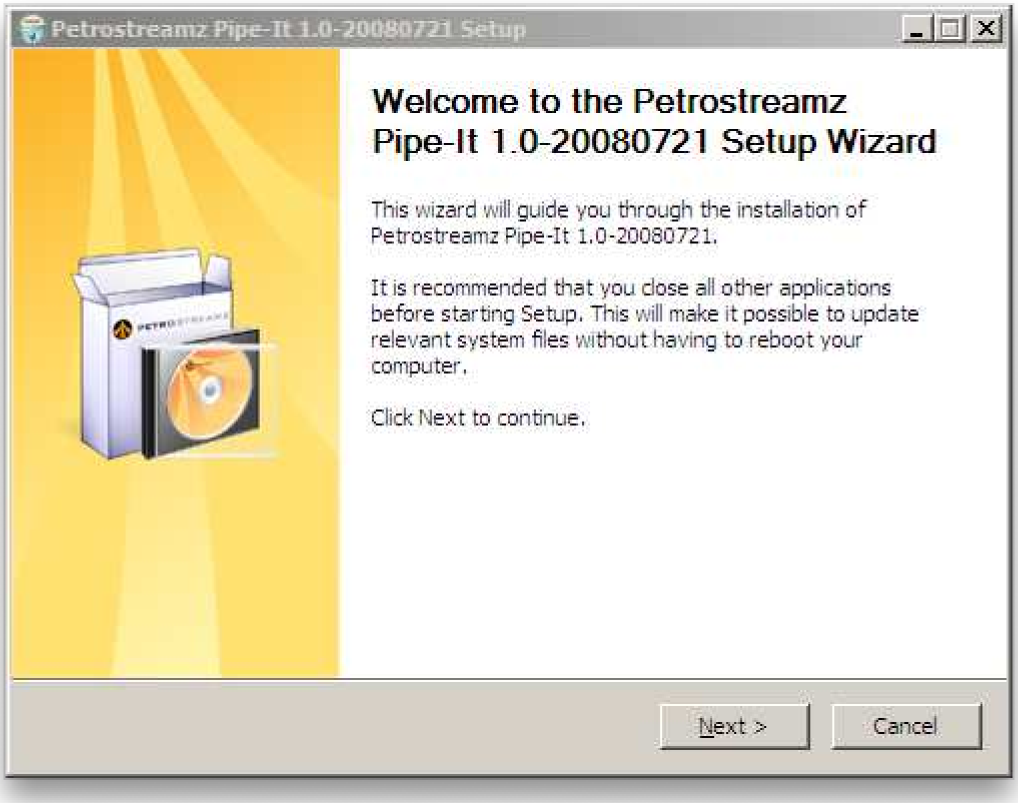
Petrostreamz Pipe-It 1.0, build 20080721.
Copyright © 2006 Petrostreamz AS.
All rights reserved.

If you experience any problems running this application or have additional questions, please contact us through any of these channels:

Executing Pipe-It Installer

Invitation Message

First screen that you see indicates that Pipe-It installation is successfully started and it will guide you with all installation steps and choices. Also it indicates build number to keep you aware of version you are installing:



Welcome to the Petrostreamz Pipe-It 1.0-20080721 Setup Wizard

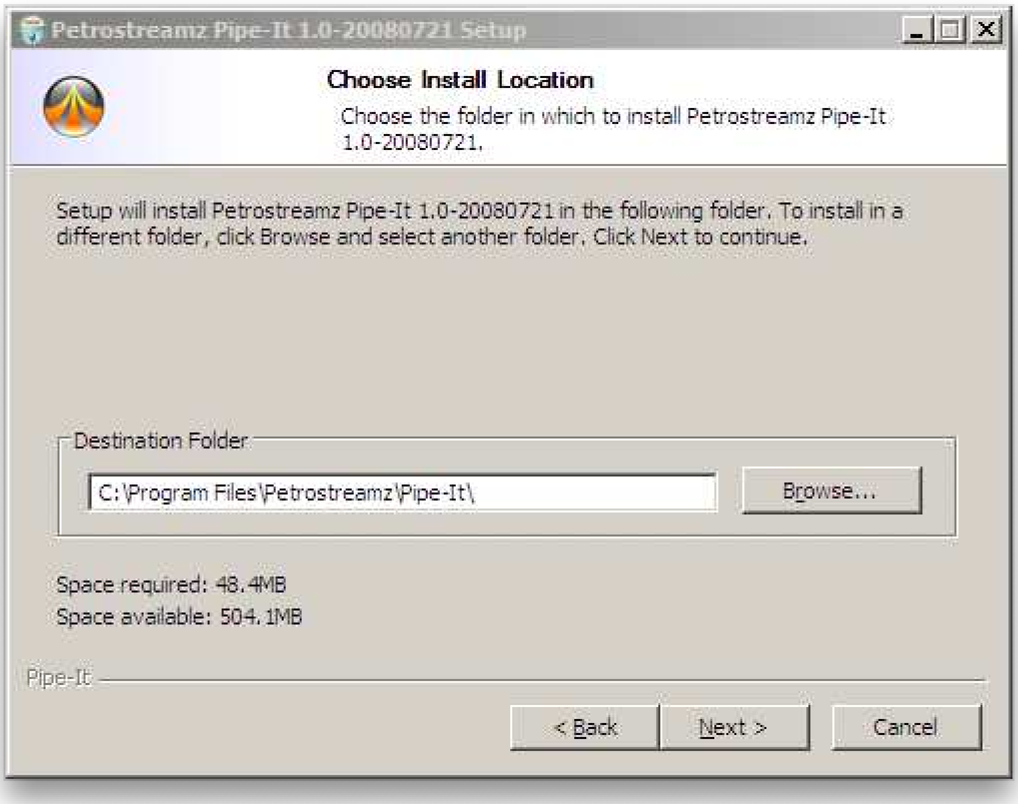
This wizard will guide you through the installation of Petrostreamz Pipe-It 1.0-20080721.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Choose Install Location

This page allows you to check installation path of Petrostreamz software and if required change it to another one that is convenient for your installation. Note that this installation path is stored in registry, so when you would like to update your current installation with new version of software you just need to run new version of Pipe-It installer and it will load the previous installation path and indicates it on the page - you need only to check this and press 'Next' button:



Petrostreamz Pipe-It 1.0-20080721 Setup

Choose Install Location

Choose the folder in which to install Petrostreamz Pipe-It 1.0-20080721.

Setup will install Petrostreamz Pipe-It 1.0-20080721 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

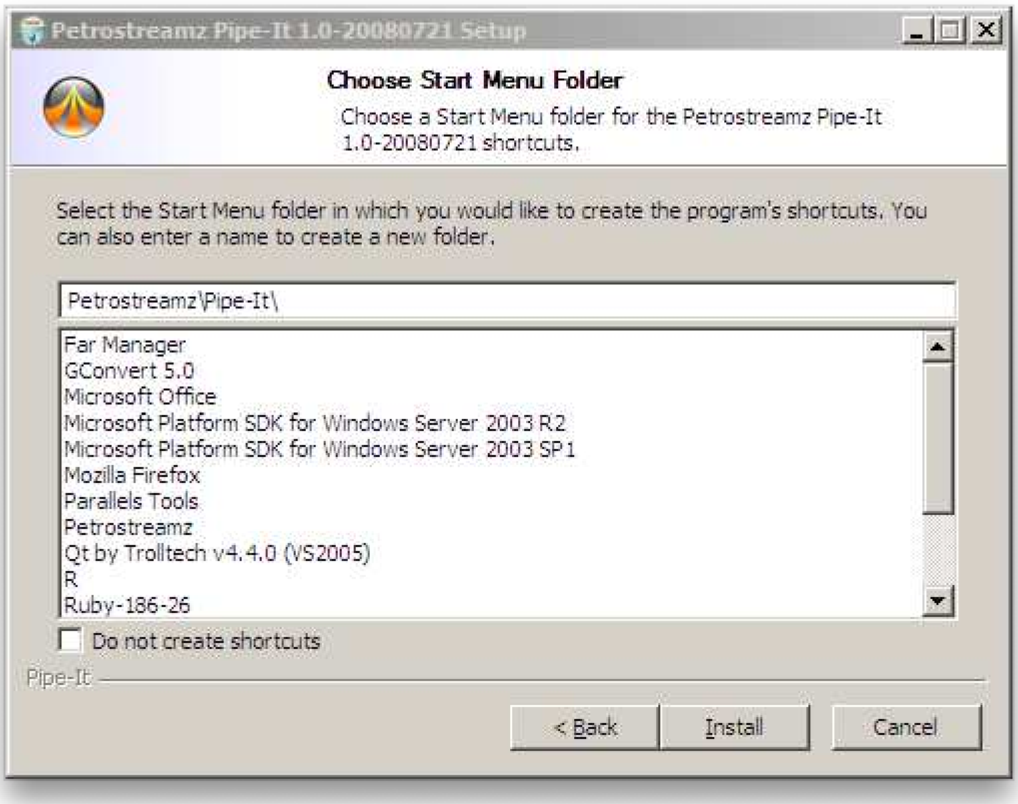
C:\Program Files\Petrostreamz\Pipe-It\

Space required: 48.4MB

Space available: 504.1MB

Choose Start Menu Folder

This page confirms your choice of creating the program's shortcuts. You can edit suggested name or totally disable creating shortcuts. In such case you have to remember where Pipe-It is installed to invoke it:


Petrostreamz Pipe-It 1.0-20080721 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the Petrostreamz Pipe-It 1.0-20080721 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

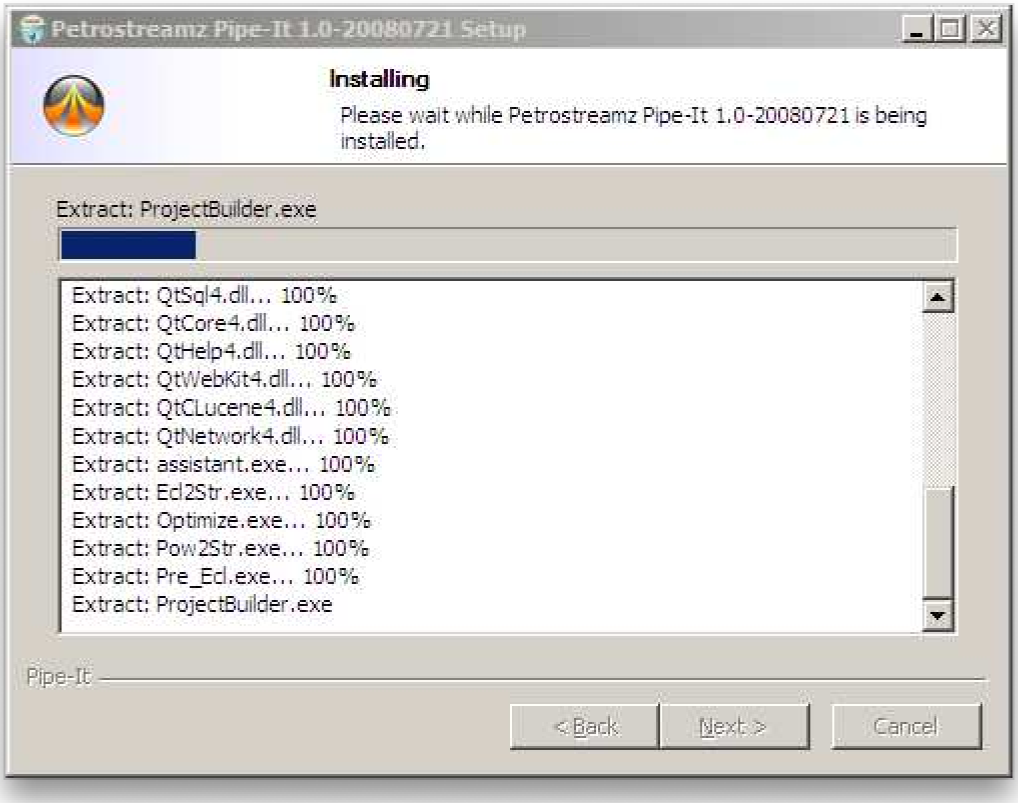
```
Petrostreamz\Pipe-It\
```

```
Far Manager
GConvert 5.0
Microsoft Office
Microsoft Platform SDK for Windows Server 2003 R2
Microsoft Platform SDK for Windows Server 2003 SP1
Mozilla Firefox
Parallels Tools
Petrostreamz
Qt by Trolltech v4.4.0 (VS2005)
R
Ruby-186-26
```

☐ Do not create shortcuts

Installation Progress

The page indicates installation progress and shows you the files being copied to your hard drive. Just wait few minutes to get the software installed:


Petrostreamz Pipe-It 1.0-20080721 Setup

Installing

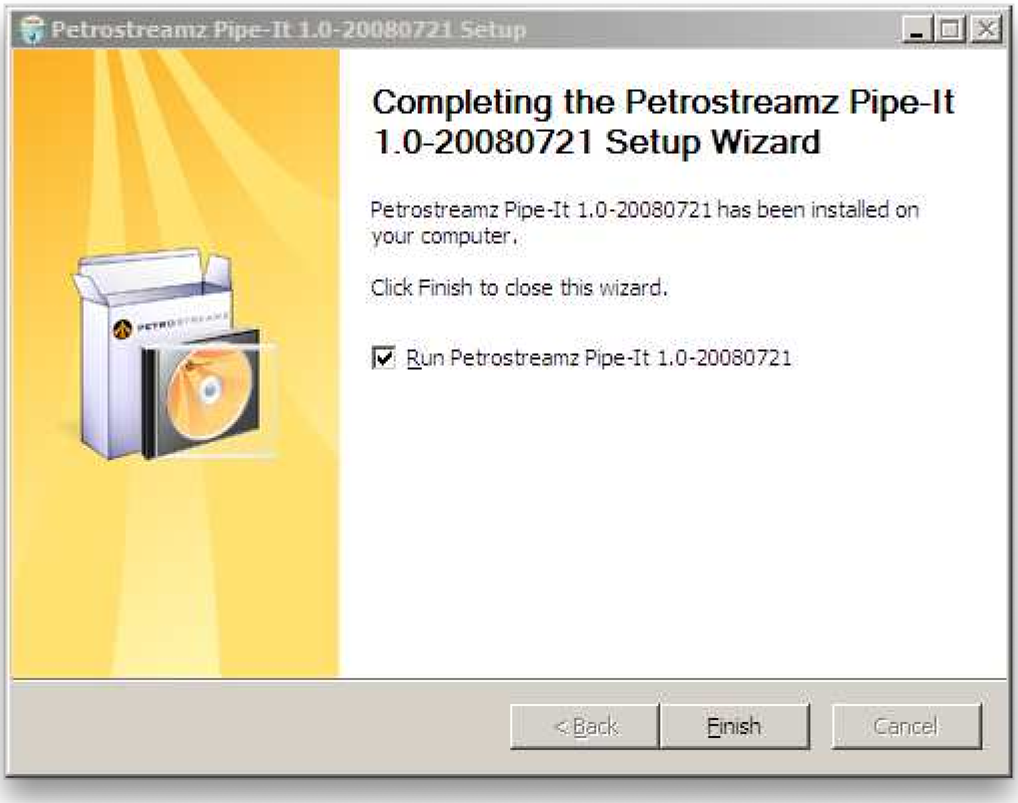
Please wait while Petrostreamz Pipe-It 1.0-20080721 is being installed.

Extract: ProjectBuilder.exe

```
Extract: QtSql4.dll... 100%
Extract: QtCore4.dll... 100%
Extract: QtHelp4.dll... 100%
Extract: QtWebKit4.dll... 100%
Extract: QtCLucene4.dll... 100%
Extract: QtNetwork4.dll... 100%
Extract: assistant.exe... 100%
Extract: Ecl2Str.exe... 100%
Extract: Optimize.exe... 100%
Extract: Pow2Str.exe... 100%
Extract: Pre_Ecl.exe... 100%
Extract: ProjectBuilder.exe
```

Installation is Completed

Finally you get a screen that indicates the completeness of installation. Also by default it allows to run Pipe-It software product just after closing the installation:


Petrostreamz Pipe-It 1.0-20080721 Setup

Completing the Petrostreamz Pipe-It 1.0-20080721 Setup Wizard

Petrostreamz Pipe-It 1.0-20080721 has been installed on your computer.

Click Finish to close this wizard.

☑ Run Petrostreamz Pipe-It 1.0-20080721

< Back | Finish | Cancel

Components

Pipe-It software product includes the following components distributed within installer binary:

Petrostreamz components

- *Pipe-It.exe* - Main GUI Application.
- *Pipe-Itc.exe* - non-GUI version of Pipe-It that can be invoked from other programs or by Pipe-It itself!
- *Streamz.exe* - The "Engine" of Pipe-It. Sophisticated stream management program with track record of industry-strength projects under its belt. Licensed using Sls Utilities (see below). Documentation available elsewhere on this site.
- *ProjectBuilder.exe* - Wizard-based builder of Pipe-It project. Uses a template (see below) designed for specific application (e.g. EclBOz) and customizes it based on user supplied data. The Wizard displays pages for user interaction and replaces/duplicates elements in the template to create a ready to run Pipe-It project.
- *Optimize.exe* - PSM/Optimize program for scheduling / in-phasing of models to meet constraints.
- Growing set of standard conversion utilities to facilitate integration of popular engineering software (e.g. Eclipse, Sensor)

3rd party components

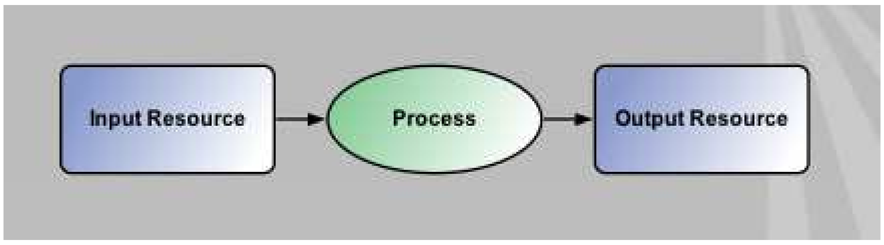
- *Qt Libraries* - Dlls to run Gui software based on.
- *UnxUtils* - Set of unix utilities built for win32 to use in projects for cross-platform issues. See also http://unxutils.sourceforge.net/ The installer program appends path to UnxUtils into system environment variable

PATH, so all these commands are available in command line.

Concepts and Terminology

Pipe-It is a completely generic framework for "piping" together a workflow. In the context of the petroleum industry one would reproduce the workflow of a project in the same way as it is physically piped together in real life. Any information related to quantity and stored in files on disk is a stream RESOURCE. Any operation that is applied on such a stream resource to produce another stream resource is a PROCESS.

The basic principle behind Pipe-It is to send a stream of information from a **resource** through a **process** into another **resource**, which, in turn, may be a resource for another process to form a chain of processes replicating the flow of petroleum in a producing field.

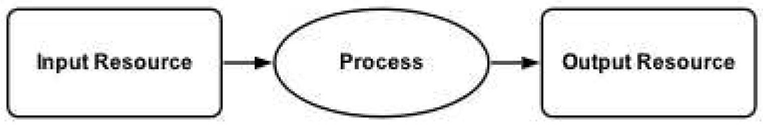


The **primary resources** are typically historical production data and/or production forecasts from e.g. a reservoir simulator. These primary resources are typically the entry point in a Pipe-It model of a petroleum asset. Each time a process is invoked, it needs to read and write into one or multiple resources. These **resources** are representations of files stored on disks. Normally this information contains the flow of petroleum components, but it can easily be translated into other monetary units like heating value or US$.

A Pipe-It **project** consist of two parts; a **visualization model** (*.ppv files) and a **logical model** (*.ppm files).

Logical Model

The **logical model** defines the flow of petroleum streams in a petroleum asset as it is “piped” together. This could often be thought of as a computational replica of a producing field exactly as it’s piped together. Opening the valve to start producing is analogous to pushing the run button in Pipe-It. Logical model contains only descriptions of connections, used files, used scripts and utilities but nothing about visual representation of model, colors, positions on canvas, etc (Let's show it as image without colors)

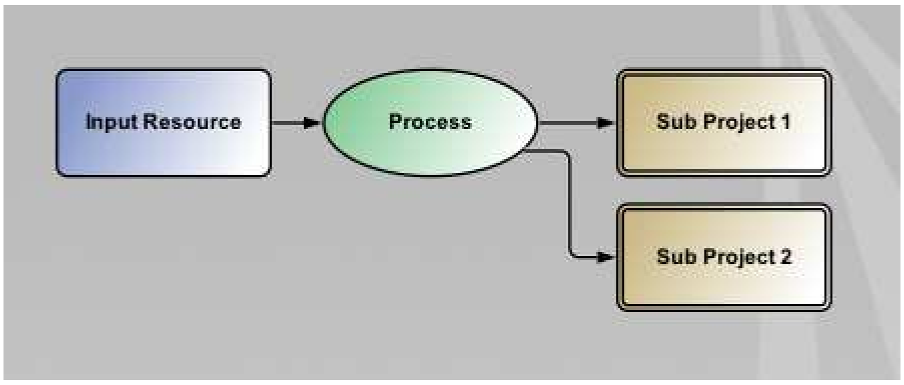


Visualization Model

The **visualization** of a logical **model** is an individual realization of the logical model. Hence two persons can work on a common logical model, but maintain individual visualizations of the same model. This unique concept is ideal for collaboration between groups with different focus.

Sub-projects or Composites

To facilitate this collaboration and individual visualization, Pipe-It uses a concept of a **composite**. A composite is a group of resources and processes collapsed into a single visual element containing a **logical sub-model**. By double-clicking on a composite element on the screen, the interior sub-model can be viewed and executed. Unlimited levels of composites can be nested inside each other.



Processes

To perform computational operations on resources Pipe-It invokes a set of **processes**. **Generic processes** (Scripters) are made available to launch any program that can be started from a command line.

Pipe-It can transparently forward information to and execute any 3rd party software executable via a command prompt (Scripter), such as Sensor, Eclipse, Hysys, VIP, OLGA, Excel etc. Hence it act as an **integrator** in complex oil and gas projects.

Projects

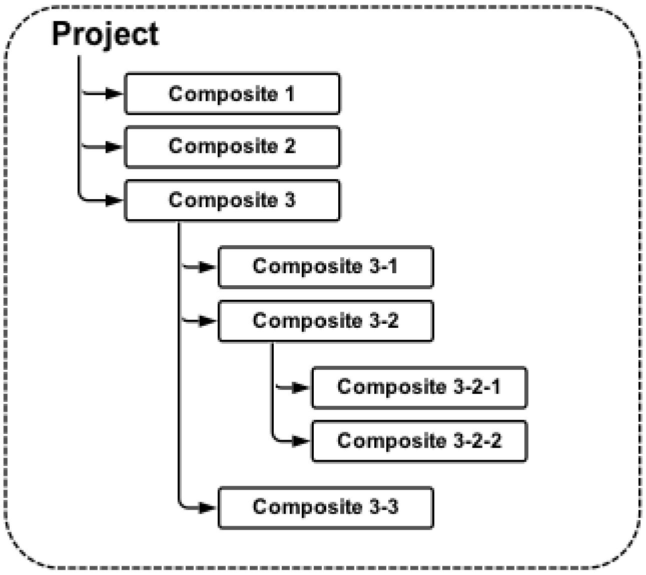
Project is container for all elements of modeling processes. Of course keeping all the elements on same canvas makes the scheme very complex for understanding and tracking processe details. To help with this some or all elements in a project can be organized in Sub-Processes or *Composites*.

We recommend to use the term *Project* for meaning whole project. Also it means a set of 3 files that defines the project - logical model + visual model + variable links (ppm + ppv + ppl).

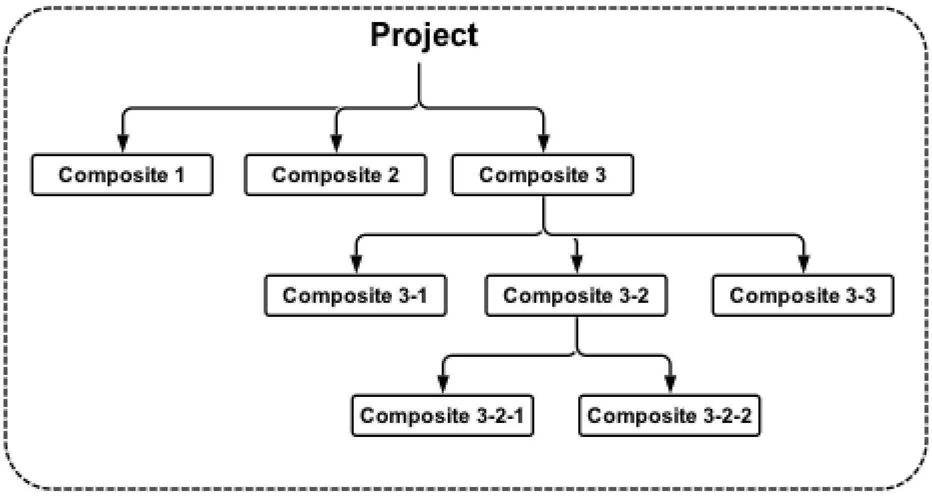
The term *Composite* is used for defining some part of whole project that is organized separately from other elements. Actually composites are small sub-projects that works in same way as other elements of canvas.

Because *Composite* is actually sub-project that defines some operations, it can also have nested composites of second level and so on.

From this point of view, the project is tree of composites:

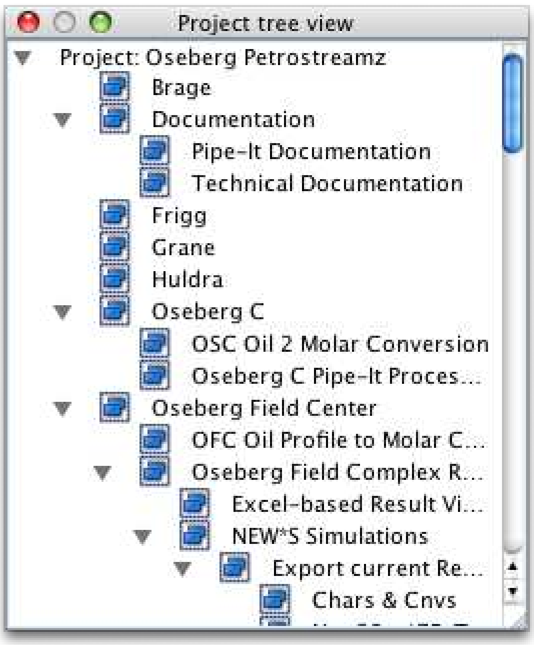


Or same schema from top to bottom:



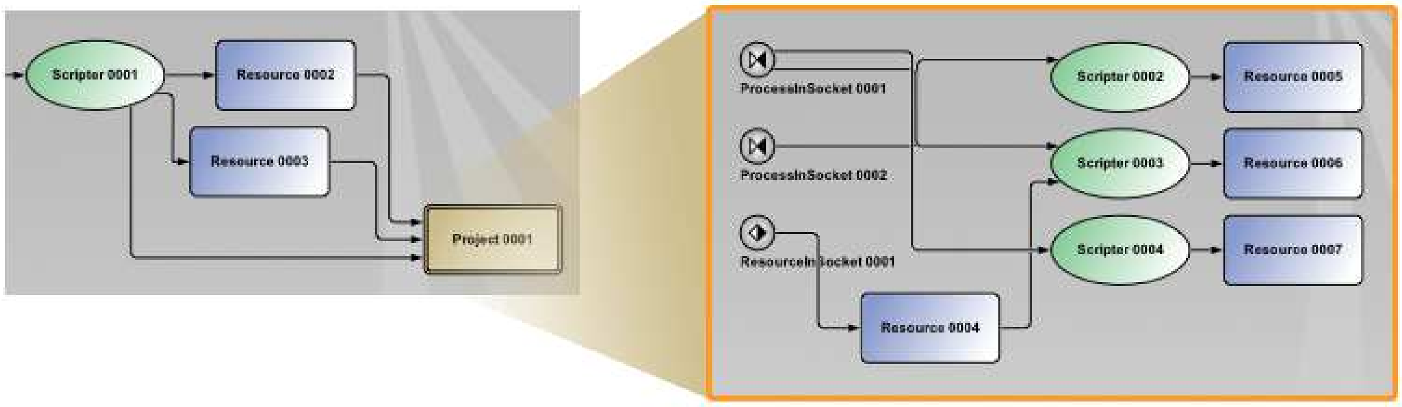
Composites

As mentioned in the section on Projects, composites are a way to make a project easier to understand. They will compose parts of a project together into one box for a less cluttered appearance. For example one may make one composite for the reservoir part of the field, and one for the processing part of the field. Each composite may also be run individually. Project can have any amount of nested composites:



Project: Oseberg Petrostreamz
Brage
Documentation
Pipe-It Documentation
Technical Documentation
Frigg
Grane
Huldra
Oseberg C
OSC Oil 2 Molar Conversion
Oseberg C Pipe-It Proces...
Oseberg Field Center
OFC Oil Profile to Molar C...
Oseberg Field Complex R...
Excel-based Result Vi...
NEW*S Simulations
Export current Re...
Chars & Cnvs

Connections from elements within a composite to elements outside (or inside other composites) are made via Sockets that look visually unique for each different type.



It is possible to export complete composites to standalone projects by using the *File→Export→Export Composite...* as a Project menu options. It is possible to import previously exported projects (or complete standalone projects) into another project as a composite. This is achieved by using the *File→Import→Import Composite...* via the menu options. Using a combination of these it is possible to clone composites.

See Also

- Sockets
- Socket Types

Resources

Resources are representations of files on disk. The default shape is a rounded rectangle with a pale blue color. They should usually be renamed to something appropriate in the context of the project. They need to be linked to the appropriate file on disk via "Set Resource file" context-menu option. An easy way to create a new (or multiple) resource is to drag the file from file explorer on to canvas. This creates the resource, gives it a default name and also set the file linking in on quick step.

Resources are connected to other processes only but never to another resource directly.

Processes

Processes are operation of some kind that modifies and upstream resource to create a new downstream resource . The default shape is an ellipse (oval) with a pale green color. They should usually be renamed to something appropriate in the context of the project. They are usually connected on the upstream and the downstream sides to resources. They can never be connected directly to another process. The most often used process is a generic Scripter where the user can specify any command line for execution.

The scripter command line can be explicitly entered by the user. For example if the scripter launches the reservoir simulator SENSOR and first argument is the input data file, the command codes into the scripter would be:

```
SENSOR.exe input.dat
```

However Pipe-It allows a flexible syntax to make projects and processes portable by using reference to connected resources. The files actually connected to those resource will then be used to make up the command line on-the-fly at time of execution. The details are provided below:

References

```
{<prefix>Rx:y@Sz<suffix>}(m)
```

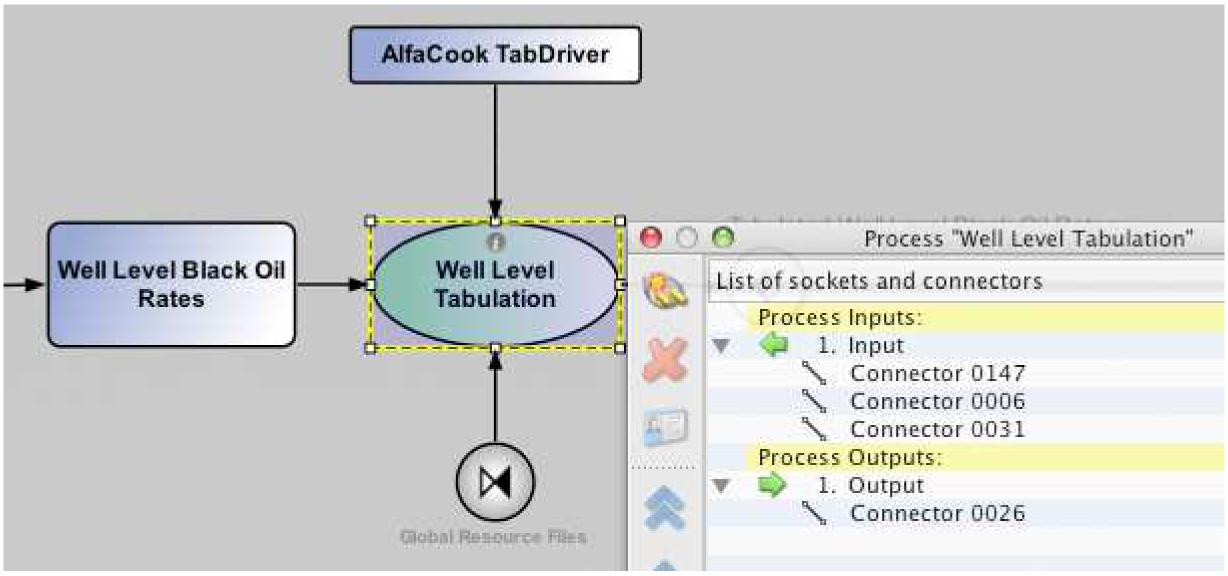
where *S* will actually be either *I* (for input) or *O* (for output), *z* will be an integer socket number, *x* and *y* will be either integer resource numbers or else of the form *n* or *n-i* (where *i* is an integer), *<prefix>* and *<suffix>* will be optional text strings (restricted only to not containing braces), and *m* will be the minimum number of matching files required for execution. The *:y* and *(m)* parts will also be optional (*y* will default to *x* and *m* will default to 0).

Sockets

Except for connectors, every element has a number of sockets. When you connect two elements, you actually connect an output socket from one to an input socket on the other. Each socket has a different purpose. Think of connecting a stereo receiver to a set of speakers. You can't just attach your wires at random. Each speaker will have exactly two terminals (sockets), but the receiver might have 20 or 30, all with a different purpose. You have to connect the receiver's

"Right Speaker Positive" terminal to the positive terminal on the right-hand speaker, and so forth.

All currently defined resources, manifolds, and distributors have exactly two sockets -- one for input and one for output. By default, they're named Input and Output, respectively. You're free to rename them, but you can't delete these sockets or add new ones. Currently defined processes (i.e., Scripters) also start with two sockets, Input and Output, but you can delete either one and/or add any number of either type, as well as rename them. Since you can define your own sockets for each process, it's up to you to define how they're to be used and to what they should be connected. Generally, though, if you plan to attach resources of different types to a process, it would make sense to define a different socket for each type of input or output.



List of sockets and connectors
Process Inputs:
1. Input
Connector 0147
Connector 0006
Connector 0031
Process Outputs:
1. Output
Connector 0026

From the contextual pop-up menu, bring up the *Edit Sockets* dialog for the Scripter. There you'll see all of your sockets and be able to edit them. You'll also see which connectors are attached to each socket and in which order (which is also editable). At the moment, to locate the attached resources, you'll have to trace the connectors manually (or else run the project and see what the Runner says about the connections in its log file).

TIP: If different "types" of resources are connected to a scripter and are to be used for different types of arguments on the command line then it is recommended to create new sockets for each type of connected resource, otherwise the wrong file may end up at the position of the wrong argument.

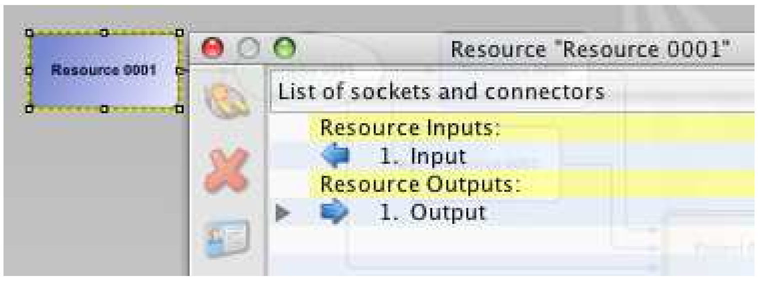
Socket Types

All elements (except connectors) have sockets. Let's pass through types to show what kinds of sockets can be used in appropriate items.

Resources

Resources have exactly two sockets -- one for input and one for output:

- Resource Input Socket
- Resource Output Socket



Resource "Resource 0001"

List of sockets and connectors

Resource Inputs:

1. Input

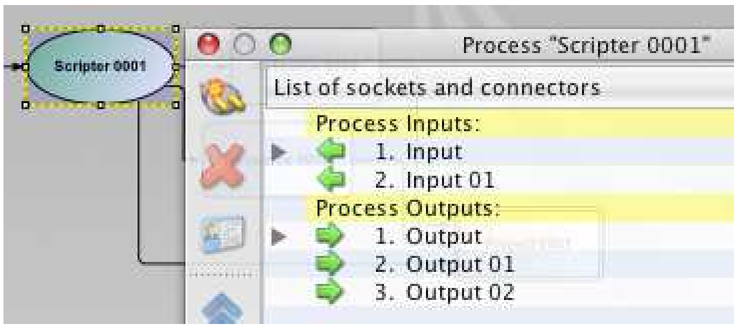
Resource Outputs:

1. Output

Processes

Scripters also start with two sockets, Input and Output, but you can delete either one and/or add any number of either type, as well as rename them.

- Process Input Socket
- Process Output Socket



Process "Scripter 0001"

List of sockets and connectors

Process Inputs:

1. Input
2. Input 01

Process Outputs:

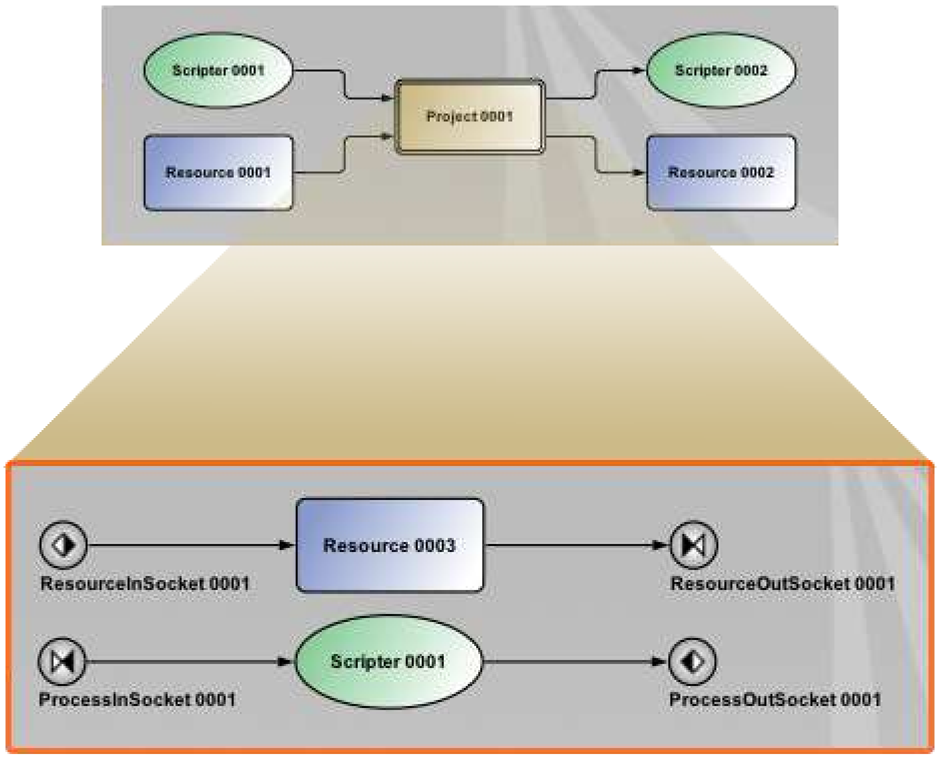
1. Output
2. Output 01
3. Output 02

Composites

Composites can have all 4 types of sockets due to all available internal elements:

- Resource Input Socket
- Resource Output Socket
- Process Input Socket
- Process Output Socket

Let's illustrate all possible types of sockets inside composites:



File types

The following file types are used by Pipe-It. The extensions are typical but not mandatory.

- ***.stz** Streamz engine driver files
- ***.str** Pipe-It resource (stream) files
- ***.psm** Pipe-It utility driver files
- ***.log** Log files containing messages logged by Streamz or utilities.
- ***.chr** Characterization files containing the format description of resources
- ***.cnv** Conversion files containing methods for conversion of resources from one format to another

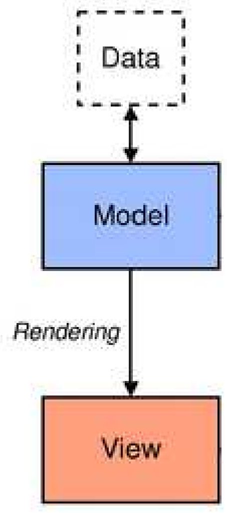
- ***.ppm** Logical project model in XML format
- ***.ppv** Visualization of the project model in XML format
- ***.ppl** Linkz library of the project in XML format

The program performs operations on a project depending on 3 required files:

- **Petrostreamz Pipe-It View** file with default extension .ppv that contains the project visual description of the elements of the project.
- **Petrostreamz Pipe-It Model** file with default extension .ppm that contains the project model description including all elements (Resource and Process) and element dependencies and execution information.
- **Petrostreamz Pipe-It Linkz** file with default extension .ppl that contains the project library of all links used in project.

To aid in starting pre-existing projects, it is recommended to associate the *.ppv file to the **Pipe-It.exe**. This way the program is launched with the project opened just by double-clicking the project file.

Pipe-It software architecture



Pipe-It is based on a model - view - controller paradigm, where the graphical user interface, *view*, (GUI) is what the user interacts with on the desktop. Behind the GUI we have a runner, or a *model* + *controller*. This runner will keep control over file and process connections, and will determine the sequence the model must be run in to give the correct answer and use valid data. One may say that the runner is responsible for interpretation and running of the logical model.

The majority of the computational effort of the model is performed by the Streamz engine running as a separate executable on your computer. The data exchange between the GUI and the Streamz engine is based on reading and writing of files in Streamz format.

A description of the Streamz file format is attached in Appendix A. A Streamz file in this example is file that contains information about quantities or additional basic information.

Software Components

Pipe-It includes the following components:

1. Gui Components
2. Runner
3. Streamz Engine
4. Optimization Engine
5. Operating Systems

GUI software

The Pipe-It is used to visually connect the different components of a petroleum asset together. The Pipe-It controls the flow of petroleum components from the reservoir to the point of sales. The logical model connection between all the components in the petroleum asset is stored as Pipe-It ppm (model) files. Each user may have independent visual representations of the model stored as ppv (view) files. Each project consists of one ppmodel file and one/multiple ppview files as well as the project related data files.

Graphics User interface of the Petrostreamz consists in two main components:

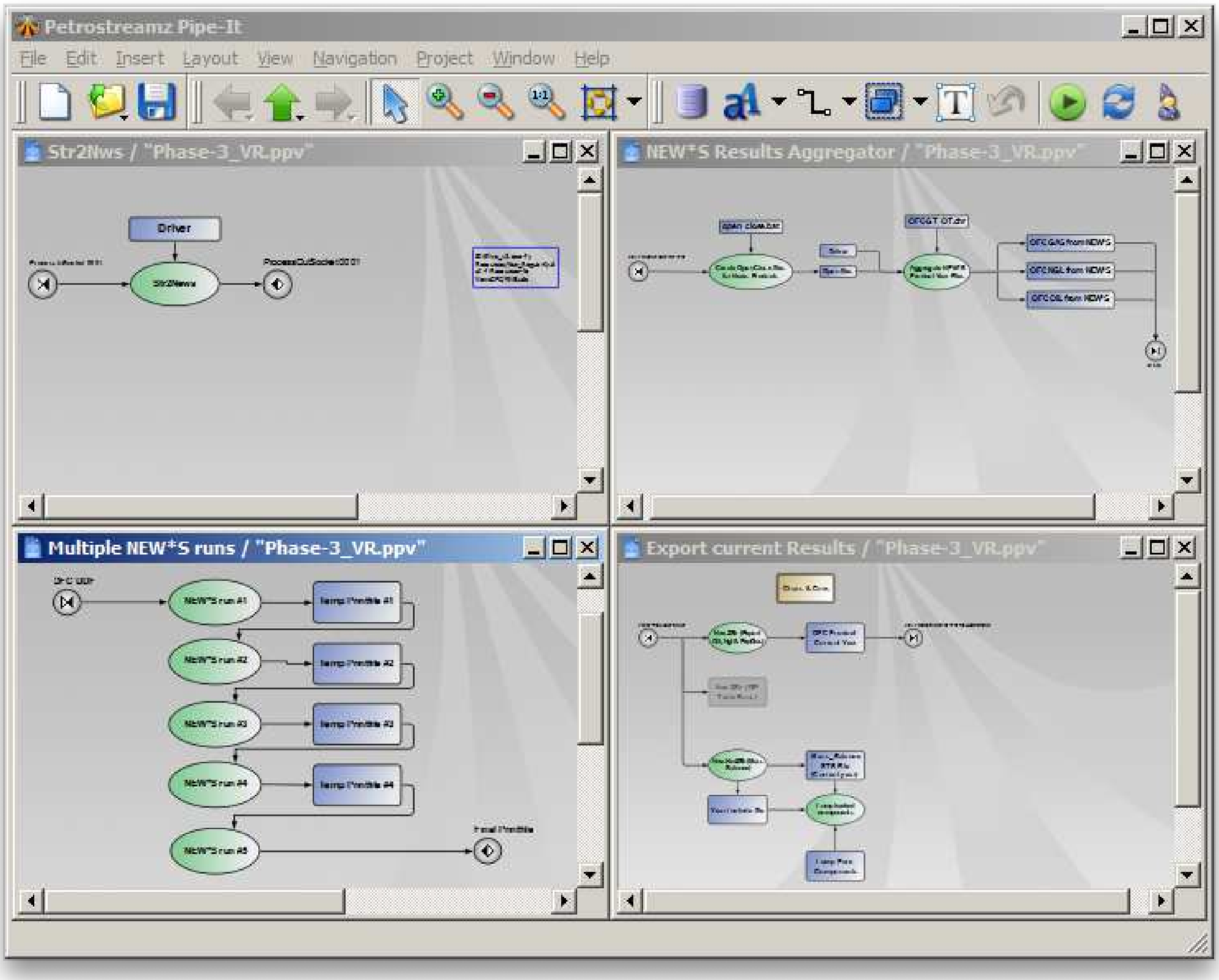
- Project Editor
- Optimizer Editor

Project Editor is user interface that allows to assemble project from elements, connect them, assign files and scripts, prepare to run and finally execute.

Optimizer Editor is tool and interface for working with completed project to define and assign variables, selecting aim and method of the optimization and execute the optimization process to find such set of parameters that provide maximum (minimum) target value by continuous runs of the designed project or composite.

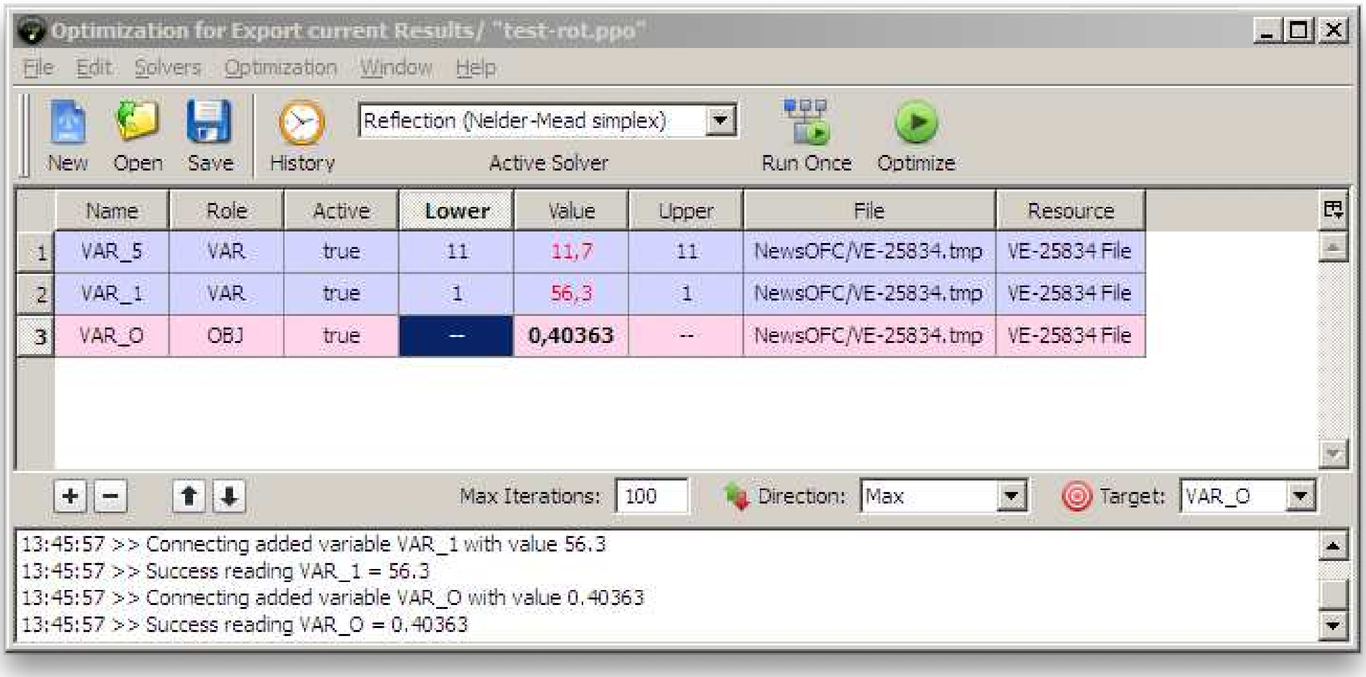
Main Window

Pipe-It Application implements Multiple Document Interface (MDI), whose windows reside under a single parent window


Petrostreamz Pipe-It
File Edit Insert Layout View Navigation Project Window Help
Str2Nws / "Phase-3_VR.ppv"
Driver
Str2News
NEW*S Results Aggregator / "Phase-3_VR.ppv"
Multiple NEW*S runs / "Phase-3_VR.ppv"
NEW*S run #1
NEW*S run #2
NEW*S run #3
NEW*S run #4
NEW*S run #5
Final Profile
Export current Results / "Phase-3_VR.ppv"

Optimizer Window

Optimizer Window shows properties of optimization process belongs to current view window of opened project



Active Solver: Reflection (Nelder-Mead simplex)

	Name	Role	Active	Lower	Value	Upper	File	Resource
1	VAR_5	VAR	true	11	11,7	11	NewsOFC/VE-25834.tmp	VE-25834 File
2	VAR_1	VAR	true	1	56,3	1	NewsOFC/VE-25834.tmp	VE-25834 File
3	VAR_O	OBJ	true	--	0,40363	--	NewsOFC/VE-25834.tmp	VE-25834 File

Max Iterations: 100 Direction: Max Target: VAR_O

```
13:45:57 >> Connecting added variable VAR_1 with value 56.3
13:45:57 >> Success reading VAR_1 = 56.3
13:45:57 >> Connecting added variable VAR_O with value 0.40363
13:45:57 >> Success reading VAR_O = 0.40363
```

Runner

The Runner is a behind-the-scene component of Pipe-It that orchestrates the execution of the project. It checks the status of all the elements, if all the resources have associated files connected, and decides the order of execution. It manages the launching of scripter command lines to the OS and waits for their completion. It determines the complete dependencies of the elements and keeps the elements on hold till all the upstream elements are up to date. The runner is visually invisible to the end user, who will only interact through the GUI.

Pipe-Itc allows running a model from a command line without invoking the GUI at all. In such case the Pipe-It Runner is only being invoked directly.

Runner's Project (Composite) execution logic

The rules for running composites are the following:

1. All *active processes* within the running composite, and within its nested, active sub-composites, will be checked and run in the proper order.

2. All *active resources* within the running composite, and within its nested, active sub-composites, will be checked and updated in the proper order.

3. All *processes* outside of the running composite will be considered inactive (as will any connection to or from such a process).

4. All *active resources* outside of the running composite will be checked and updated if attached to the running composite by a chain of active connections that is either strictly upstream or strictly downstream.

If one wants to completely isolate a running composite from all outside elements, the easiest way to do that is to deactivate the composite itself (non-recursively). The composite can still be opened and run in that state, but all outside connections will be considered inactive. This could prove to be a useful technique for testing individual composites. Be advised that the results might differ from those obtained by running the complete project, however.

Streamz Engine

The Streamz Engine is equivalent to the core of the previous PSM software. It incorporates much of the power of Pipe-It by providing a robust and time-tested stream conversion, management, filtering and separation functionality.

Streamz is a driver file driven and command-line invoked executable that is part of Pipe- It. The current version of Pipe-It can launch Streamz with pre-generated driver files in the same way it launches any other command-line program.

The commercial release version of Pipe-It will create Streamz driver files on-the-fly based on user specifications in the GUI and launch Streamz with those files. The Streamz primitives Copy, Tabulate & Separation would be supported and available.

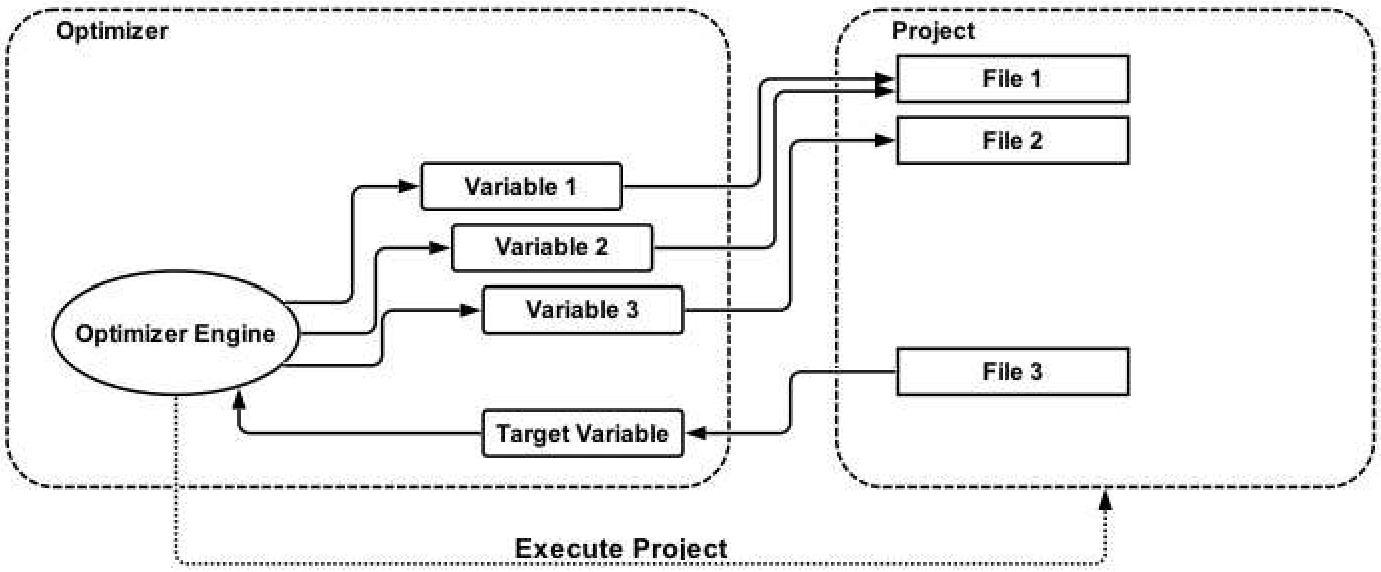
Refer to Streamz documentation for further (and more detail) information:

- Streamz Documentation

Optimization Engine

Pipe-It is designed to perform optimization globally or locally at any process being part of a Pipe-It project. The Optimization engine will use both in-house solvers as well as 3rd party solvers from Frontline Systems Inc. http://www.solver.com. The solvers used by Pipe-It uses plug-in technology to allow users & researchers to plug-in their solvers encapsulated in dlls, Documented of the API is provided to facilitate this.

Pipe-It optimizer allows the interface to attach, modify & monitor numerical variables in files linked to resources. This functionality in itself is quite unique and powerful and can be used for “manual” optimization without a proper solver (or regression routine).



This diagram shows relationship between Optimizer and Project. Optimizer has user-define variables that can read and write variables values from / to Project resources. *Optimizer Engine* set variables values and run Project. As execution is finished, Variables load values from updated resources and provide them to *Optimizer Engine*. *Optimizer Engine* evaluates target values and calculates new values for variables for next iteration and inserts them to Variables. Then everything is repeated until *Optimizer Engine* fits defined criteria - then the optimization is stopped, the aim is completed and set of parameters that provide maximum (or minimum) target value is found.

See Also

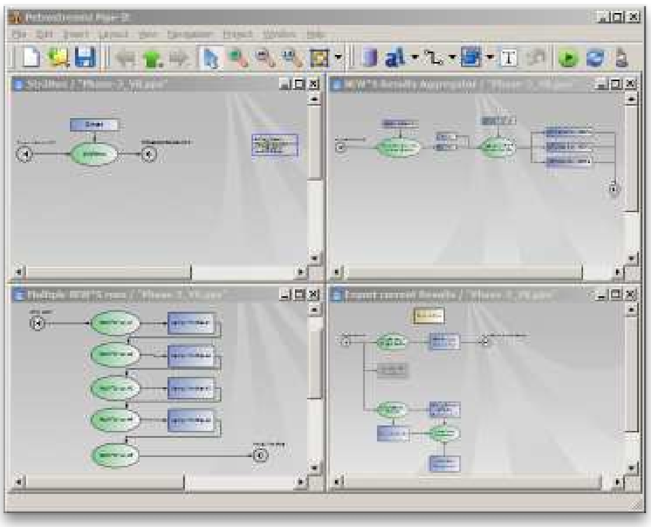
- Optimizer
- Optimization Variables

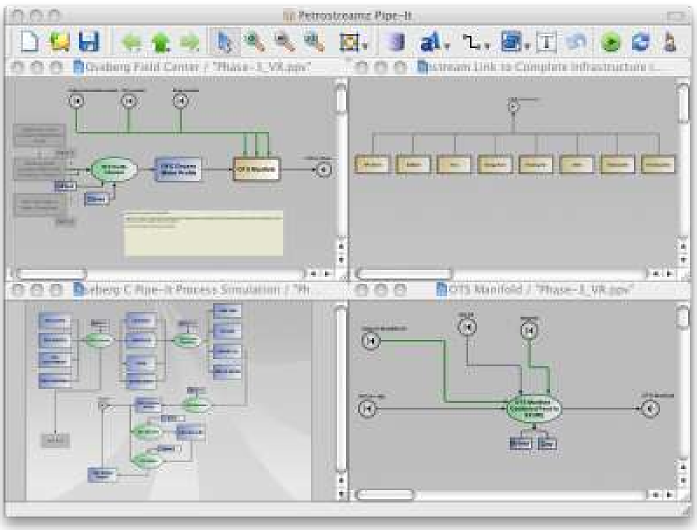
Operating Systems

Pipe-It is written in C++ using the Qt toolkit from Trolltech. Hence the product is developed in a cross platform environment and made commercially available for the following operating system platforms:

- Microsoft Windows 2000/XP
- MacOS

Versions running RedHat Linux or additional Linux flavors can be made available upon request.

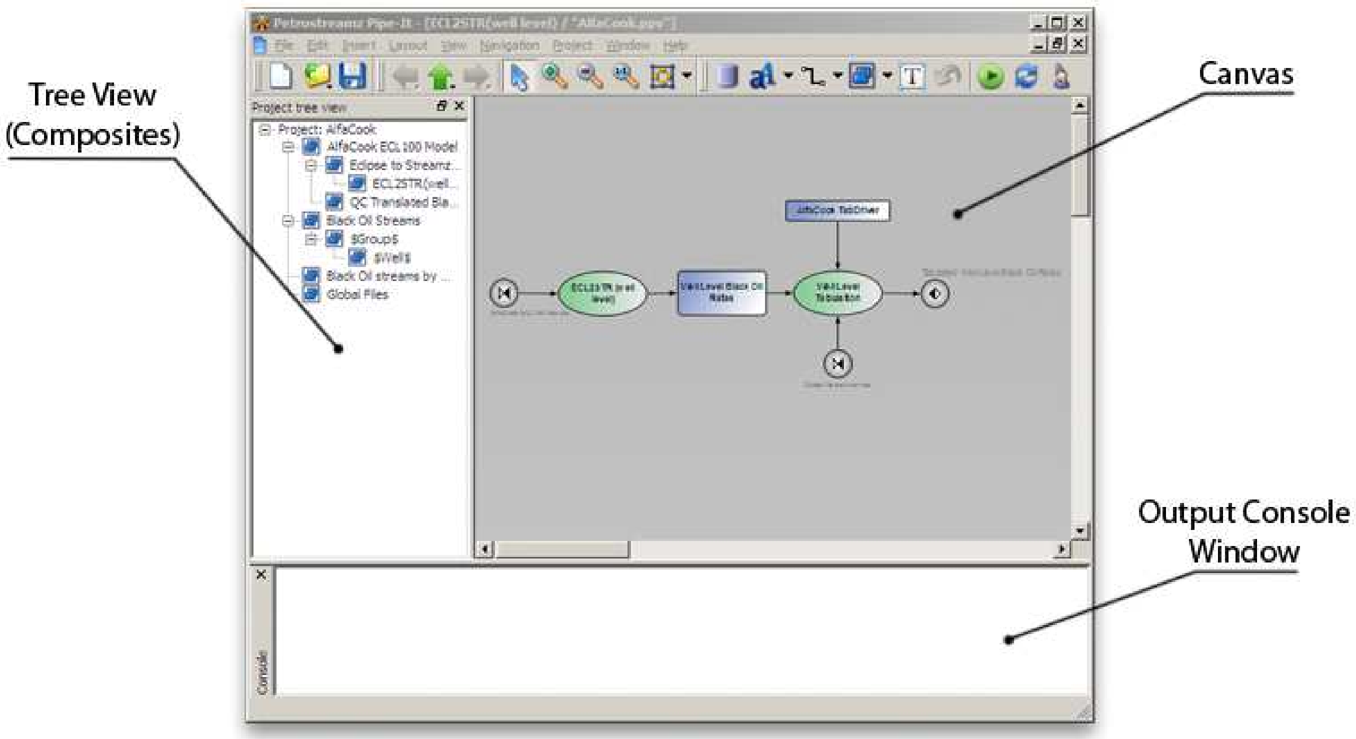




Graphical User Interface

The user interface in Pipe-It is designed to take advantage of the average PC familiarity with Microsoft products. It is not a coincidence that what you see is similar to what you get in a Microsoft Office application, pull down menus, toolbars and icons. All are comparable to the ones found in Pipe-It. The user interface is designed according to the Microsoft standard.

There are 3 main parts of the Pipe-It user interface:



1. **Pipe-It Canvas** – is the main window where the end user creates his/her projects by dragging and dropping process and resource elements to the canvas and directing the flow direction by drawing connections between the elements. A single project may use several nested canvases to display the details of a project.

1. **Pipe-It Text Output** – is an output window where messages from the under laying engine is displayed to the end user. This includes messages from other 3-rd Party software that is part of the project and was invoked as part of the "run".

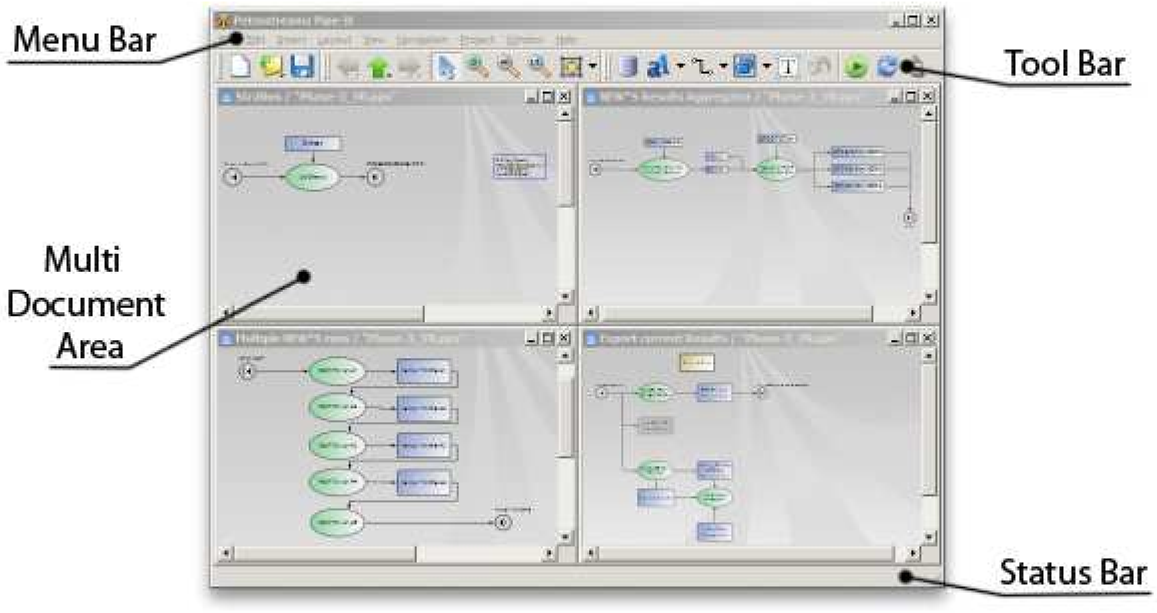
1. **Pipe-It Explorer** (Tree view) – The Pipe-It Explorer has the same look and feel as the Windows Explorer, objects for each piece of data organized into folders and sub-folders. Objects can be dragged and dropped into folders, double clicking on an object opens its settings dialog, while right clicking displays a range of functions to perform on the object.

The Pipe-It Explorer and the Pipe-It Text Output are dockable windows and may be moved to a more convenient location for the end user. If the end user has multiple Pipe- It canvases, one may cascade or tile them for better visibility.

Main Window

The main window of Pipe-It Application has visual standard components such as:

- Menu Bar
- Tool Bar
- MDI Area
- Status Bar

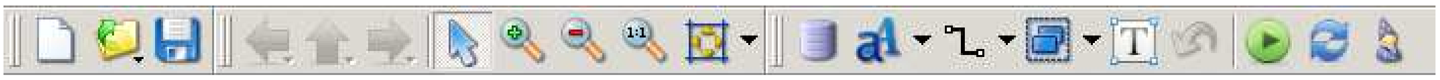


Consider these components more detailed:

- Tool Bar
- Mdi Area

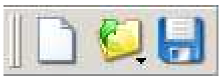
Tool Bar

Application Tool Bar consists of 3 main sections:



Now describe all available actions in the tool bar:

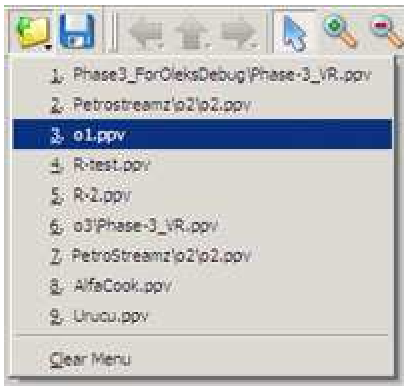
File Operations



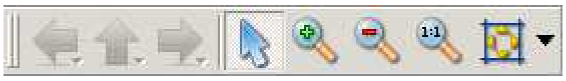
This section has 3 main standard file actions as:

- Create New Project
- Open Project from disk
- Save Project

Note that *Open* Icon has small arrow in right bottom corner. If the *Open* icon is pressed and holden for one second then drop-down menu appeared with list of recently opened files.



Navigation, Selection and Zoom



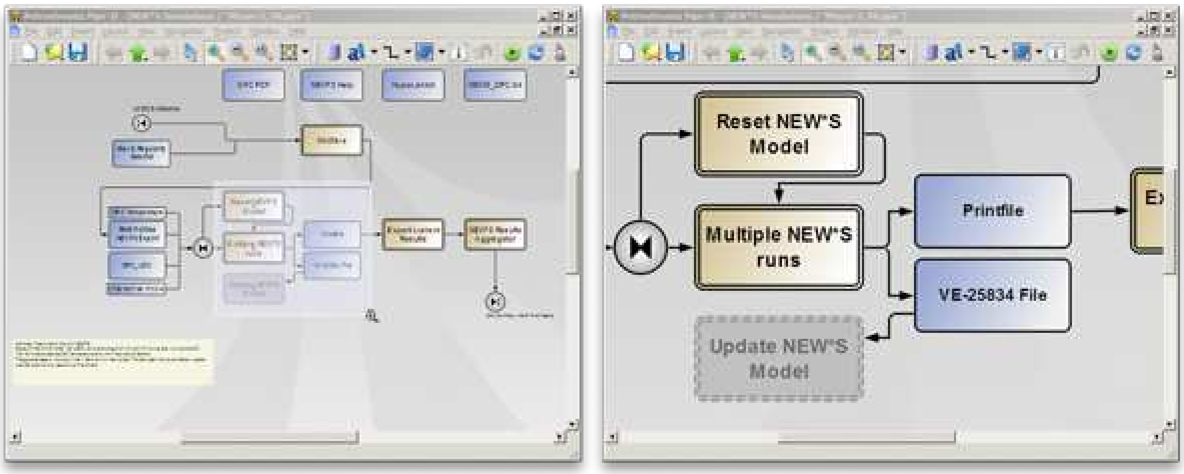
First 3 buttons allow navigation through currently opened composites. *Back* and *Forward* work in same way as navigation issues in web browser. *Up* button changes view to upper-level parent composite. All these buttons can be pressed and holden for one second to get list of history entries or, in case of *Up* button, for list of all parents of current composite.

Next 3 button as *Select*, *Zoom In* and *Zoom Out* manage the current mouse cursor mode.

- *Select* allows selection, resizing, moving etc canvas actions on mouse click
- *Zoom In* on click to canvas changes the zoom level for +10%
- *Zoom Out* on click to canvas changes the zoom level for -10%

When *Zoom In* or *Zoom Out* is selected, then holding *Alt* (on Mac: *Option*) button temporally switch to opposite zoom action.

Also in *Zoom In* or *Zoom Out*, area selection on canvas switches view to *Zoom-To-Fit* the selection area into view:



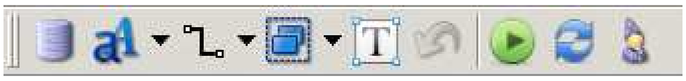
Next, **1:1** button is not *mode* button, it just switches current view to un zoomed (1:1) scale or in other words to 100% scale.

Finally, Last button in the tool bar is *Zoom-To-Fit* action with drop-down menu. Just pressing the button switch zoom level to fit all your presented canvas elements in current view. It has drop-down menu to apply *Zoom-To-Fit* to:

- Current Window - *Ctrl+9*

- Opened windows of Current Project - *Ctrl+8*
- All Windows - *Ctrl+7*
- Zoom-to-Fit whole Canvas - *Ctrl+6*

Elements to insert and Run

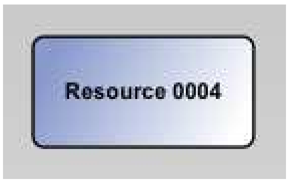


First there are 5 buttons which allows to add new Elements to Canvas.

Insert Resource

This button means adding a Resource to canvas. The resource object represents a file on disk. Though you can give a several name to the resource without a connection to actual used file name of the associated file on disk.

On canvas the object looks in such way:



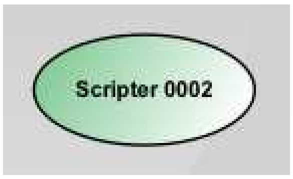
First double-click on the object opens *Select File Dialog* to associate a file with the object. A double-click on the object, that already has association, runs OS defined default Application associated with the file. For example, clicking on a resource with associated "text.txt" file, will open *Notepad.exe*

Also you can just *drag-and-drop* from Window Explorer to Canvas a file object. It will create a Resource automatically, associate the file to resource object and renames the resource it to *File Name* of dropped file.

Insert Scripter

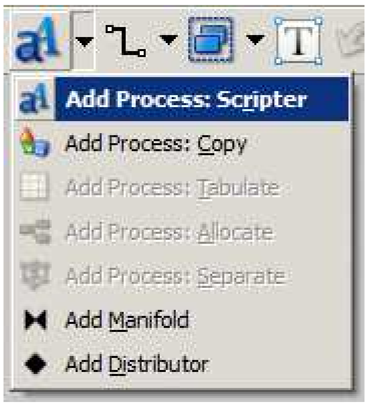
This button means adding a Scripter to canvas. The scripter object represents a shell command to execute.

On canvas the object looks in such way:



Double-click on the object opens *Edit Script Dialog*.

Sub-menu associated with the button allows to change default action to insert other elements that are bound to *Processes* category.



Insert Manifold

This button means adding a Manifold to canvas. *Manifold* – gather information from multiple resources into one visual connector.

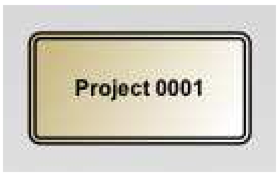
Insert Distributor

This button means adding a Distributor to canvas. *Distributor* – distribute output from a process into multiple equal connectors.

Insert Composite

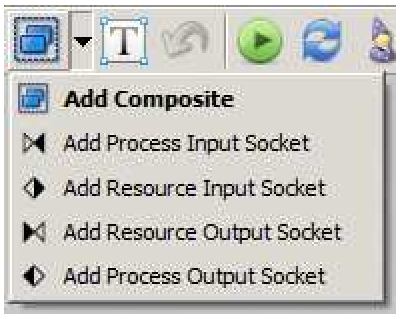
This button means adding a Composite (Sub-Project) to canvas.

On canvas the object looks in such way:



Look at Composites? and Socket types? pages for more details.

Sub-menu associated with the button allows to change default action to insert other elements that are bound to *Composites* category.


Add Composite
Add Process Input Socket
Add Resource Input Socket
Add Resource Output Socket
Add Process Output Socket

It allows to insert sockets objects which allows to drive connection from / to the current composite.

Insert Annotation

This button means adding a object with text or image that does not play role in logic but can represent some illustration for the project.

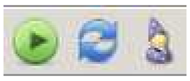
Undo Button

provides fast way to revert your changes for one step back.

Double clicking for continuos insert

You may also double click on buttons on the tool bar. This will “turn on” the tool until you choose another. This will prove practical if you want to add more than one element of the same type to the canvas. For example if you have lot of connectors to make, just double click and make all your connectors as you want them to be, without having to turn the connector tool on for each one.

Group of Run actions



The section has 3 buttons that allows you control a running of current composite.

- First is *Run* that means to run current composite.
- *Refresh* clean status indications that appear on checking states of Elements during a running the composite.
- *Optimizer* button invokes the optimization window bound with current project

MDI Area or Workspace

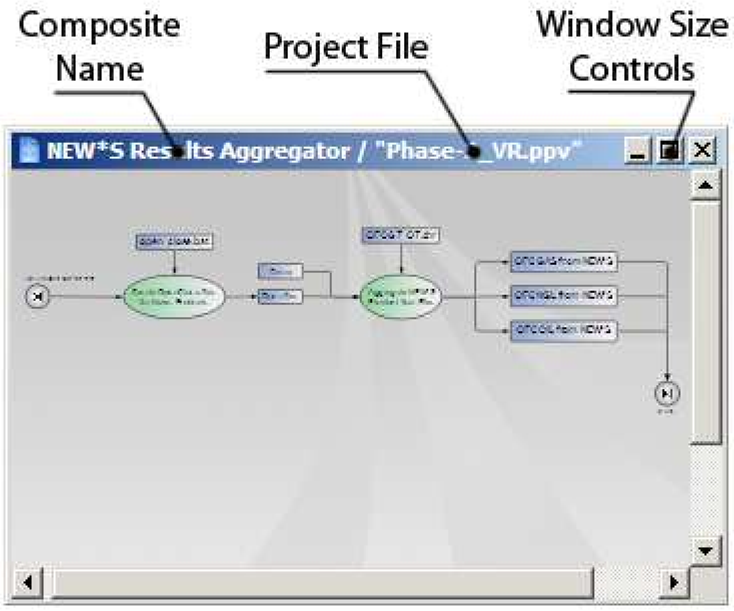
The component provides a workspace area that can be used for MDI. The workspace itself is used to display a number of child windows, each of which is a view of project or composite.

Pipe-It Architecture allows to keep several opened projects (project files) and each such project can have few opened windows (views) same time.

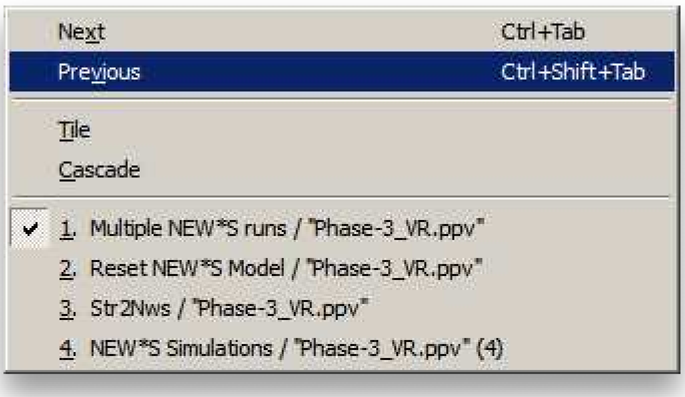
The workspace component keeps together all visible views from all opened projects.

If user closes last window of some project then the project is being closed. So it means that each opened project has at least one window inside workspace area.

Each window in MDI Area has own controls for window size management. Window title indicates Composite Name opened in the window and File Name of Project file. It keeps you aware of project you are working with in case of few opened project files.



Behavior and geometry of child windows is controlled via *Window* menu. It is possible to switch to next or previous window in workspace stack, tile them or cascade or just quickly switch to required window by appropriate menu item:


Next Ctrl+Tab
Previous Ctrl+Shift+Tab
Tile
Cascade
1. Multiple NEW*S runs / "Phase-3_VR.ppv"
2. Reset NEW*S Model / "Phase-3_VR.ppv"
3. Str2Nws / "Phase-3_VR.ppv"
4. NEW*S Simulations / "Phase-3_VR.ppv" (4)

Keyboard Shortcuts

There are such Keyboards Shortcuts available for switching through child windows:

Windows:

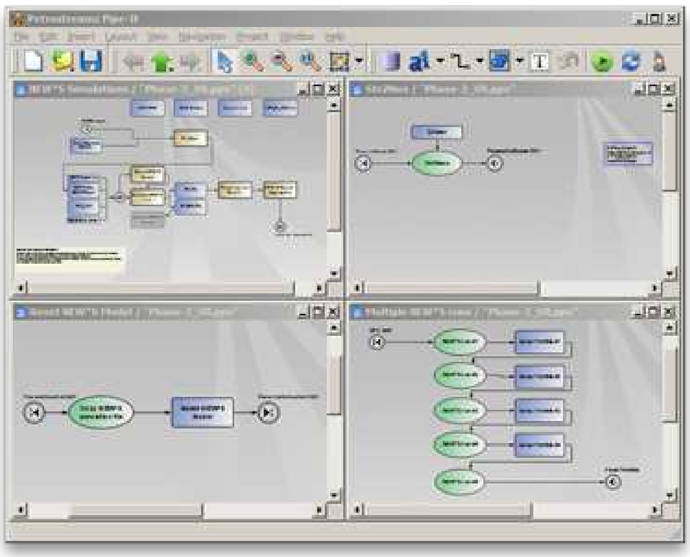
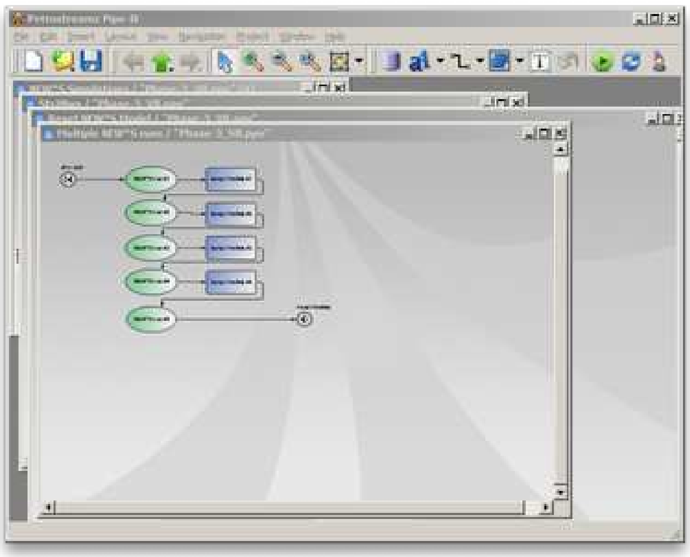
- *Ctrl + Tab* : Next window in workspace
- *Ctrl + Shift + Tab* : Previous window in workspace

Mac OS X:

- *Cmd +* `: Next window in workspace
- *Cmd + Shift +* ` : Previous window in workspace

Tile and Cascade

These two functions can be used to reorganize workspace of your windows you are working on, but maybe most used mode can be stack of maximized windows.

Interactions with Canvas

Adding new Elements to Canvas

Insert Menu items and corresponding toolbar icons are typically used to insert new elements like resources, processes and connectors. To add multiple instances double-clicking the icon allows repeated additions on the canvas. Every subsequent click adds an instance of the element currently chosen. To come out of this mode use the ESC key.

Connectors are invoked in similar fashion from the Insert menu item or toolbar Icon, but one needs to move the mouse to the source element first (resource or process) when the element comes alive and shows its socket dock positions. The nearest dock becomes enlarged and the user chooses it by clicking (and releasing) the mouse button. The start of the connector is set and then the user moves to the destination element and repeats the procedure to set the end of the connector.

NOTE that connections between resource-resource and process-process pairs are prohibited and the GUI will not allow it (the wrong element does not expose its socket dock positions).

Selecting, Moving and Resizing

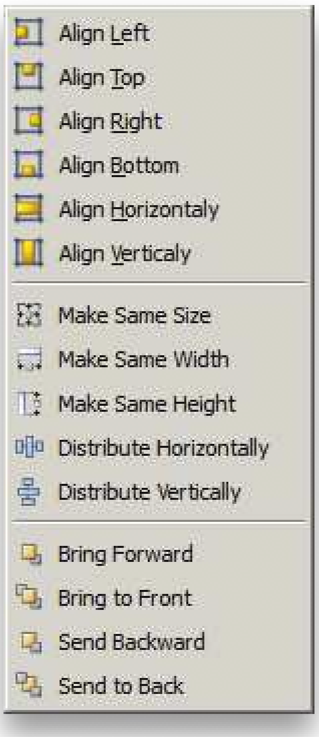
Selecting a single element is simple - just click on it. Selecting multiple elements can be done by shift-clicking and/or ctrl-clicking on the additional items. Also a rubber-band selection can be obtained by enclosing the required element by means of the mouse within a rectangle. The selected element(s) become visually highlighted.

Moving an element is by moving the currently selected element while keeping the mouse button pressed. If multiple elements are selected they will all move. If two elements are connected and one of them is moved, the appropriate end of the connector moves too.

Align, resize and Z-order

Resizing of individual elements are done by dragging the "handles" in appropriate direction. More advances alignment, resizing and Z-order of multiple elements are controlled by the Layout menu item.

Layout menu



All these options are about appearance of your project. One can align the elements chosen in many different ways, including the most used, horizontally or vertically.

Under the resize option one can make two or more elements the same size, same width, height. One may also distribute them horizontally or vertically for a clean look. (This option sets the distance between elements the same).

The Z-order option is only useful if you have overlapping elements. Then you can bring an element to the front or to the back of the canvas. If you have more than two elements on top of each other, the two last options under Z-order is useful, to alter the order of the overlapping elements.

Grouping Elements into Composites

See Also

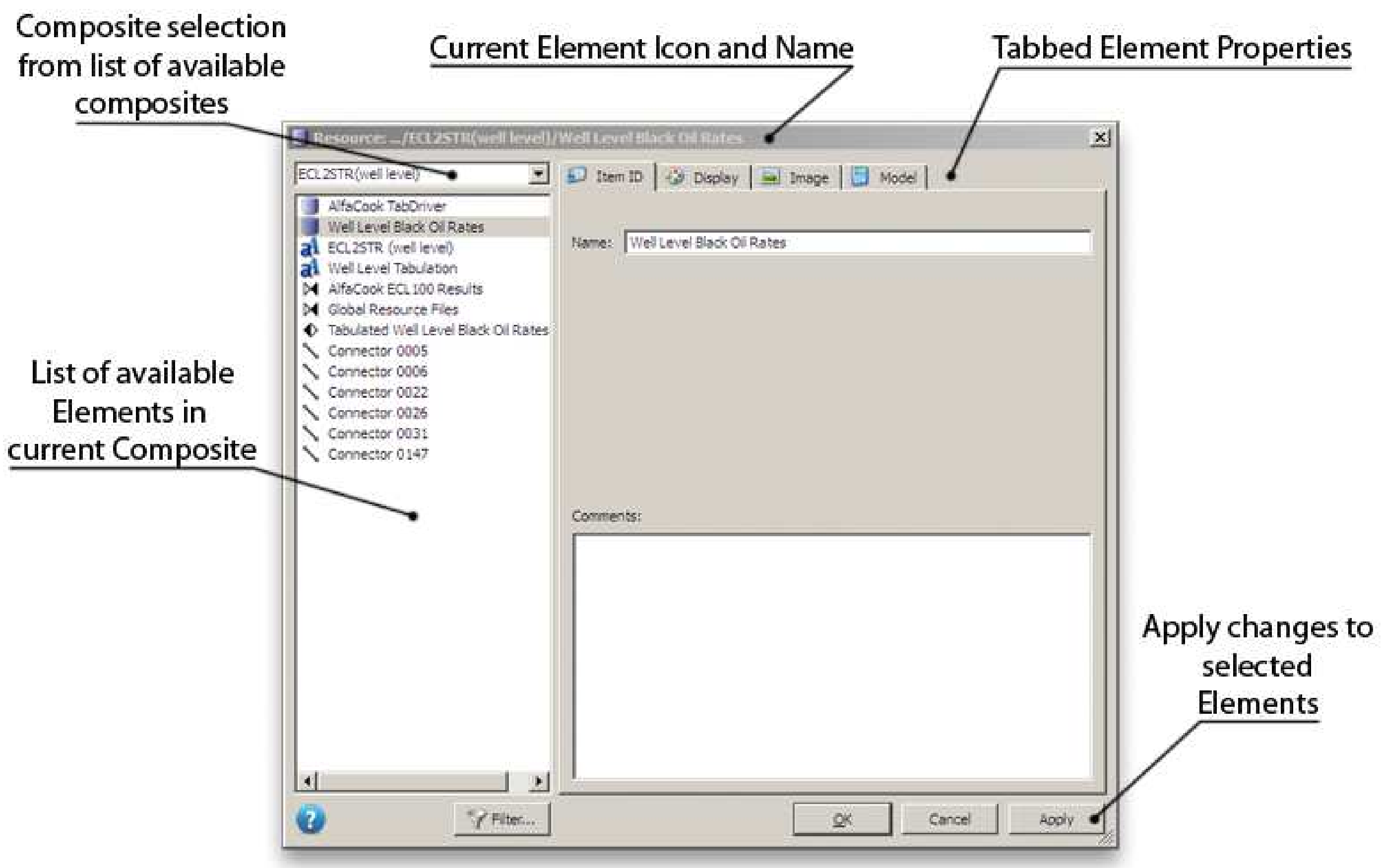
- Item Properties
- Item Filtering
- Double Clicking
- Context Menu

Item Properties Dialog

Item properties are available by making Right-button click on selected items, or by *Keyboard shortcut* - **Ctrl+I** (Mac: **Cmd+I**). Also it is accessible from

Edit > Item Properties...

The action shows tabbed interface for all properties of current or selected item(s) such as names, visual properties etc.


Composite selection from list of available composites
Current Element Icon and Name
Tabbed Element Properties
Resource: .../ECL2STR(well level)/Well Level Black Oil Rates
Item ID
Display
Image
Model
List of available Elements in current Composite
AlfaCook TabDriver
Well Level Black Oil Rates
ECL2STR (well level)
Well Level Tabulation
AlfaCook ECL100 Results
Global Resource Files
Tabulated Well Level Black Oil Rates
Connector 0005
Connector 0006
Connector 0022
Connector 0026
Connector 0031
Connector 0147
Comments:
Filter...
OK
Cancel
Apply
Apply changes to selected Elements

ECL2STR(well level)

Name: Well Level Black Oil Rates

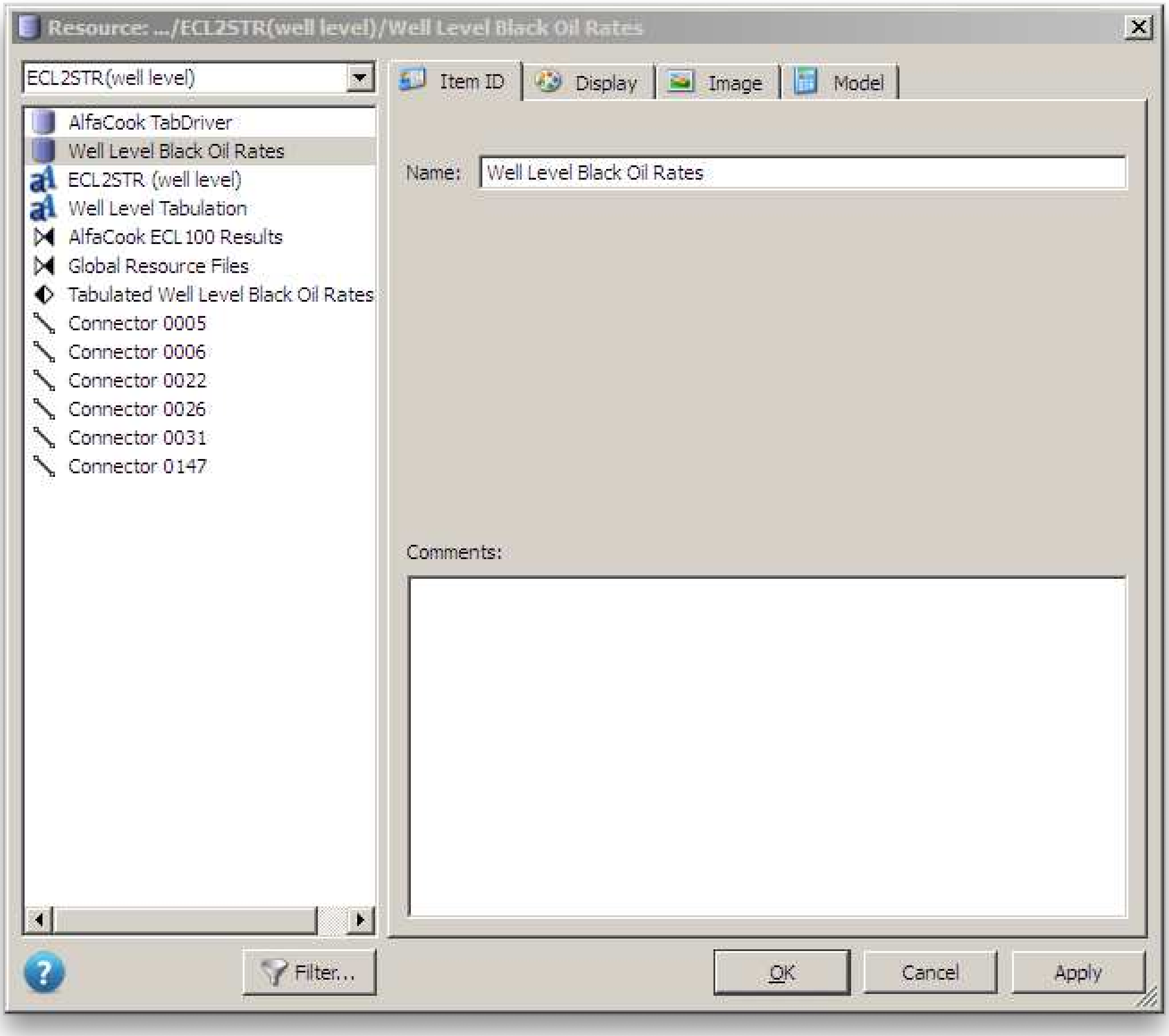
Now you can see it in details on the image. Left side contains list of available Elements in current composite or all Elements in whole project (then in *Composite selector* you have to choice *<All composites>*). You can select items which you would like to apply properties changes to by mouse or using *Filter...* button at bottom.

See Also: Item Filter dialog

Now we can consider all tabbed properties:

Name and Comments

This page has *Name* and *Comments* properties. Note that *Name* is unique identifier so if you enter same identifier as already presented in current composite, the dialog can not accept the identifier.



AlfaCook TabDriver
Well Level Black Oil Rates
ECL2STR (well level)
Well Level Tabulation
AlfaCook ECL100 Results
Global Resource Files
Tabulated Well Level Black Oil Rates
Connector 0005
Connector 0006
Connector 0022
Connector 0026
Connector 0031
Connector 0147

Name: Well Level Black Oil Rates

Comments:

Display Properties

Display page contains all settings related how Element is shown on canvas. It can be grouped by next sections:

Shape Geometry

Here you can apply geometry changes and select shape type you would like to use for the item.

- Ellipse
- Rectangle
- Rounded Rectangle

(for last case property *rounding %* is available)

Line Style

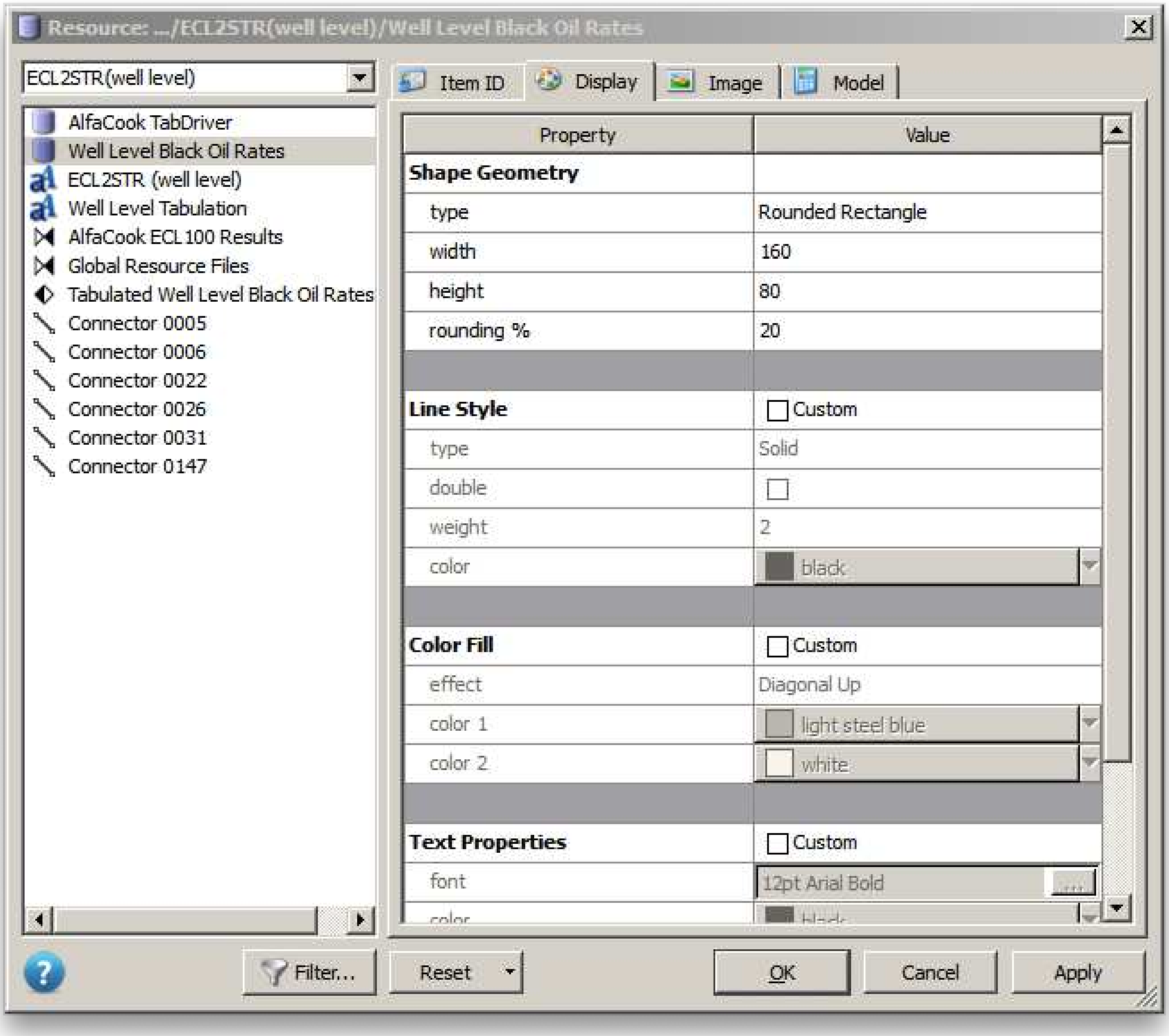
If the Element is Connector then the section means properties of the line. In case of shapes the section provides properties of border line of these shapes.

Color Fill

The section allows to change way how the element is filled by color. You can apply different types of gradient with color selection of begin and end points gradient. Also you can make your elements transparent by selecting appropriate *Transparency* property in color selection.

Text Properties

Here you can change font, alignment and text positioning around shape (inside / outside)



Resource: .../ECL2STR(well level)/Well Level Black Oil Rates

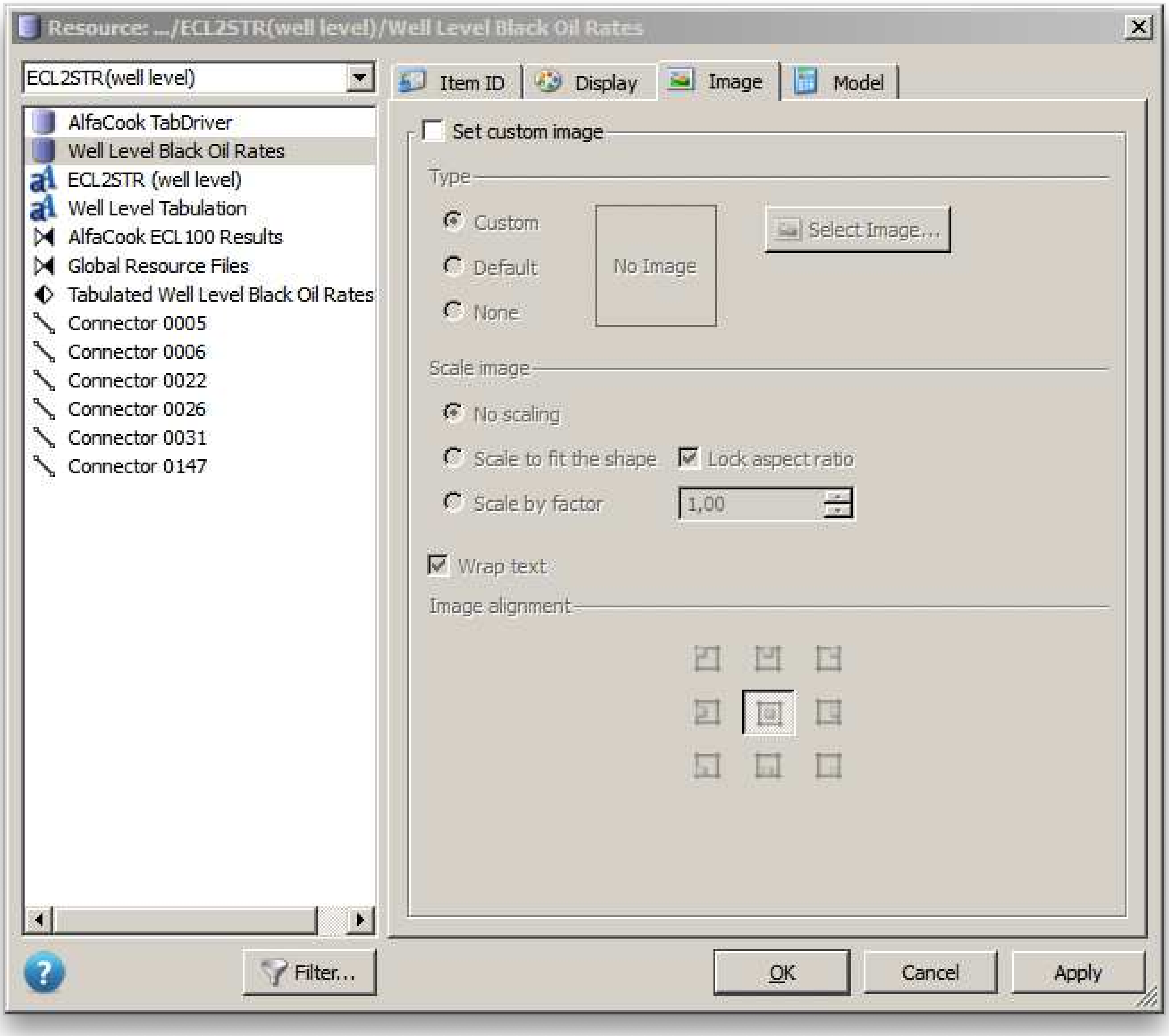
ECL2STR(well level)

- AlfaCook TabDriver
- Well Level Black Oil Rates
- ECL2STR (well level)
- Well Level Tabulation
- AlfaCook ECL100 Results
- Global Resource Files
- Tabulated Well Level Black Oil Rates
- Connector 0005
- Connector 0006
- Connector 0022
- Connector 0026
- Connector 0031
- Connector 0147

Property	Value
Shape Geometry	
type	Rounded Rectangle
width	160
height	80
rounding %	20
Line Style	☐ Custom
type	Solid
double	☐
weight	2
color	black
Color Fill	☐ Custom
effect	Diagonal Up
color 1	light steel blue
color 2	white
Text Properties	☐ Custom
font	12pt Arial Bold
color	black

Image

The page allows to assign picture to the Element and change such properties image placing as *scaling*, *wrapping text* and *alignments*.


Resource: .../ECL2STR(well level)/Well Level Black Oil Rates

ECL2STR(well level)

- AlfaCook TabDriver
- Well Level Black Oil Rates
- ECL2STR (well level)
- Well Level Tabulation
- AlfaCook ECL100 Results
- Global Resource Files
- Tabulated Well Level Black Oil Rates
- Connector 0005
- Connector 0006
- Connector 0022
- Connector 0026
- Connector 0031
- Connector 0147

Item ID | Display | Image | Model

Set custom image: ☐

Type: [✓ Custom] Default None — No Image — Select Image...

Scale image: [✓ No scaling] Scale to fit the shape — Lock aspect ratio ✓ — Scale by factor 1,00

Wrap text ✓

Image alignment

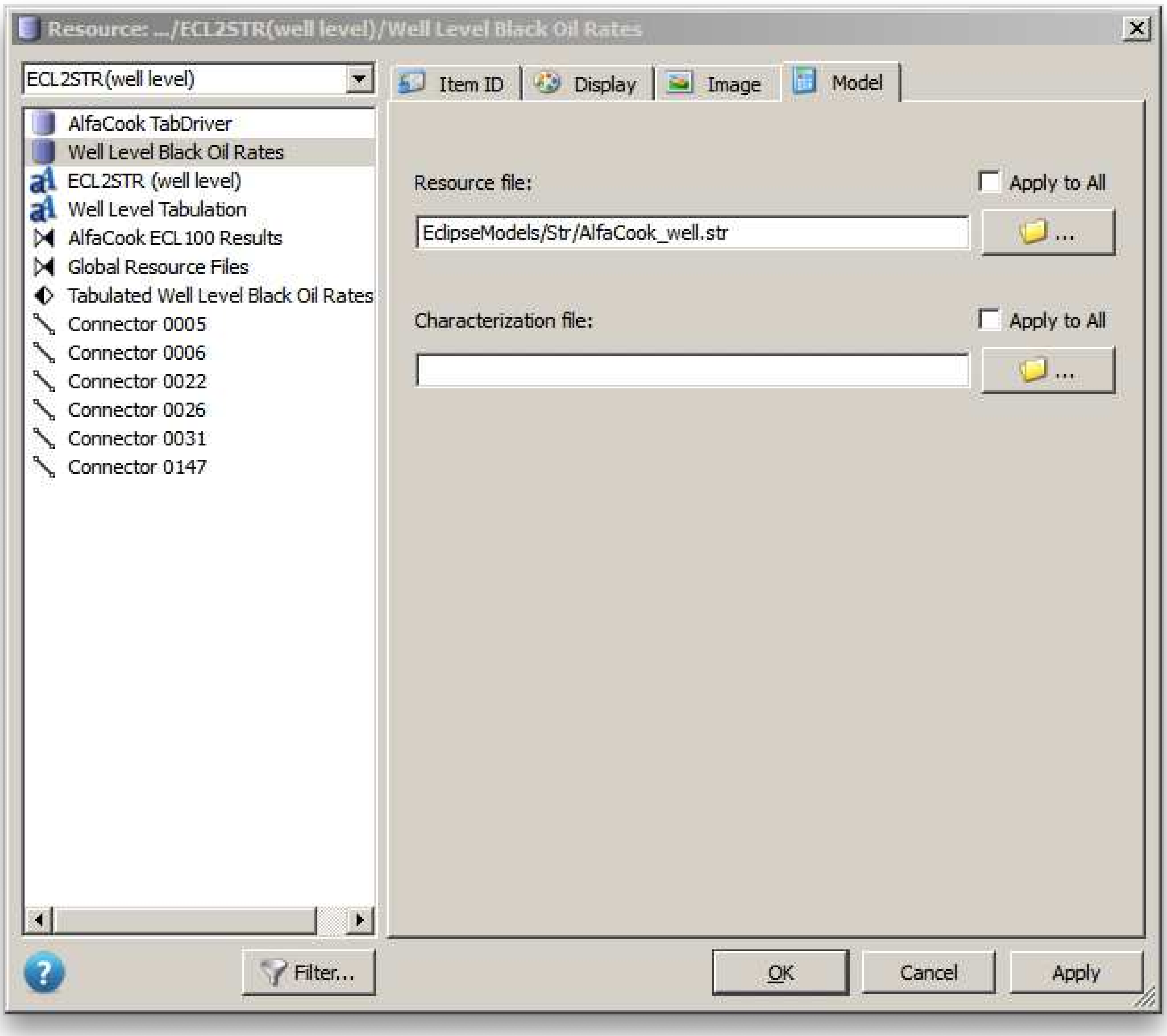
Model

Model page has logic properties of project model depending of type of Element,

For example, in case of *Resource* they are *File* and *Characterization file*

If the Element is *Scripter*, then page has same interface as dialog for editing scripts.

Composites have settings to allow or not the *Parallel* running .


Resource: .../ECL2STR(well level)/Well Level Black Oil Rates

ECL2STR(well level)

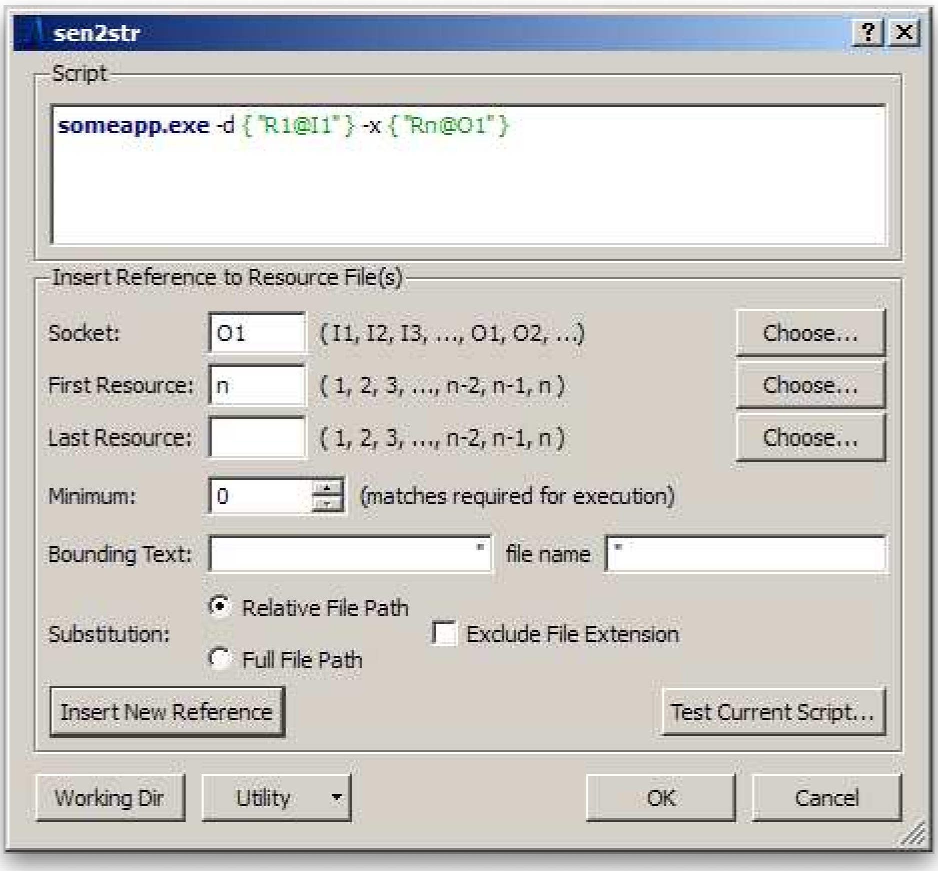
- AlfaCook TabDriver
- Well Level Black Oil Rates
- ECL2STR (well level)
- Well Level Tabulation
- AlfaCook ECL100 Results
- Global Resource Files
- Tabulated Well Level Black Oil Rates
- Connector 0005
- Connector 0006
- Connector 0022
- Connector 0026
- Connector 0031
- Connector 0147

Resource file: ☐ Apply to All

EclipseModels/Str/AlfaCook_well.str

Characterization file: ☐ Apply to All

Editing Scripters



```
someapp.exe -d { "R1@I1" } -x { "Rn@O1" }
```

Insert Reference to Resource File(s)

Socket: O1 (I1, I2, I3, ..., O1, O2, ...)

First Resource: n (1, 2, 3, ..., n-2, n-1, n)

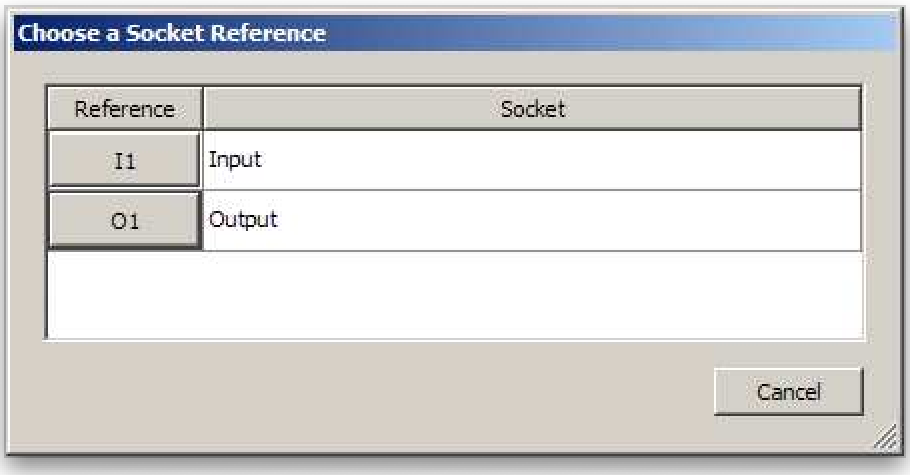
Last Resource: (1, 2, 3, ..., n-2, n-1, n)

Minimum: 0 (matches required for execution)

Bounding Text: " file name "

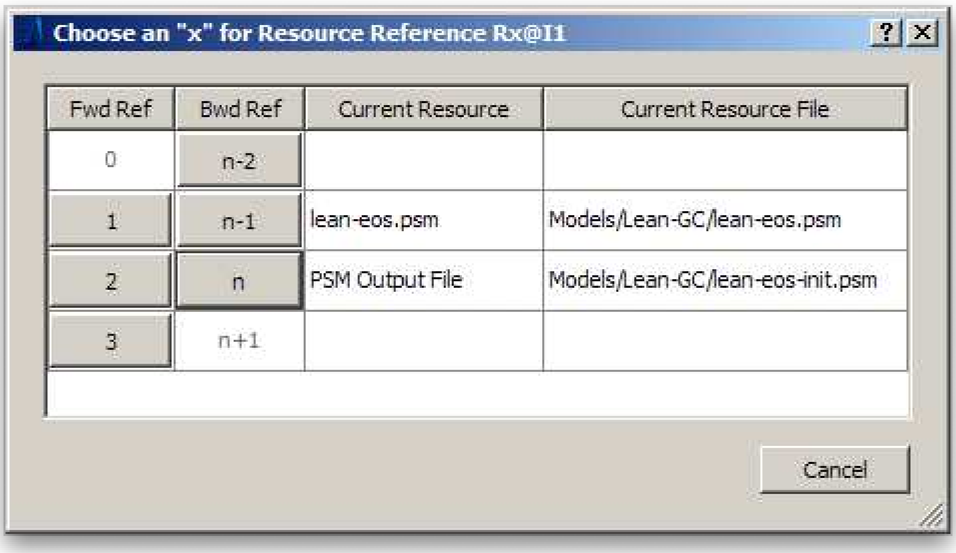
Substitution: [✓ Relative File Path] Full File Path; Exclude File Extension [ ]

Now to select a socket or resource reference (if you don't want to just type in the reference manually) you need click *Choose...* button. If you're looking for a socket, you'll get a new dialog like this:



Reference	Socket
I1	Input
O1	Output

Where you can just click on any of the shaded buttons to select the desired reference. If you're looking for a resource, you'll get a dialog like this:



Fwd Ref	Bwd Ref	Current Resource	Current Resource File
0	n-2		
1	n-1	lean-eos.psm	Models/Lean-GC/lean-eos.psm
2	n	PSM Output File	Models/Lean-GC/lean-eos-init.psm
3	n+1		

Where, again, you can click on any of the shaded buttons. If there's a resource currently connected to a given reference location, you'll see it and its file to the right of the reference. In the example above, *PSM Output File* can be referred to as either R2@I1 or Rn@I1 (because n = 2, as you can tell from the side-by-side buttons, and the socket is I1, as you can tell from the title bar).

Referring back to the "Edit Script" dialog, you can see that there's "Substitution" option to choose either Relative or Full File Paths. So, now you are able to refer to the resource connected to the socket, for example, as either {R1@O1} (for a Relative Resource file path substitution) or as {F1@O1} (for a Full resource File path substitution), and the dialog will generate whichever syntax you request.

References Syntax

```
{<prefix>Rx:y@Sz<suffix>}(m)
```

where *S* will actually be either *I* (for input) or *O* (for output), *z* will be an integer socket number, *x* and *y* will be either integer resource numbers or else of the form *n* or *n-i* (where *i* is an integer), *<prefix>* and *<suffix>* will be optional text strings (restricted only to not containing braces), and *m* will be the minimum number of matching files required for execution. The *:y* and *(m)* parts will also be optional (*y* will default to *x* and *m* will default to 0).

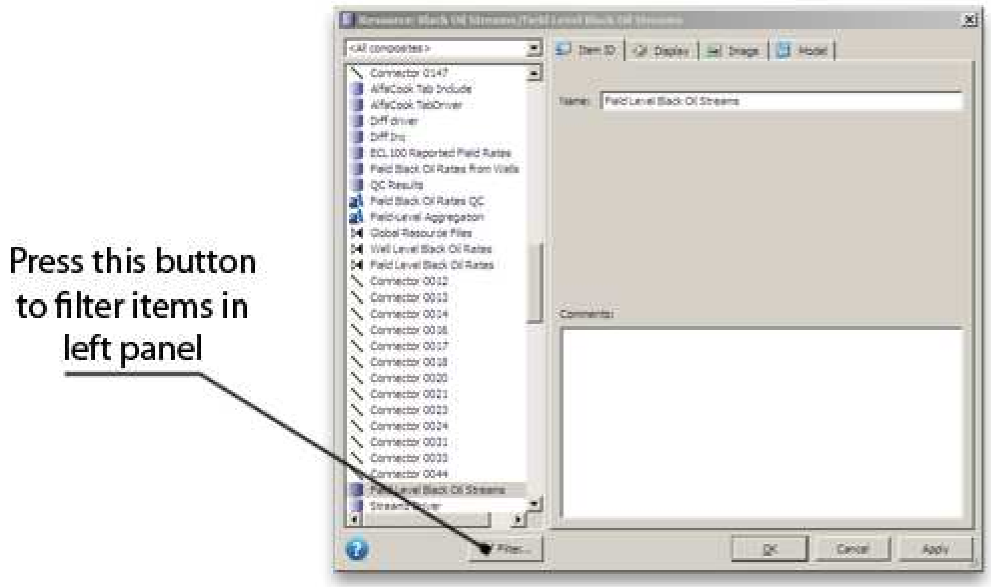
Item Filter Dialog

Pipe-It application has functionality to select Elements of project by some criteria for next operations with these Elements. There are two possibilities to use Filtering for selecting Elements.

First is possibility to select Elements on current Canvas by choosing:

Edit > Select By Filter...

Also when you re working in *Item Properties* dialog and would like to select multiple Elements in left panel. You can press *Filter...*button at bottom and get same dialog to filtering Elements and then making selection of these Elements:

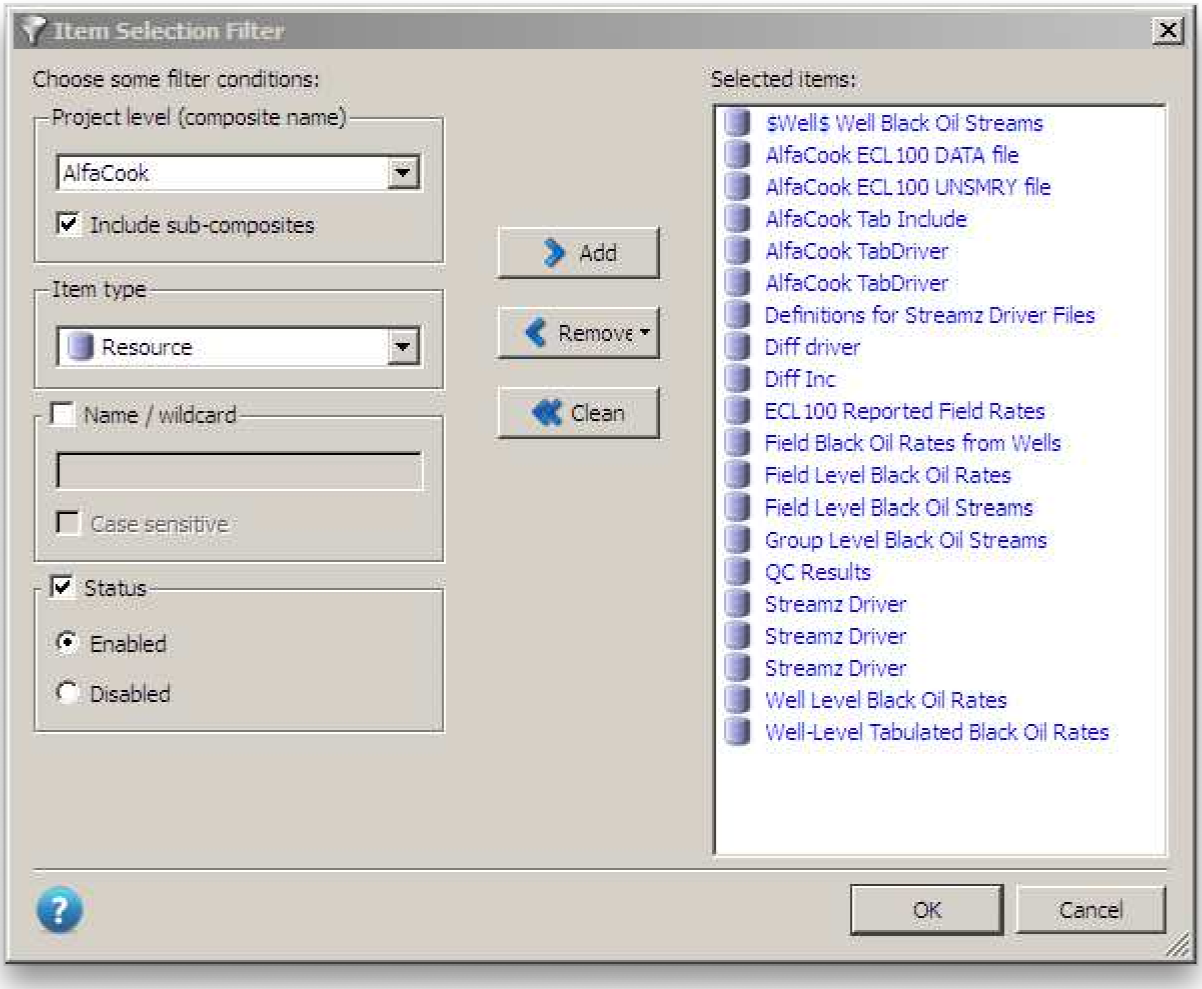


Name: Field Level Black Oil Streams

Connector 0147
AlfaCook Tab Include
AlfaCook TabDriver
Diff driver
Diff Inc
ECL 100 Reported Field Rates
Field Black Oil Rates from Wells
QC Results
Field Black Oil Rates QC
Field-Level Aggregation
Global Resource Files
Well Level Black Oil Rates
Field Level Black Oil Rates
Connector 0012
Connector 0013
Connector 0014
Connector 0016
Connector 0017
Connector 0018
Connector 0020
Connector 0021
Connector 0023
Connector 0024
Connector 0031
Connector 0033
Connector 0044
Field Level Black Oil Streams
Streamz Driver

Comments:

Let's consider the filtering in details:



Choose some filter conditions:

Project level (composite name): AlfaCook

[✓] Include sub-composites

Item type: Resource

[ ] Name / wildcard

[ ] Case sensitive

[✓] Status

[✓ Enabled] Disabled

Selected items:

$Wells Well Black Oil Streams
AlfaCook ECL100 DATA file
AlfaCook ECL100 UNSMRY file
AlfaCook Tab Include
AlfaCook TabDriver
AlfaCook TabDriver
Definitions for Streamz Driver Files
Diff driver
Diff Inc
ECL100 Reported Field Rates
Field Black Oil Rates from Wells
Field Level Black Oil Rates
Field Level Black Oil Streams
Group Level Black Oil Streams
QC Results
Streamz Driver
Streamz Driver
Streamz Driver
Well Level Black Oil Rates
Well-Level Tabulated Black Oil Rates

The dialog consists of two left and right sides, where left side defines filtering conditions to defines which Elements you would like to select. Right side shows the resulting items.

Between these two sides, placed 3 buttons that allows to add or remove Elements from resulting list.

To get a list of resulting Elements you need to define filtering conditions:

1. Select *Composite* (with sub-composites if enabled)

2. Make choice for *Item Type* if needed
3. Filter by *Name/Wildcard* if needed
4. Filter by Status is needed

Then press *Add* button to "move" items into resulting list.

You can *Add* or *Remove* to / from resulting list of Element several times until you get list that meets your requirements.

Finally you can filter resulting list by selecting several Elements and Pressing *Remove > Remove Selected by mouse*

When you see that list is completed, press *OK* and the resulting list will be provided to *Item Properties* dialog or these Elements will be selected on canvas - depending where you called for filtering.

Double clicking

Resources

Double clicking Resources launches the operating system (OS) association for the file type. It is very convenient if you have defined association to process the file or you have Text Editor bound to the file type.

Remember that with **F4** or *Context Menu > Edit Item* you can open the resource file in the editor of your preference.

See Also: Text Editor Preferences for details about defining default *Text Editor*

Scripters

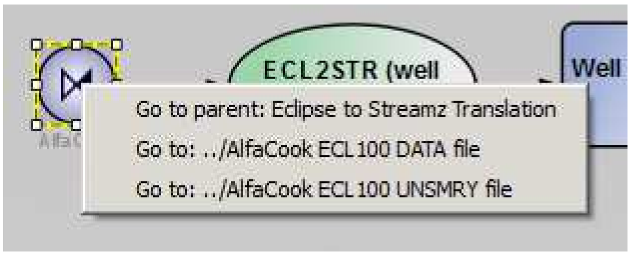
Double clicking Scripter Processes opens *Edit Script* dialog

Composites

Double clicking the Composite leads you into the composite itself.

Composite Sockets

Double clicking the composite Socket opens a menu that allows you to navigate to parent composite or connected Resources or Scripters to locate them:

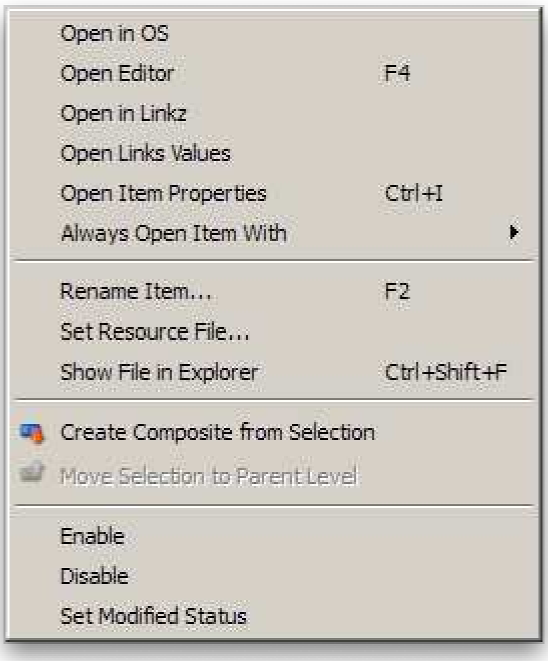

ECL2STR (well
Well
Go to parent: Eclipse to Streamz Translation
Go to: ../AlfaCook ECL100 DATA file
Go to: ../AlfaCook ECL100 UNSMRY file

Other Elements

Double clicking all other elements on the canvas leads you into the item properties. If you try to double click the canvas nothing will happen.

Right-click Context Menu

The right click menu is available when you right click on the canvas or when you right click on an element on the canvas. The options not available for a particular element are grayed out. The menu will give you different opportunities explained below.



Open

Context menu provides options for opening the item in several ways

Open in OS (Resources only)

This item runs the OS association for the File which is assigned to the Resource. You can specify the the association in your system preferences or in your favorite file manager like Total Explorer

Open Editor (Resources only)

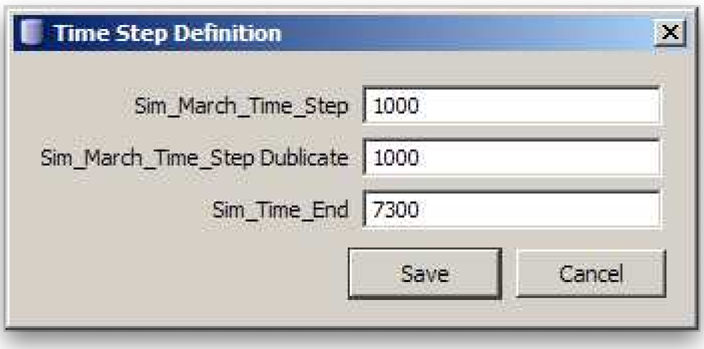
This one will run application which is set as Editor for the type of File which is assigned to the Resource. By default this is *notepad* window application (or *TextEdit* for Mac) and *paint* for image files. You can easily change these settings in the Preferences Window, *Text Editor* tab (you can access it with *menu -> Edit -> Preferences...*) Remember that you have keyboard shortcut F4 for the item.

Open in Linkz (Resources only)

Linkz has own file viewer optimized for showing and navigating huge files for defining links to values encoded in these files. This menu item provide fast access to the Linkz File Viewer to define new links or edit existing ones.

Open Linzk Values (Resources only)

For files which have already defined links to some values, the menu item shows you simple form for editing these values:



Sim_March_Time_Step	1000
Sim_March_Time_Step Dublicate	1000
Sim_Time_End	7300

White Save button new changed values are written to the file.

See also: Linkz

Open Item properties

When you choose this option a menu will appear. On the left in the menu you may pick the elements you want to alter, either by picking them manually, or by using the **filtering function**.

See Item Properties for details

Create composite from Selection, and move to parent level

The function Create composite from Selection wraps all that you have selected together into a composite. If things are connected, it takes care of the manifolds and distributors for connecting the elements inside the composites to the elements outside.

If you choose move to parent level, it will move the element you right clicked on up one level in the hierarchy.

You can also just pick an element and drag it into a composite element if you want to. The software will take care of the connections for you, but you may want to change the appearance to make it look the way you want.

Edit sockets (Scripters, Composites)

Here you can see what the different elements are connected with (Connector names). This can be useful when you have elements inside a composite connected to another element outside the composite. Also you will get a question about this if you have one resource outside a composite, and two input sockets inside the composite. If you then make a connector from the resource to the composite, you will be asked to which socket you want the resource to be connected.

Rename item

This gives you the opportunity to rename the element.

Set Resource file

Here one may associate a resource to its originating source of information. This is typically a text file. This option is only available for the resource element.

Set Script Command

This option is only available for a Scripter. This allows the specification of the script to be executed when this process is launched by the runner. This is done by setting the command in the box that appears when ones choose the Set script command option.

For utilities provided with Pipe-It there is another, maybe simpler way to set this command. Choose the option Item properties by right-clicking on the process and click the Model tab. Here a button called “Open Utility Editor” appears. Click on that and one may choose the most used utilities. This way much of the work writing the script command is done automatically, but it is only an option for the utilities included with the software.

Streamz is currently launched via the Scripter and the proper driver file is set as the first argument to the Script command. To obtain a log file of the Streamz run it is also typical to specify the log file as the second argument. Hence the Script command specified becomes:

At present, it may be a clever move to connect a resource to the Scripter process that only is associated with driver file it self. (DriverFile.stz file in example above) This will make the program detect eventual changes in the driver file automatically and rerun from the first changed element. For an example, take a look at the picture below.

See Using Utility Method for details

Enable, Disable and Set Modified Status

Enable let you enable an element that you previously have disabled. Disabling an element can be done if one does not want it to be included in the project execution at run time. Although visually the element is part of the project the runner will exclude it when determining the execution order and that actual run.

Pipe-It keeps track of the “up-to-date” status of each and every element and runs only those that are not up-to-date. This could be critical in huge projects with large run times of individual parts. Set Modified Status is an option that is used to force the model to be rerun from a specific point. Even though a resource is up-to-date the user can force Pipe-It to include the resource in the run.

Define File Variable

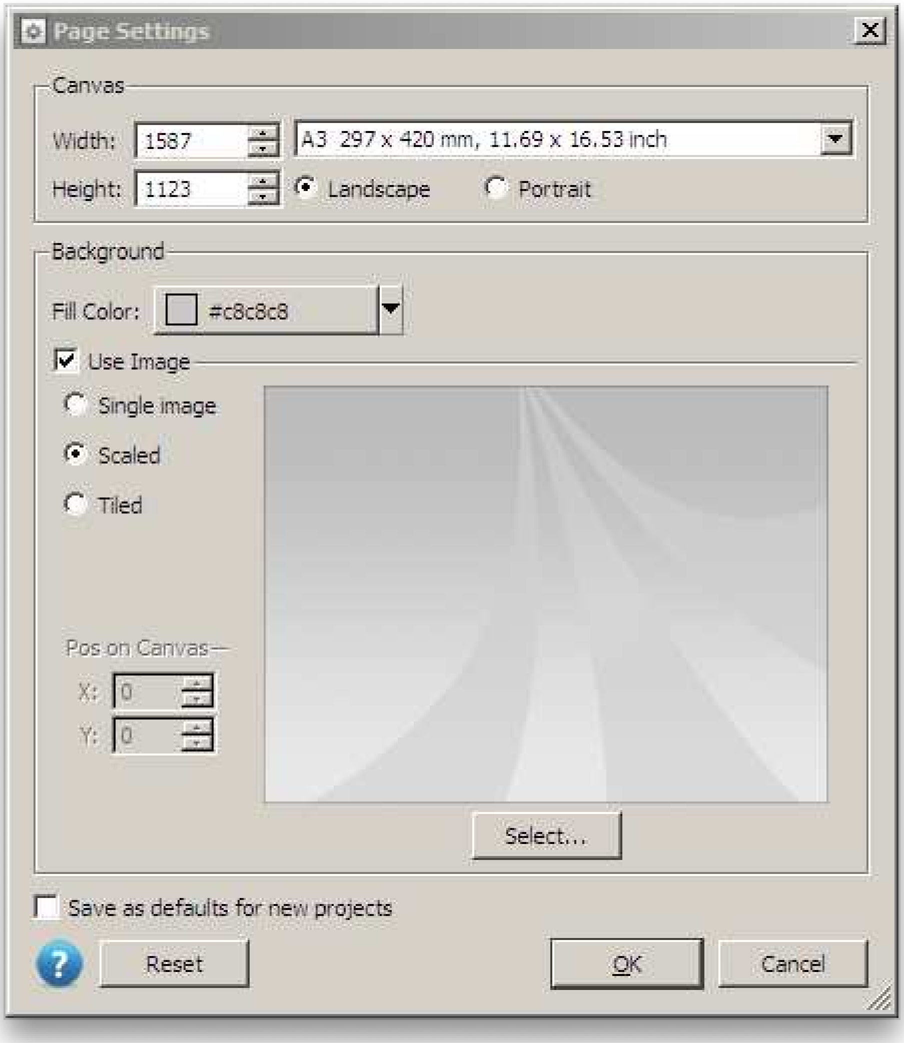
This is mainly made to set variables, constraints, and objects for the optimizer, but it can also be used to view text files. This may be useful to check that one has connected the correct file to a resource. Please note that one can not edit a file with this feature.

Canvas Settings

To view or change settings of canvas of current composite:

Choose *Edit > Canvas Settings*

The dialog allows to change canvas size using predefined sizes and select the way of filling background of canvas. By default all new projects get Petrostreamz background image.


Page Settings

Canvas
Width: 1587 — A3 297 x 420 mm, 11.69 x 16.53 inch
Height: 1123 — [✓ Landscape] Portrait

Background
Fill Color: #c8c8c8
[✓] Use Image
Single image
[✓ Scaled]
Tiled
Pos on Canvas
X: 0
Y: 0

You can fill background with color or image which can fill canvas in next 3 ways:

- *Single Image* + you can define position on canvas
- *Scaled* to fill whole canvas
- *Tiled*

Browse button allows to select image from Project images.

By checking *Save as default for new projects* your new settings will be saved as default for all new project you will create in Pipe-It next time.

Save As Dialog

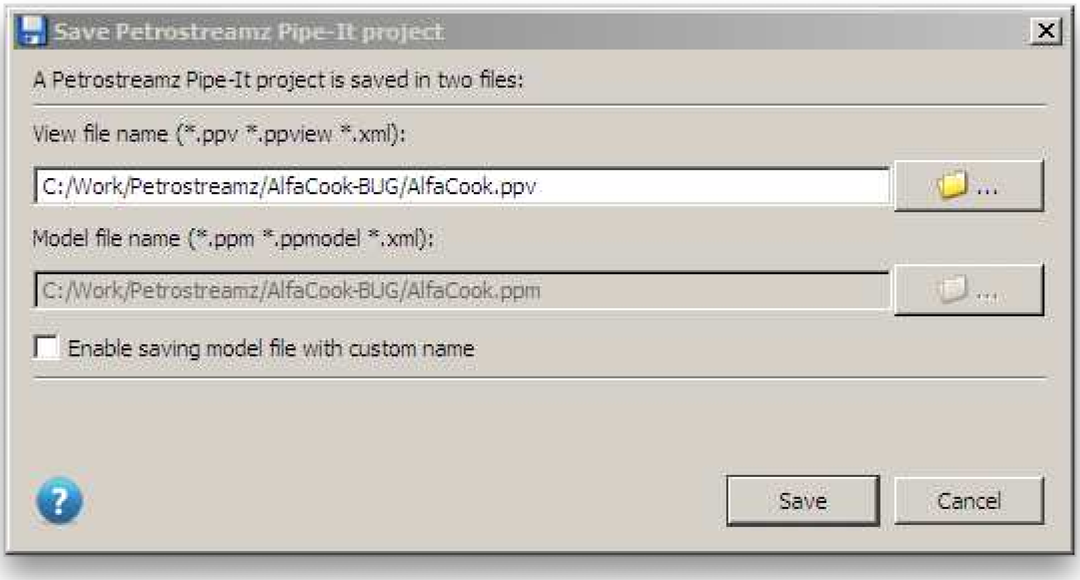
Pipe-It projects consist of two files which define *logic* and *view*.

Logic is defines in *ppm* file. View in *ppv*.

When you save a project you can see that Pipe-It prompt you for view file name. *Logic* file name is generated basing on your choice for file name of ppv file.

Though you can *Enable saving model file with custom name* and enter different file names for model and view parts of project.

Remember that *Save As* stores only model and view files and do not do anything regarding files bound to Resources. So if you save your project under another location, you have to copy all supplementary files.


Save Petrostreamz Pipe-It project

A Petrostreamz Pipe-It project is saved in two files:

View file name (*.ppv *.ppview *.xml):

C:/Work/Petrostreamz/AlfaCook-BUG/AlfaCook.ppv

Model file name (*.ppm *.ppmodel *.xml):

C:/Work/Petrostreamz/AlfaCook-BUG/AlfaCook.ppm

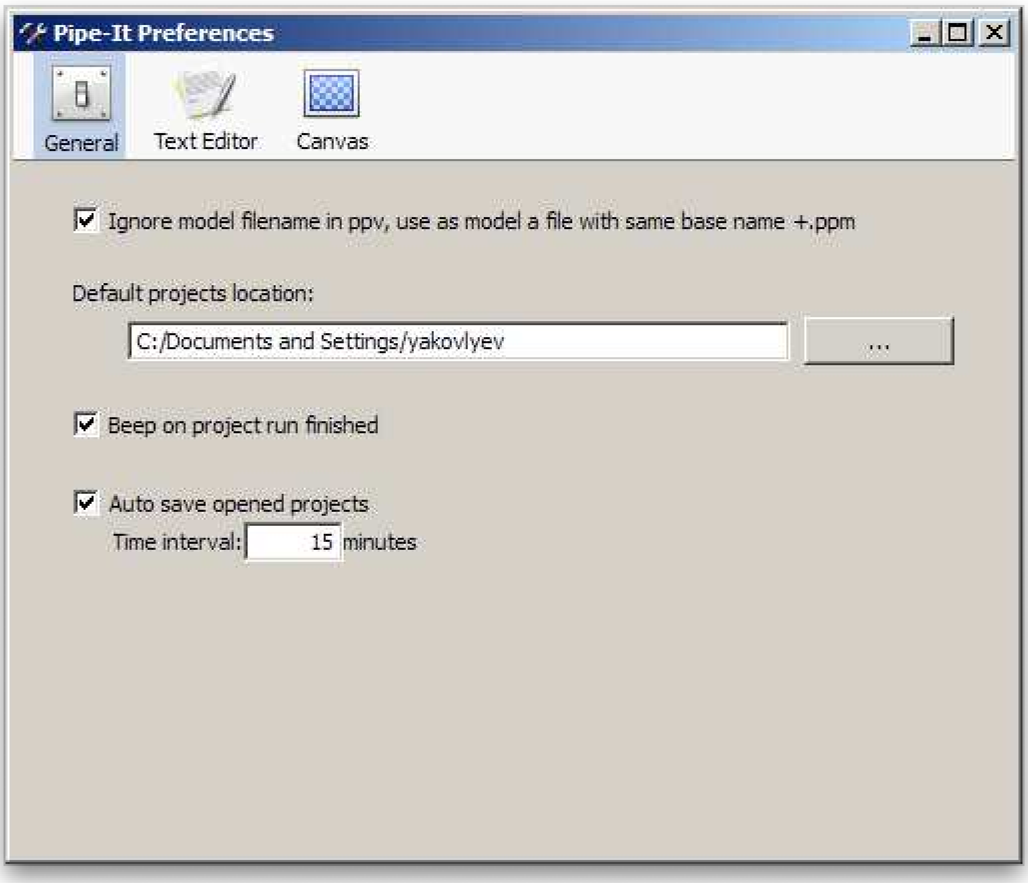
☐ Enable saving model file with custom name

Using default naming for Model file

You can have the default naming schema for model ppm files when model file uses same base file name as view ppv file for all operations of Open / Save files.

In such case you can rename your pair of ppm / ppv files without opening them in Pipe-It and resaving with new names.

To enable this option you need to open *Preferences Window*, navigate to *General* tab and enable *Ignore model filename in ppv, use as model a file with same base name + .ppm*


Pipe-It Preferences

☑ Ignore model filename in ppv, use as model a file with same base name +.ppm

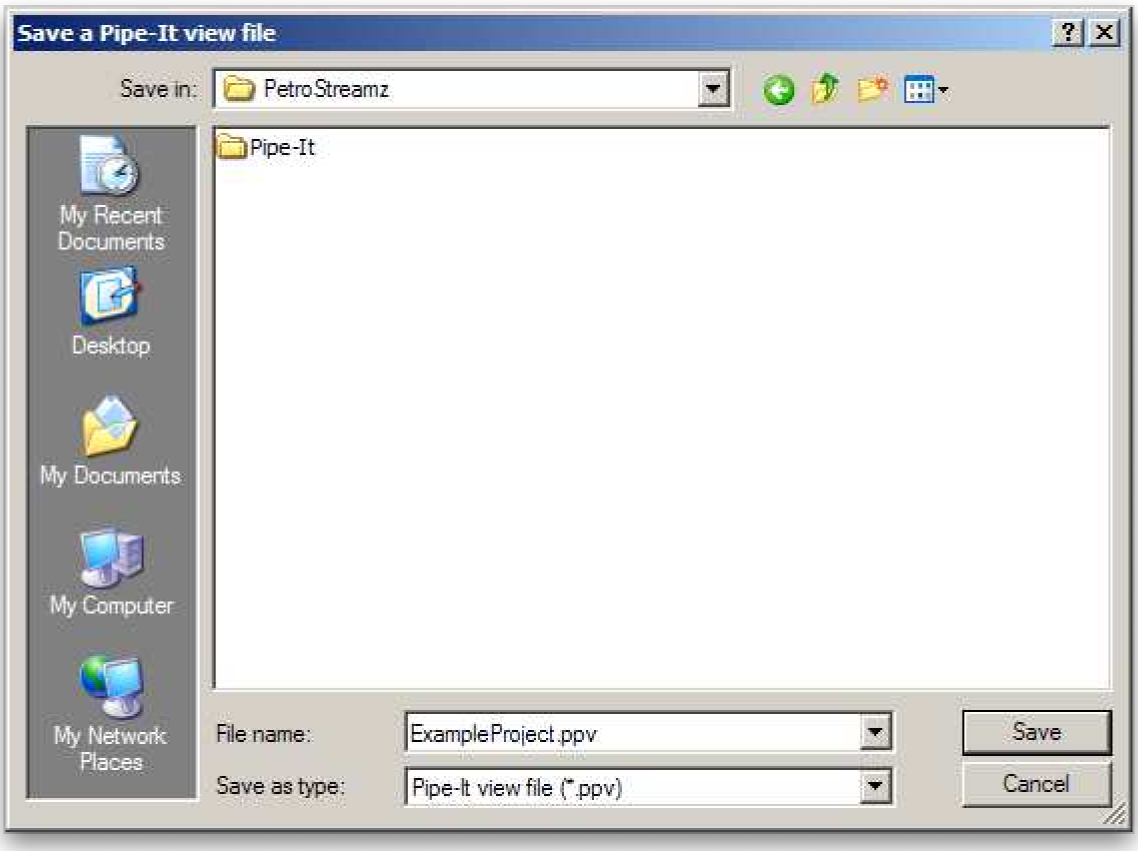
Default projects location: C:/Documents and Settings/yakovlyev

☑ Beep on project run finished

☑ Auto save opened projects

Time interval: 15 minutes

Now, you work only with view (ppv) files and Pipe-It assume default naming schema for model files. With *Save As* action you will see standard operation system file dialog, like this one:


Save a Pipe-It view file

Save in: PetroStreamz

Pipe-It

File name: ExampleProject.ppv

Save as type: Pipe-It view file (*.ppv)

Designing Pipe-It projects

Pipe-It projects consists of many components. The Pipe-It Project View (*.ppv) file store the visual content of the project. The Pipe-It Project Model (*.ppm) file is most important and stores the logical model describing the all the elements in the project their connections, their grouping into composites, the files associated with each resource, the command lines of each scripter etc. The Pipe-It Project Linkz (*.ppl) file stores all the variables collected from all text files that are part of the project, their current values and algorithms to locate them in later runs.

The location of these 3 files is called the root folder of the project. Most other files mentioned under can be located anywhere on the disk. However, to make the project portable it is recommended that all such files are located at or under the root folder.

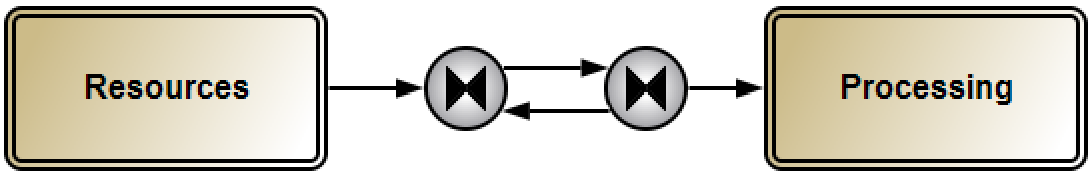
If the project contains optimization, one or a series of Pipe-It Project Optimization (*.ppo) files are also present. They can be located anywhere.

A complete project will also contain data files that are connected to each resource. All files connected to initiating resources (those that are not downstream of any process) are also mandatory for the proper execution of the project. These files are typically located at the root or in folders under the root.

Designing a Pipe-It project requires some planning. Simple projects might contain only a handful of data files linked to resources and these can live in the project root folder. But real life projects contain hundreds if not thousands of files. It makes sense to organize these files in a sensible folder hierarchy. The project within Pipe-It should also follow the folder structure using composites and sub-composites as far as possible. This helps in cloning of composites in case a new instance of the functionality encapsulated in the composite is to be created. Pipe-It estimates the % of files that resides within a particular sub-folder of the project and offers to the user the possibility of duplicating files. This is really a powerful option as all files, their connections to resources, their Linkz variables, and all process command lines are cloned.

Making order of the hundreds and thousand files is another reason for collecting related files together.

Use Recycle Loops



Pipe-It runs projects with processing only outdated parts. It follows all resources in project, check modification dates of files and decide which part of project should be executed. It prevents from multiple execution of same processing to generate same data.

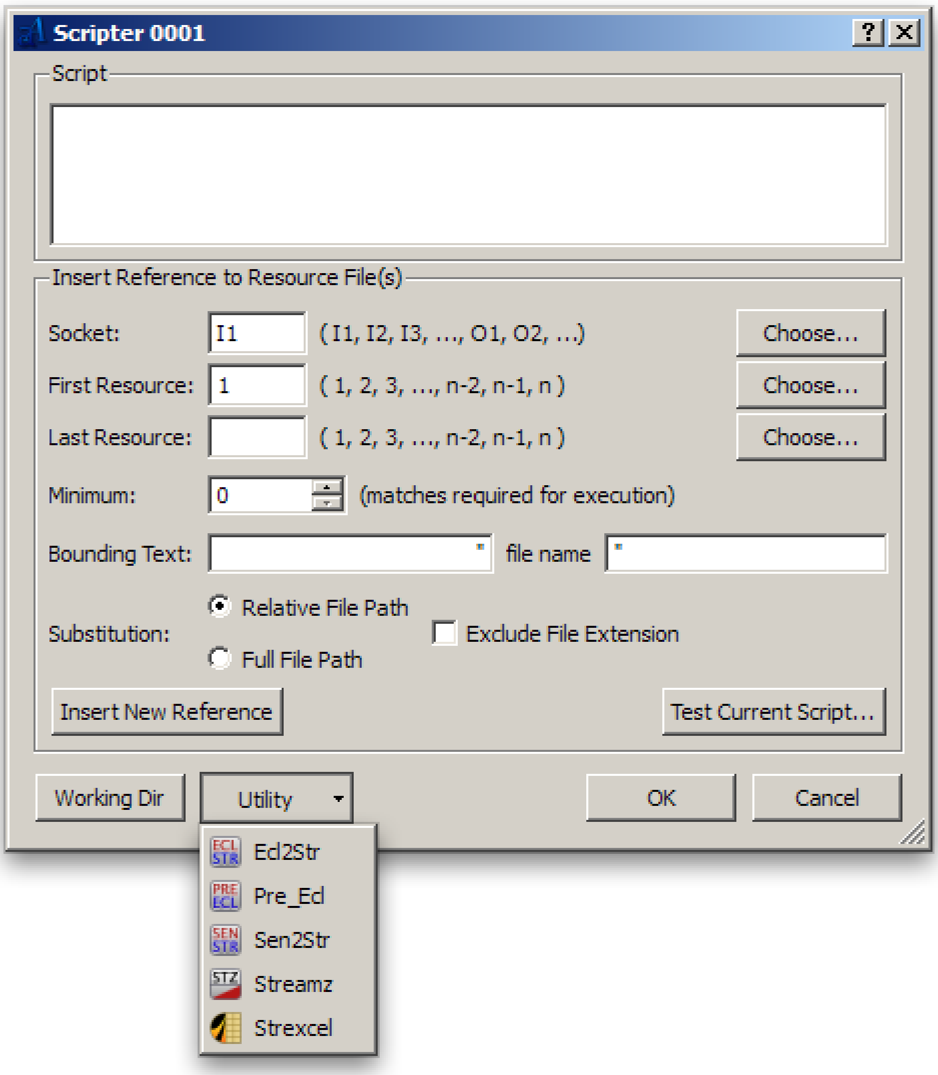
But there are cases when it is no connection to modified files and some composite or part of the project should be executed each time you press Run button. For example, imagine a project which fetch input data from some network services (web services).

To resolve the situation and have execution of project or composite each time you run the project you can design the *Recycle Loop*. Everything within and downstream of a recycle loop is guaranteed to run every time the project is run (because the loop will always outdate itself).

Utility Method Dialog

In process of designing projects you may find that you use similar script commands repeatedly. Usually they only differ in filenames and few other arguments. To make the script writing easier and some-what automated, raher than writing it from scratch each time, you can use *Utility* method to use templates applicable to your project.

Access to Utilities is available from *Edit Script* dialog by pressing *Utility* button:



Socket: I1 (I1, I2, I3, ..., O1, O2, ...)

First Resource: 1 (1, 2, 3, ..., n-2, n-1, n)

Last Resource: (1, 2, 3, ..., n-2, n-1, n)

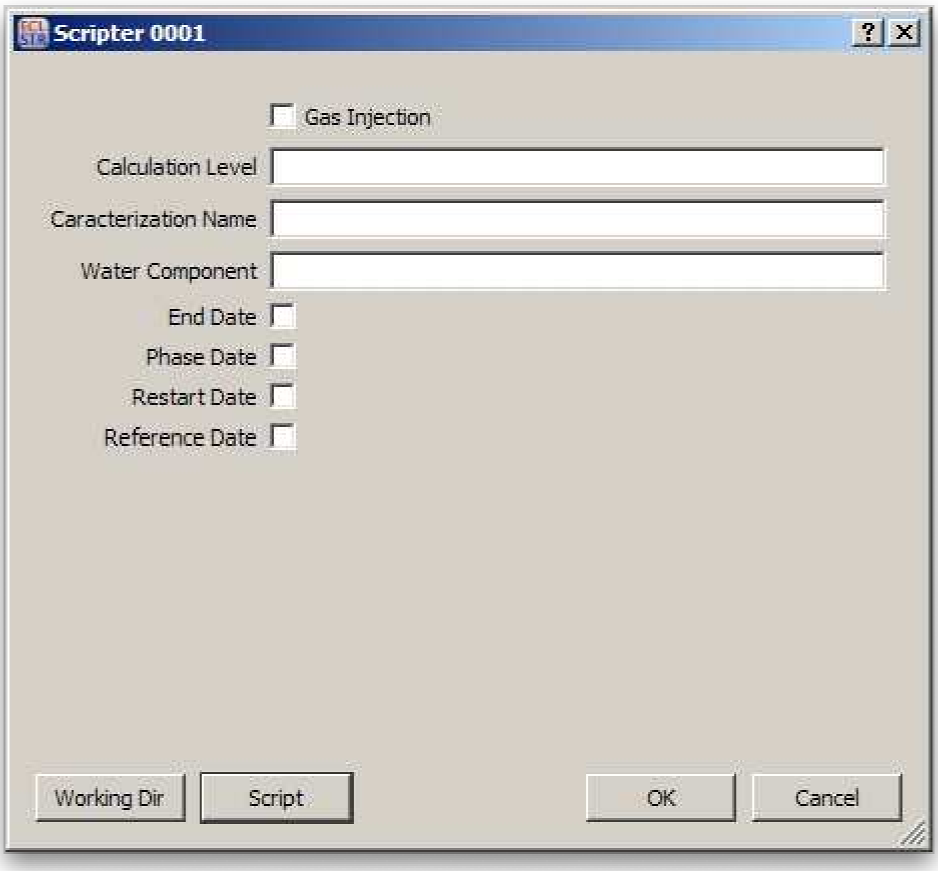
Minimum: 0 (matches required for execution)

Bounding Text: " file name "

Substitution: [✓ Relative File Path] Full File Path

Exclude File Extension

With your choice of an utility the dialog switches to generic dialog-box with fields specific to the chosen utility. As an example you see here the dialog for entering values of *Ecl2Str* utility.


Scripter 0001

- [ ] Gas Injection

Calculation Level	
Caracterization Name	
Water Component	
End Date	☐
Phase Date	☐
Restart Date	☐
Reference Date	☐

The *Utility Method Dialog* provides very simple way to create new or change current script by using predefined set of possible scripts.

Rather than to require users to remember the exact syntax of the command line for a particular frequently-used Scripter, the user is presented with a graphical interface where the proper options are chosen. These chosen options end up as values to arguments in proper syntax. All arguments requiring file names are implemented as named-sockets and user is presented with the options to choose the socket when resources are connected to this scripter.

Pipe-It is distributed with the following utilities and the associated executables:

- Ecl2Str
- Pre_Ecl
- Sen2Str
- Streamz
- Strexcel

Designing your utilities

Utilities provide a fast and elegant means of incorporating user or project specific programs or tools into Pipe-It. These would not be distributed with Pipe-It but can be made to appear an integral part of Pipe-It without any need of going to the developers of Pipe-It. The engineer or consultant (or very quickly the client themselves) can obtain an icon of the utility on the Pipe-It toolbar and use it repeatedly as if it came with Pipe-It.

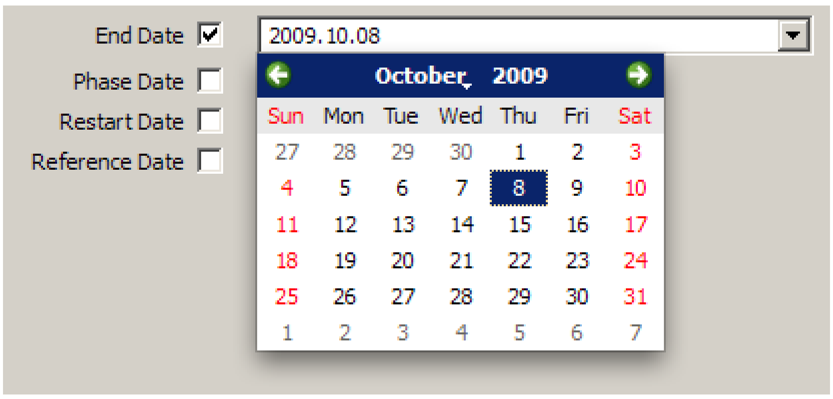
Utilities are defined in xml files with *.ppu* extension. On startup Pipe-It looks for files with the extension in same folder where Pipe-It.exe is located + *Utilities* subfolder and load them for the use in Scripters. Also if you used some utilities in your Scripters then appropriate utilities are also saved in your Project files - so you can distribute your project easily - double click on these Scripters show correct utility loaded from your project.

The xml file (text file) of one of the distributed utilities is shown below to provide an example of its simple nature:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PetrostreamzUtilities>
   <Utility name="Ecl2Str">
       <Icon>Ecl2Str.png</Icon>
       <Path>Ecl2Str.exe</Path>
       <Argument name="Gas Injection" type="Check" prefix="-g" />
       <Argument name="Calculation Level" type="String" prefix="-l " />
       <Argument name="Caracterization Name" type="String" prefix="-c " />
       <Argument name="Water Component" type="String" prefix="-w " />
       <Argument name="End Date" type="Date" prefix="-e " />
       <Argument name="Phase Date" type="Date" format="yyyyMMdd" prefix="-a " />
       <Argument name="Restart Date" type="Date" format="yyyyMMdd" prefix="-r " />
       <Argument name="Reference Date" type="Date" format="yyyyMMdd" prefix="-f " />
       <Argument name="Input .DAT" >{ -d "R1@I1" }</Argument>
       <Argument name="Input .SUM" >{ -s "R1@I2" }</Argument>
       <Argument name="Input .PVT" >{ -p "R1@I3" }</Argument>
       <Argument name="Input .PSM" >{ -x "R1@I4" }</Argument>
       <Argument name="Output .STR">{ -o "R1@O1" }</Argument>
       <Socket type="ProcessInput" name=".DAT File" />
       <Socket type="ProcessInput" name=".SUM File" />
       <Socket type="ProcessInput" name=".PVT File" />
       <Socket type="ProcessInput" name=".PSM File" />
       <Socket type="ProcessInput" name="Other Input Dependencies" />
       <Socket type="ProcessOutput" name=".STR File" />
       <Socket type="ProcessOutput" name="Other Output Dependencies" />
   </Utility>
</PetrostreamzUtilities>
```

This Xml defines *Ecl2Str* utility based on *Ecl2Str.exe* executable with following options:

- *Gas Injection* which has *Check* type - when it checked on, prefix "-g" is added to command line
- *Calculation Level* which is just text field and if non-empty, is added with prefix "-l"
- *End Date* which has *Date* type, so in dialog you will see a checkmark to make it non empty and field to select date

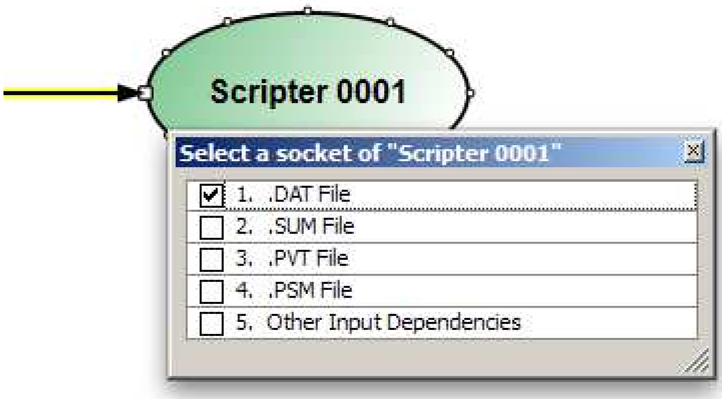


Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Also you can see additional arguments without Type. These arguments added to script by default. Usually they are used for defining references (like in this example).

See Also: Scripters for references syntax

Next in list are sockets definitions. There are two types of sockets suitable for Scripters - *ProcessInput* and *ProcessOutput*. When you connect your scripter with resource you will see these names: (example of connecting inputs)

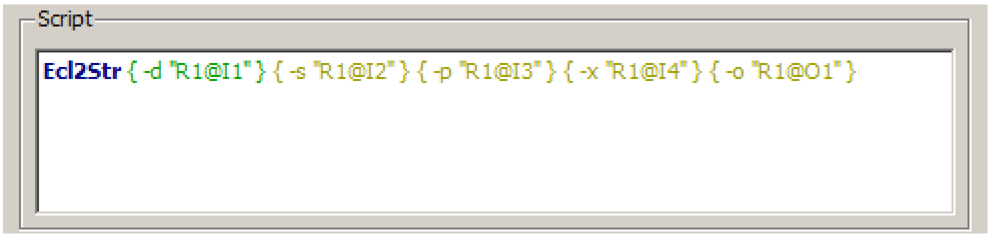


- [✓] 1. .DAT File
- [ ] 2. .SUM File
- [ ] 3. .PVT File
- [ ] 4. .PSM File
- [ ] 5. Other Input Dependencies

If in this example we connect the Resource (file: somefile.txt) to first *Input .DAT* socket, then reference { -d "R1@I1" } will be expanded as

```
-d somefile.txt
```

If in scripter we switch back to Script (with *Script* button at bottom side of dialog) you can see first reference highlighted by green which means connected file there:



```
Ecl2Str { -d "R1@I1" } { -s "R1@I2" } { -p "R1@I3" } { -x "R1@I4" } { -o "R1@O1" }
```

A simple Utility Editor under the Edit menu is also provided to end-user for customizing these utilities. Future versions will provide ability to create new ones from scratch graphically rather than having to write in XML and only imaging its final functionality.

The users are encouraged to to use this very powerful feature of Pipe-It that power users will grow to like and appreciate'''

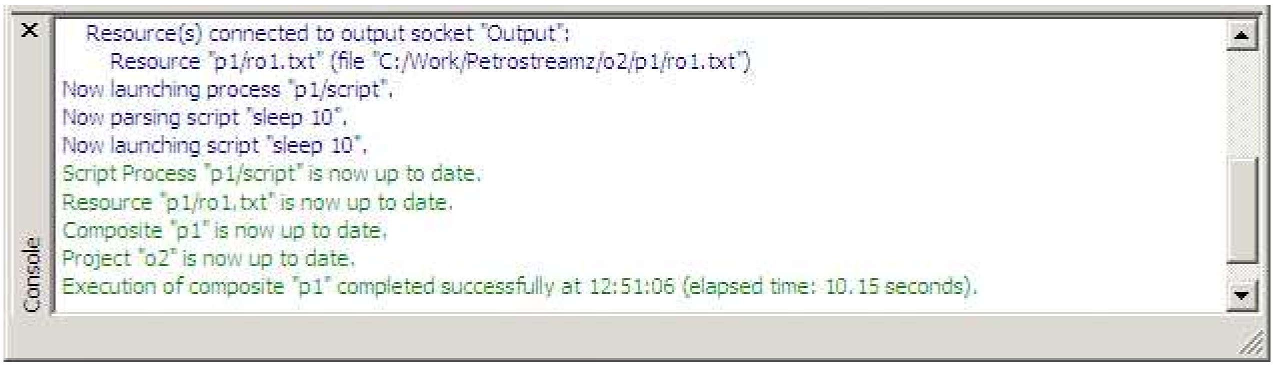
Displaying the model status and results

Once the model is built and the end user pushes the **run** button, Pipe-It will validate the model and return with visual element describing the status of the model. Each resource and process will initially be marked with a (light bulb) indication that the element's status is pending.

- 💡 - means that the Element is under processing now or pending processing.
- ✓ - execution of the Element completed OK.
- 🚩 - Element is broken due to problem in execution.
- 🏳 - processing of the Element is incomplete.

Once the model starts processing the individual elements, the element will either turn the status to OK with a (green tick mark), or if some problem has been encountered, show a (red flag). Notice that the status of the elements will be updated in the Pipe-It Explorer (Tree View) as well.

While running the model, the Runner will output text to the Pipe-It Standard Output (Console) window. Any error or warning messages will be displayed in this window. The Pipe-It Standard Output window will be refreshed for each run and the end user is urged to copy and paste the results into an editor, or save as a date-time-stamped log file to compare the output from two consecutive runs.



```
    Resource(s) connected to output socket "Output":
        Resource "p1/ro1.txt" (file "C:/Work/Petrostreamz/o2/p1/ro1.txt")
Now launching process "p1/script".
Now parsing script "sleep 10".
Now launching script "sleep 10".
Script Process "p1/script" is now up to date.
Resource "p1/ro1.txt" is now up to date.
Composite "p1" is now up to date.
Project "o2" is now up to date.
Execution of composite "p1" completed successfully at 12:51:06 (elapsed time: 10.15 seconds).
```

Exporting and Importing data

Exporting data

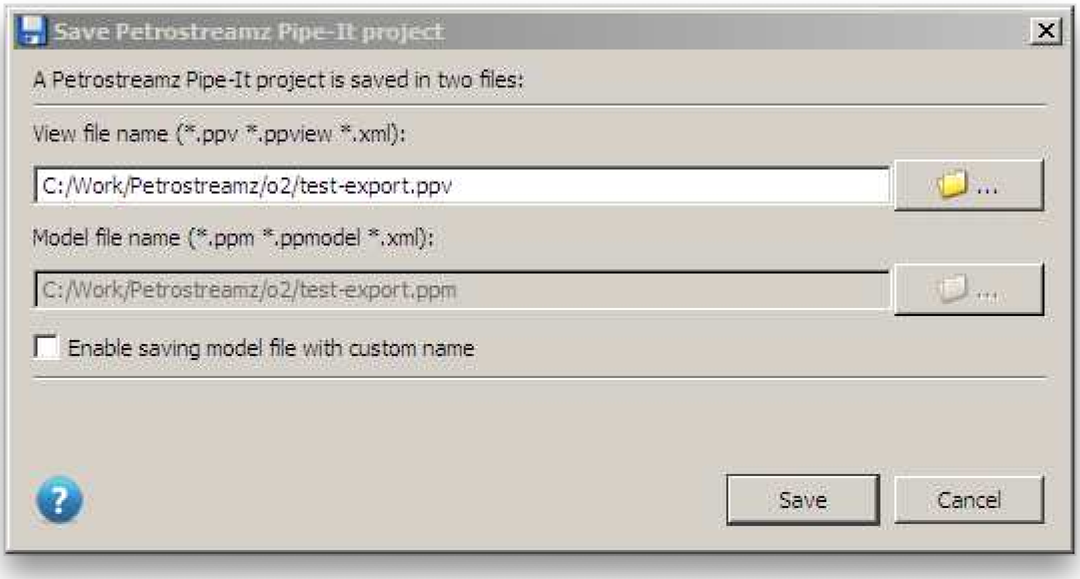
Use *File > Export* menu

Pipe-It has possibility to export the following:

First you can export your current canvas or selection as image file to disk, or directly to system clipboard to paste it into running applications that accept images.

For more details look at: Export Image

Export Composite opens *Save Project* dialog which offer to save current composite (not whole project) as separate project defined by ppm+ppv files:



Save Petrostreamz Pipe-It project

A Petrostreamz Pipe-It project is saved in two files:

View file name (*.ppv *.ppview *.xml):

C:/Work/Petrostreamz/o2/test-export.ppv

Model file name (*.ppm *.ppmodel *.xml):

C:/Work/Petrostreamz/o2/test-export.ppm

☐ Enable saving model file with custom name

Saved ppm+ppv pair can be easily opened as separate project in Pipe-It. If you are going to save the exported project into different than main project place, remember that all associated to Resources files are not copied to separate location and you need to do this manually or keep same location as main project.

Exporting ZIP Package

Pipe-It can export the complete package as a zip file including all folder structure and all files that have been connected to resources. This is a wonderful way to package your project for sending out or exchanging with your co-workers. All files in the project root folder are included but only those files that have resources in the project are included. An option allows you to include or exclude input files only (starting resources without any connection with processes upstream) or result files only (ending resources without any connection to a process downstream).

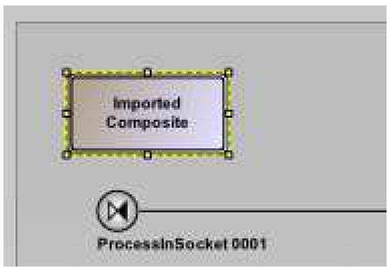
To use it profitably, users should try to always have a resource representation to any file that is important to the project. i.e. there should not be any "hidden" file.

Importing data

Use *File > Import* menu

Import Composite provides way to import separate project as composite into current canvas. It allows to exchange parts beetween projects using Export/Import Composite features.

Import Composite opens *Open PPV File* dialog which allows to locate ppv file and loads it as *Composite* object into current canvas. The composite is inserted into top-left corner of canvas.

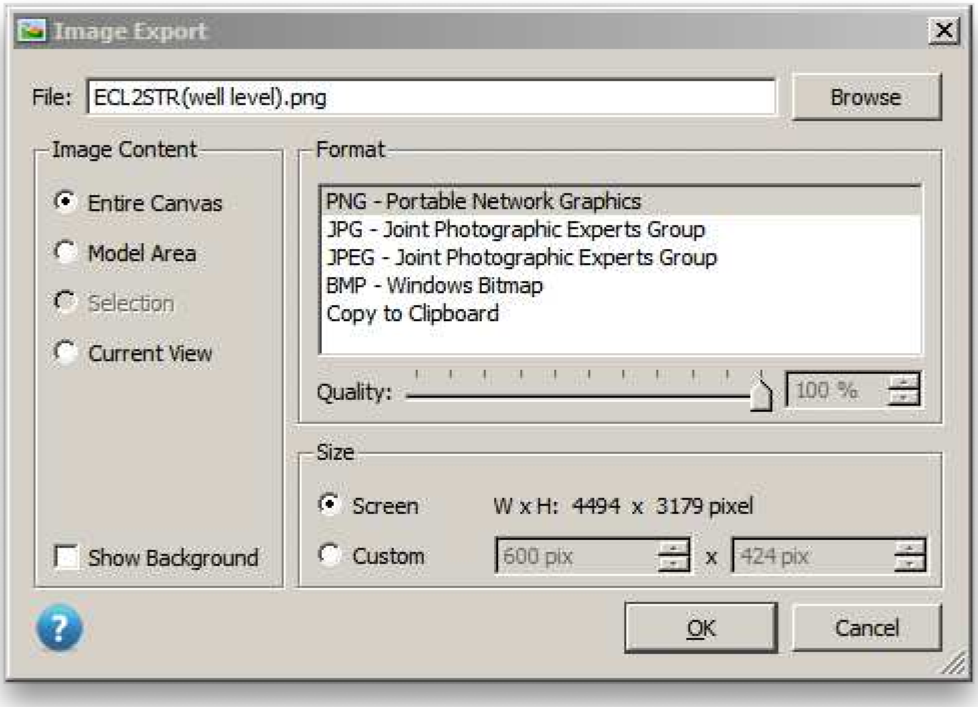


Exporting Images

Menu *File -> Export -> Export Image...* allows to export current canvas area as image. There are possibility to export

- Canvas
- Model Area (same as Zoom-To-Fit)
- Selection
- View

into 3 formats as PNG, JPEG and BMP. Also you can export the image directly into *System Clipboard* to paste into another application that can accept images.

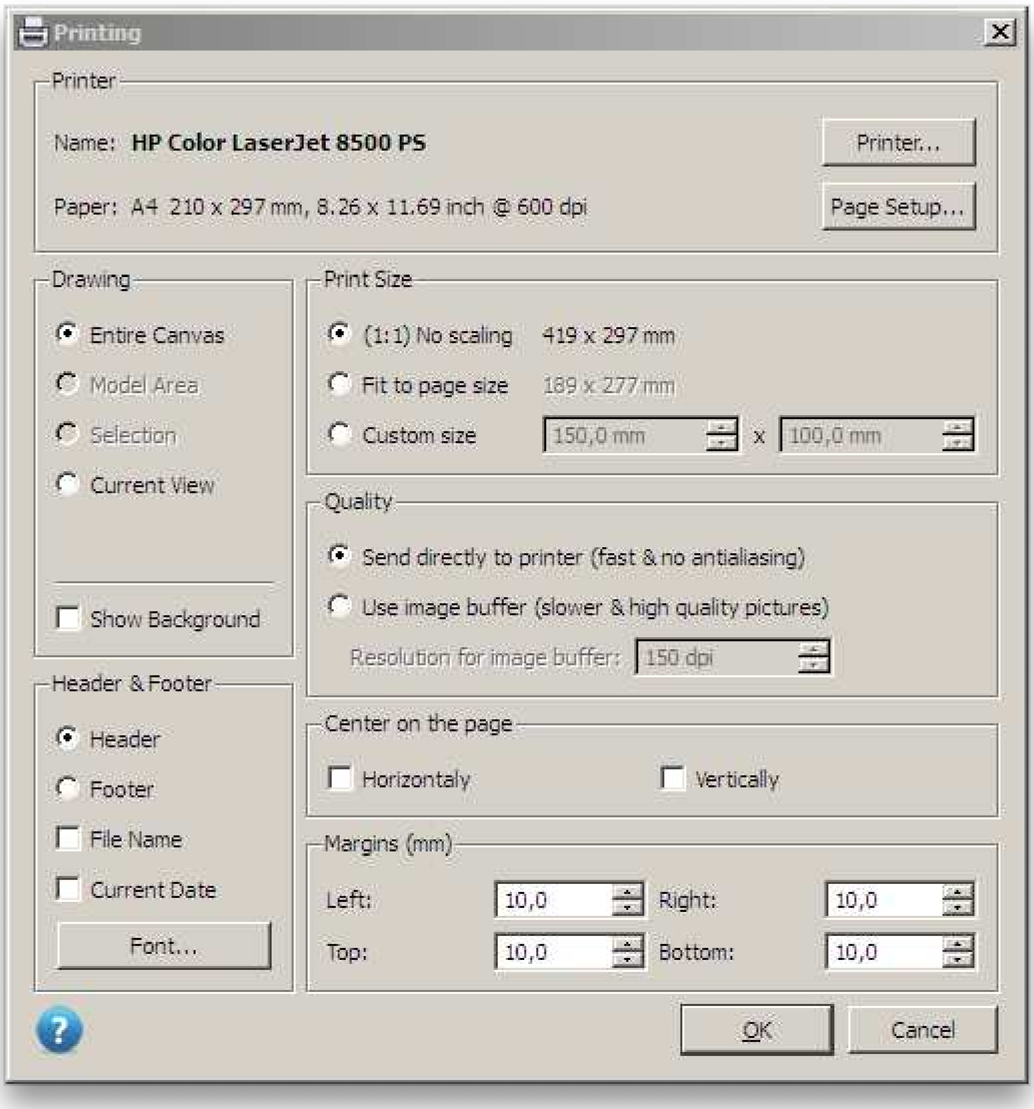


By default the dialog suggest you name for image same as current name of composite. You can easily change it.

Default location of exported image is *Project folder* though you can easily change it by using *Browse* button.

Print Dialog

Choose *File* > *Print*.


Printing

Printer

Name: **HP Color LaserJet 8500 PS**

Paper: A4 210 x 297 mm, 8.26 x 11.69 inch @ 600 dpi

Drawing
- [✓ Entire Canvas]
- Model Area
- Selection
- Current View
- ☐ Show Background

Print Size
- [✓ (1:1) No scaling] 419 x 297 mm
- Fit to page size 189 x 277 mm
- Custom size 150,0 mm x 100,0 mm

Quality
- [✓ Send directly to printer (fast & no antialiasing)]
- Use image buffer (slower & high quality pictures)
- Resolution for image buffer: 150 dpi

Header & Footer
- [✓ Header]
- Footer
- ☐ File Name
- ☐ Current Date

Center on the page
- ☐ Horizontaly
- ☐ Vertically

Margins (mm)

Left:	10,0	Right:	10,0
Top:	10,0	Bottom:	10,0

In the dialog you can specify the page size, orientation, or scaling.

Use *Printer* and *Page Setup* buttons to select your paper size and printer you would like to use.

In *Drawing* group you can specify the area you would like to print

To include the background image, enable *Show Background*

There are two way to print:

- Directly to Printer
- Use image buffer

It is recommended to use image buffer for better quality.

Application Preferences

Petrostreamz Pipe-It, like most of the applications on your computer, has a set of preferences you can customize. You use preferences to specify things such as auto-save settings and a default folder for storing projects.

To view or change preferences:

Win: Choose *Edit > Preferences*

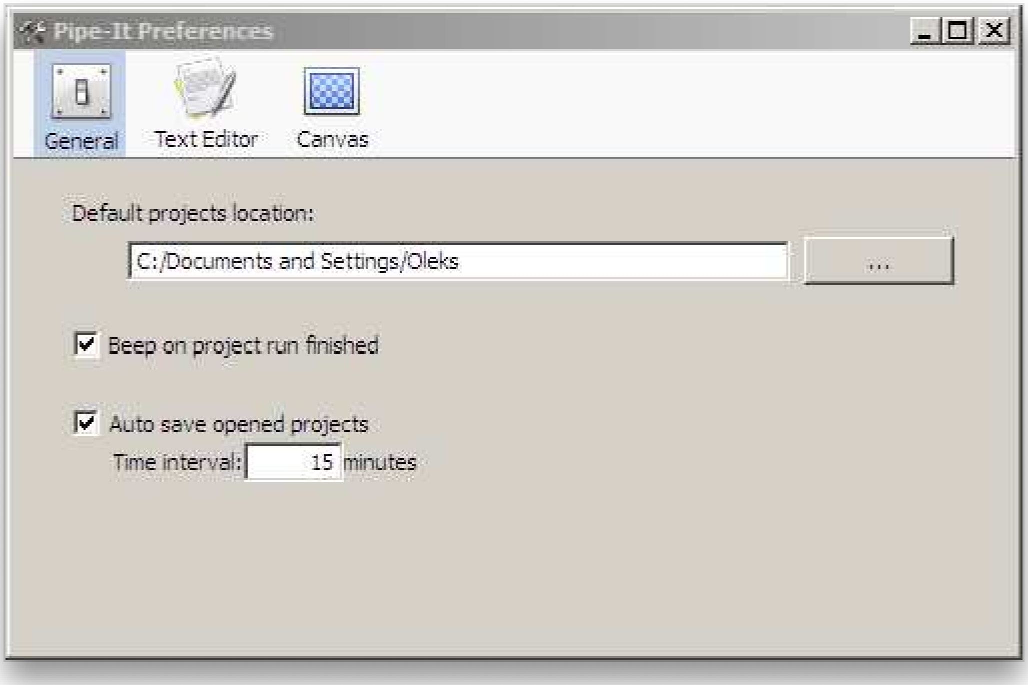
Mac: Choose *Pipe-It > Preferences*

Preferences are tabbed into 3 pages:

- General
- Text Editor
- Canvas

General Preferences

The page of preferences defines settings:



Default projects location:
C:/Documents and Settings/Oleks

☑ Beep on project run finished

☑ Auto save opened projects
Time interval: 15 minutes

Default project location - the location is offered to you when you would like to save new project designed from scratch that has not been saved yet. User setting if you would like to locate all your projects under some folder.

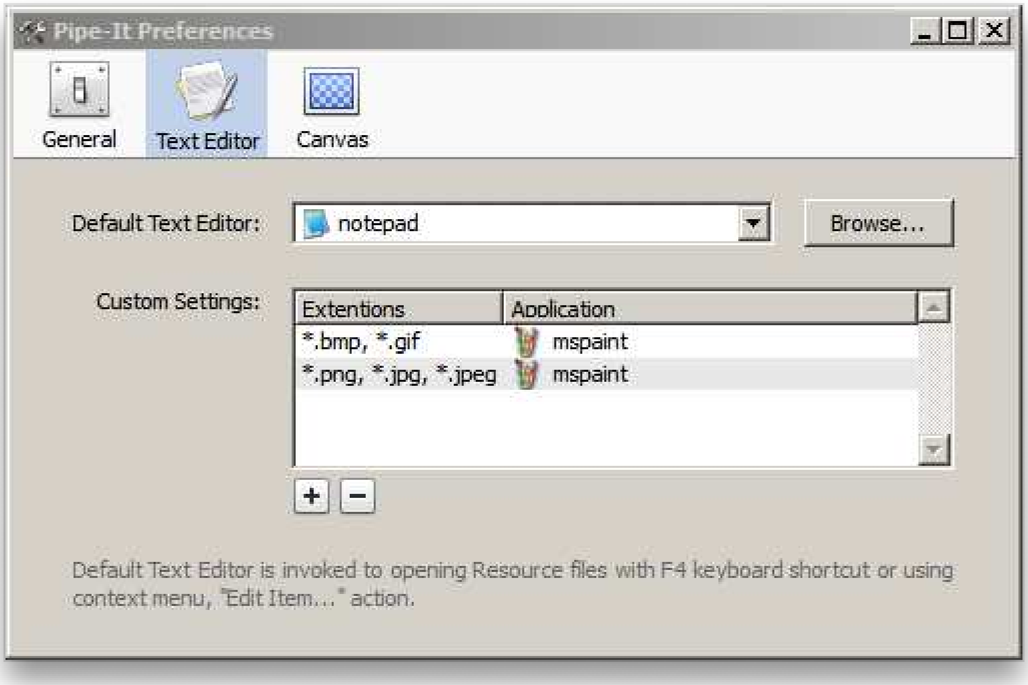
Beep on project run finished - the checkbox makes Pipe-It beeping when execution of your project is finished. By default the setting is enabled, but you can disable it when running Optimizations and it produces several beeps too often. Anyway this is good indication for long term executions to attract attention.

Auto-save opened projects - the option with defined interval enables timer to save your project after defined time from last save operation. It saves project into model+view files, ppm and ppv with same file name as opened project but prepended with "~". In case of power failure you have possibility to extract lost changes to project by getting information from auto-save files.

Text Editor Preferences

Text Editor page allows you to define default text editor for *Edit Item* action for Resources. Note that it is different than double-click on Resource which invokes Application from default system association with this file type. Instead Right click on Resource and choosing *Edit Item*, or just pressing **F4** invokes association defined by this preferences page.

It can be very useful if you would like to have fast way of starting File edition process but without changing the default association in OS to start editor with double-click. Using *F4* invokes Editor specified in Pipe-It and allows to keep double-clicking for processing the file by some utility.



Default Text Editor: notepad

Custom Settings:

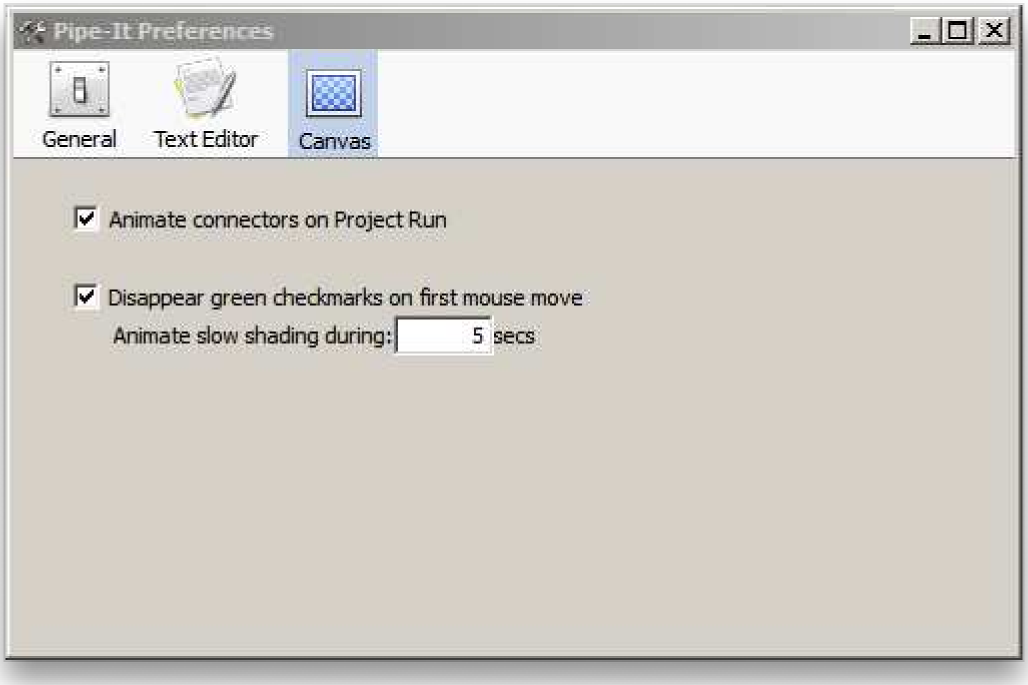
Extentions	Application
*.bmp, *.gif	mspaint
*.png, *.jpg, *.jpeg	mspaint

Default Text Editor is invoked to opening Resource files with F4 keyboard shortcut or using context menu, "Edit Item..." action.

You can see that the page allows you to extend associations for several file types. By default it provides additional association for image files to show you example of defining new bindings to your editing applications.

Canvas Preferences

You can customize next canvas settings:

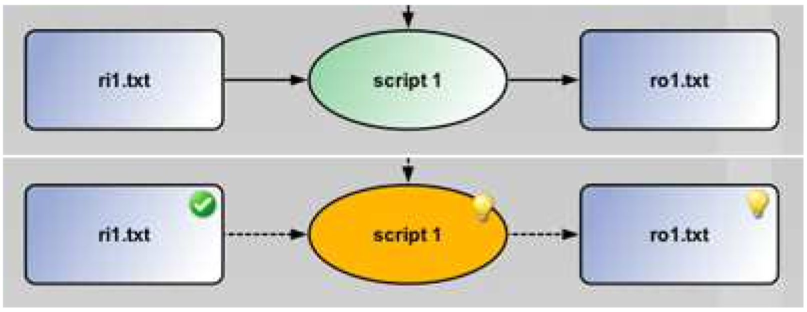


[✓] Animate connectors on Project Run

[✓] Disappear green checkmarks on first mouse move

Animate slow shading during: 5 secs

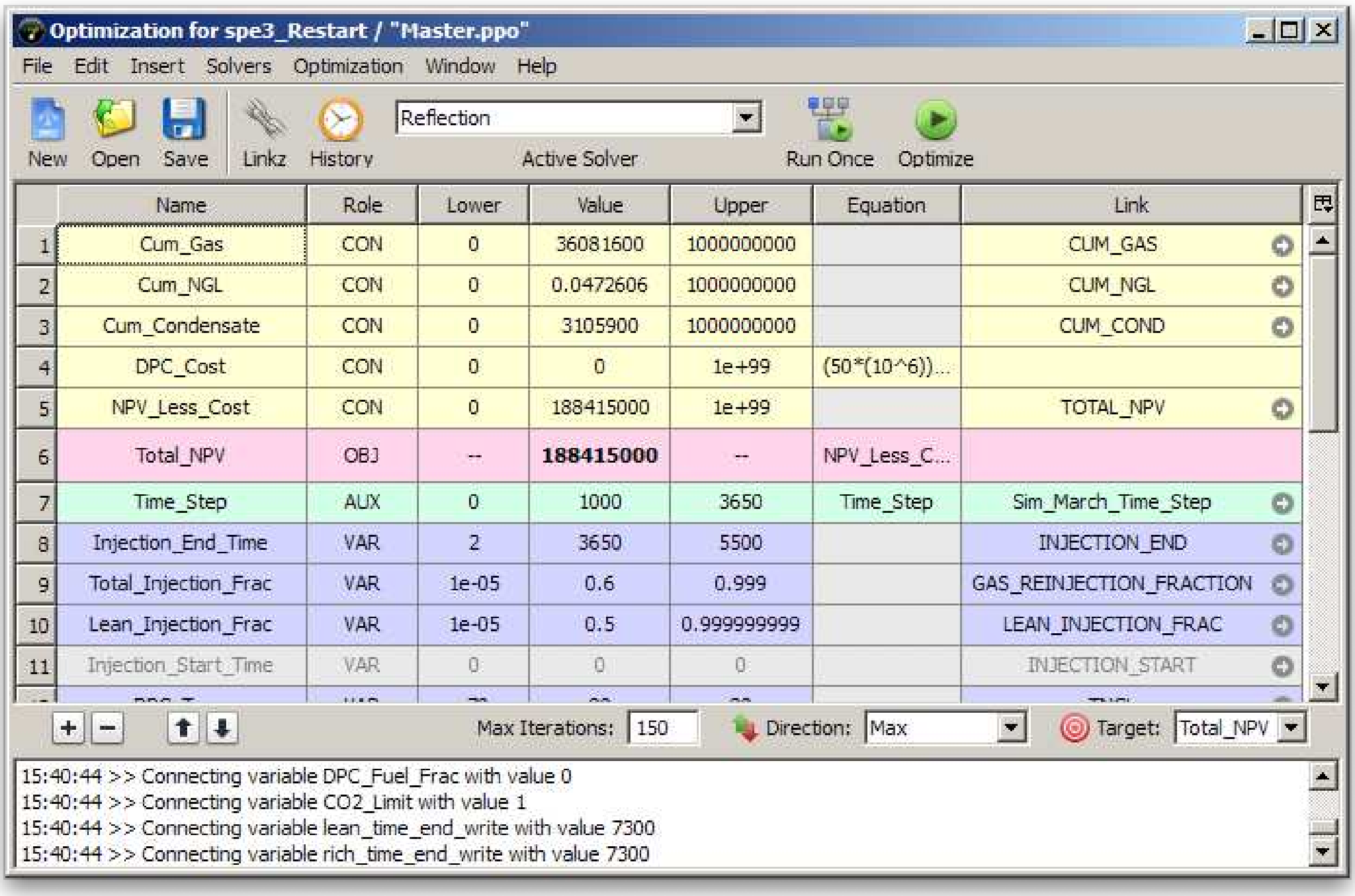
Animate connectors in Project Run - when you run your project, then active connections (which are connected to elements that are being processed this time) are animated by moving dots to show you which Resources are currently used or produced by running Scripters.



Disappear green checkmarks on first mouse move - When execution of project is completed then all processed elements got checkmarks (in right top corner) to indicate completeness of executed elements. Checkmarks that indicates problems attracts attention to resolve situation and start execution again. But in case of successful execution all elements are marked with green checkmarks and not required to be kept on canvas to continue your operations. For such case you can use the option to keep green checkmarks until first mouse move - you are definitely noticed about project state and can continue work.

It is convenient if project run time is huge and you need indication when you return to computer to look at completeness of the execution.

Optimizer Window



Reflection

	Name	Role	Lower	Value	Upper	Equation	Link
1	Cum_Gas	CON	0	36081600	1000000000		CUM_GAS
2	Cum_NGL	CON	0	0.0472606	1000000000		CUM_NGL
3	Cum_Condensate	CON	0	3105900	1000000000		CUM_COND
4	DPC_Cost	CON	0	0	1e+99	(50*(10^6))...	
5	NPV_Less_Cost	CON	0	188415000	1e+99		TOTAL_NPV
6	Total_NPV	OBJ	--	**188415000**	--	NPV_Less_C...	
7	Time_Step	AUX	0	1000	3650	Time_Step	Sim_March_Time_Step
8	Injection_End_Time	VAR	2	3650	5500		INJECTION_END
9	Total_Injection_Frac	VAR	1e-05	0.6	0.999		GAS_REINJECTION_FRACTION
10	Lean_Injection_Frac	VAR	1e-05	0.5	0.999999999		LEAN_INJECTION_FRAC
11	Injection_Start_Time	VAR	0	0	0		INJECTION_START

Max Iterations: 150 Direction: Max Target: Total_NPV

```
15:40:44 >> Connecting variable DPC_Fuel_Frac with value 0
15:40:44 >> Connecting variable CO2_Limit with value 1
15:40:44 >> Connecting variable lean_time_end_write with value 7300
15:40:44 >> Connecting variable rich_time_end_write with value 7300
```

Interface explanations:

1. *Toolbar* with most used actions for file operations, selecting solver and actions to run optimization.

1. *Column Selector* button that allows you to show or hide columns of variable properties that you would like to see. There are next properties are available:

- Name
- Role
- *Active* - you can enable / disable selected variables for your current optimization, in case of *inactive*, the variable looks like frozen
- Type
- *Lower* - minimal allowed value of the variable
- Value
- *Upper* - minimal allowed value of the variable
- Equation
- *File* - filename where the variable is assigned
- Text
- Occurrence
- Position
- *Resource* - element in Pipe-It model assigned with the variable

1. *Variables* list with names, values and properties

1. *Control bar* that allows you to add new variable, remove existing, rearrange variables for your suggested order, direction of optimization process and *Target* variable. There are available next direction of optimization as:

- *Minimum* - Solver will try to find minimal value of *Target* variable
- *Maximum* - Solver will try to find maximum value of *Target* variable
- *Find Feasible*

See Also: Optimization Variables

Optimization Variables

Summary:

VAR: User- or optimizer-specified. May be written to file. Updated before any other variable and before model execution.

AUX: Either set by equation, read from file, or user-specified, in that order of priority. If set by equation, may also be written to file. Updated after VARs but before model execution.

CON: Unless a user-specified constant, either read from file or set by equation, after model execution. (Note: Constraints are recognized for all variables, not just the results labeled CON).

OBJ: Same as CON, except the optimizer will try to minimize or maximize it.

Detailed explanations:

1. Before each model run (which could be an optimization iteration), all VARs are updated first. They will have either been manually specified by the user, or else set to new values by the optimizer engine. VARs are never set by equation. If a VAR is associated with a file, it will be written to its file during this update phase.

2. Then, still before the model run, each AUX variable will be updated in turn. An AUX may be associated with an equation, a file, both, or neither. If it has an equation, it will always be set by that equation (using the current values of any referenced variables). If it also has a file, the updated value will then be written to the file. If it has a file but no equation, then it will be read from (instead of written to) that file. If it has neither equation nor file, it will just be used as a user-specified constant.

3. After the VARs and AUXs have been updated and (if necessary) written to their files, the model will then be executed.

4. After the model execution, all other variables (CON and OBJ) will be updated in the order specified, regardless of type. Each of these variables can have either an equation or a file, but not both (it can also have neither, which would just make it a user-specified constant). If it's not just a constant, then it will either be set from its equation (using the current values of any referenced variables) or read from its file.

5. Constraints will then be tested for all variable types (not just CON, which is now really a misnomer) and fed back, along with the OBJ, to the optimizer (if necessary).

Pipe-It Optimizer is the main interface for multiple launches of the entire (or subset of) project in an optimization context. Variables (with roles VAR, AUX, OBJ, CON & DER) defined in the Optimizer window are used by the chosen Solver to optimizer the OBJ.

Some Solvers are provided with the default installation of Pipe-It. They are described below. Additionally, complete documentation is provided to plugin any other Solver to the Optimizer. These will then be available as choices in the Solver drop-down list if the Plugin DLLs are stored in the Solvers folder under the Pipe-It installation folder.

Random Sampler Solver

This solver is intended to provide a succession random values (within the defined Upper and Lower bounds) of the VAR variables defined in the Optimzer and launch the project with those values. The values of OBJ & CON are read after the completion of run and stored as part of the optimization History. The number of iterations will depend on the Max Iterations value. At the end of all the iterations, the best solution will be restored.

This solver is ideal for getting a feel of the solution space and is often the first step in an optimization problem.

Case Matrix Solver

This solver provides a spreadsheet to the user to enter the values (sets) with which to run the project. In older versions of Pipe-It this is referred to as Experimental Design. No facility for populating the values in the Case Matrix is currently available and the user is responsible for filling them up with values needed. The project can then be executed for any row (set of values) or for all the cases. The values of OBJ & CONS are similarly read after teh completion of the run(s) and stored as part of History.

Easy cut-n-paste facility allows setup of the case matrix in Excel and paste the cases into the Case Matrix spreadsheet. It is possible to save the case matrix as tab-delimited files.

This solver is ideal for performing scenario studies and evaluating the performance of the project for a suite of input variables.

Nelder-Mead Simplex-Reflection Solver

The Nelder and Mead (1965) reflection simplex algorithm is the default "real" solver provided with Pipe-It. This algorithm does not require derivatives and is very robust. A direct search method of optimization that works moderately well for stochastic problems. It is based on evaluating a function at the vertices of a simplex, then iteratively shrinking the simplex as better points are found until some desired bound is obtained. Lagarias et al. (1998) studied the convergience properties of the Nelder-Mead reflection simplex, and concludes that they are at best linear.

A plugin variation of the built-in Reflection solver allows specification of two parameters that affect the convergence of this solver.

Fractional Convergence Tolerance (ftol): the solver converges if the absolute value of the difference between the best and worst objective values within the current simplex (including penalties for constraint violations) is less than or equal to ftol times the absolute value of the best objective value. This tolerance determines the accuracy to which the optimum objective can be found, but it should be kept larger than the precision (i.e., noise or round-off level) of the model. Otherwise, before converging, the algorithm could encounter conflicting directional trends, which could lead to expansions of the simplex before the second convergence criterion (below) could be met. The current default (5e-7) is intended to give 6 digits of accuracy in the optimum objective, but it probably requires at least 7 digits of precision in the model.

Movelength Convergence Tolerance (ltol): the solver converges if the simplex contracts to where the best vertex is less than or equal to the square root of ltol in distance from any other vertex (with the distance between the upper and lower bound for any independent variable defined as 1). This tolerance determines the accuracy to which the independent variables at the optimum solution can be found, but it should be kept larger than the minimum relative change in the independent variables that would produce a measurable change (greater than the noise level) in the objective. Unfortunately, this is a little difficult to estimate without prior knowledge of the second derivatives of the objective at the optimum solution. The current default (1.e-12) is intended to determine the independent variables to within 1e-6 times the differences between their bounds, which might be a higher accuracy than many problems warrant.

The solver converges if **either** criterion is met.

TIP: If the defaults don't seem to work well for a particular problem, and the precision of the problem is estimated to be epsilon, say, then one might try setting ftol to somewhere between 10 and 100 times epsilon and ltol to roughly ftol squared.

IPOPT Solver

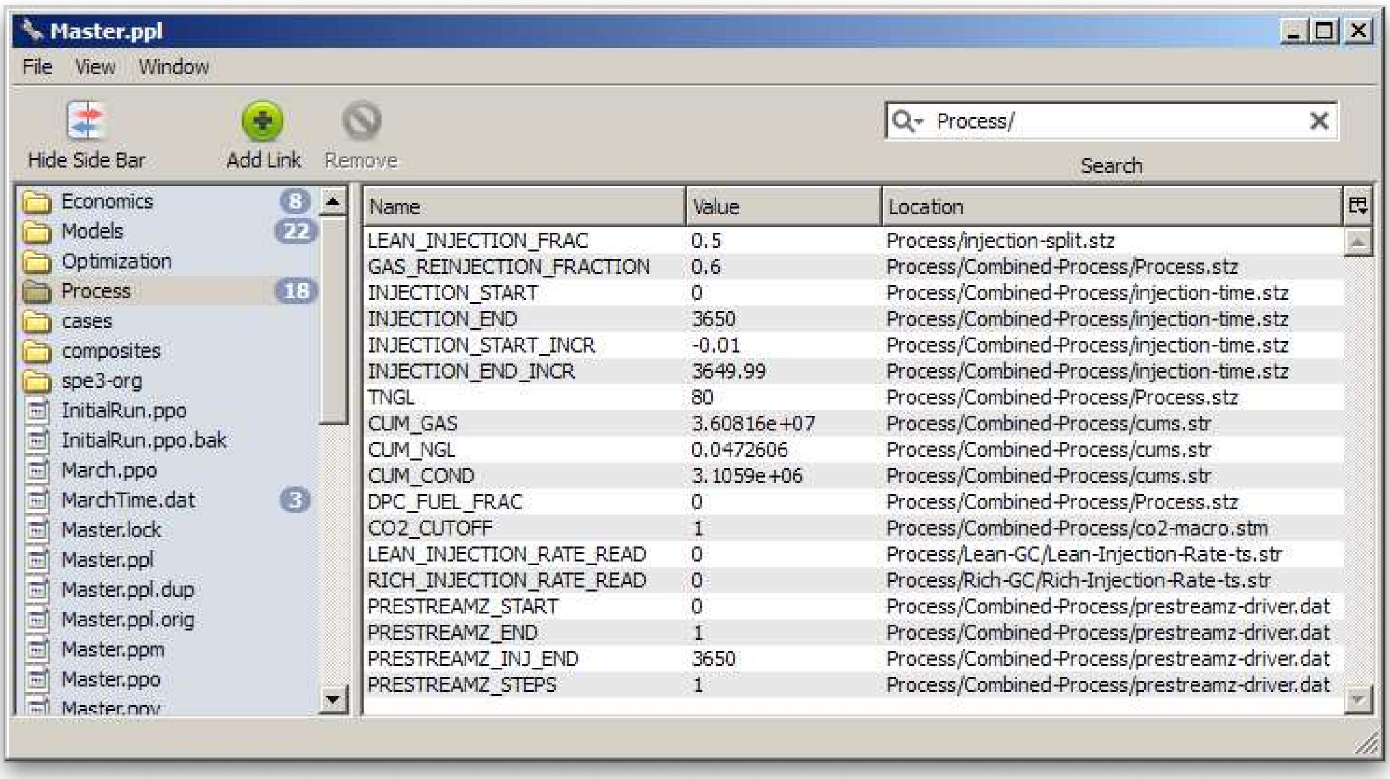
IPOPT, short for "Interior Point OPTimizer, pronounced I-P-Opt", is a software library for large scale nonlinear optimization of continuous systems. Pipe-It includes an experimental version of this solver implemented as a Plugin. This will be the default derivatives-based solver provided with Pipe-It. More documentation will be available at this site when the complete implementation is included.

Create Own Solver

C code

Linkz

Linkz Window



Name	Value	Location
LEAN_INJECTION_FRAC	0.5	Process/injection-split.stz
GAS_REINJECTION_FRACTION	0.6	Process/Combined-Process/Process.stz
INJECTION_START	0	Process/Combined-Process/injection-time.stz
INJECTION_END	3650	Process/Combined-Process/injection-time.stz
INJECTION_START_INCR	-0.01	Process/Combined-Process/injection-time.stz
INJECTION_END_INCR	3649.99	Process/Combined-Process/injection-time.stz
TNGL	80	Process/Combined-Process/Process.stz
CUM_GAS	3.60816e+07	Process/Combined-Process/cums.str
CUM_NGL	0.0472606	Process/Combined-Process/cums.str
CUM_COND	3.1059e+06	Process/Combined-Process/cums.str
DPC_FUEL_FRAC	0	Process/Combined-Process/Process.stz
CO2_CUTOFF	1	Process/Combined-Process/co2-macro.stm
LEAN_INJECTION_RATE_READ	0	Process/Lean-GC/Lean-Injection-Rate-ts.str
RICH_INJECTION_RATE_READ	0	Process/Rich-GC/Rich-Injection-Rate-ts.str
PRESTREAMZ_START	0	Process/Combined-Process/prestreamz-driver.dat
PRESTREAMZ_END	1	Process/Combined-Process/prestreamz-driver.dat
PRESTREAMZ_INJ_END	3650	Process/Combined-Process/prestreamz-driver.dat
PRESTREAMZ_STEPS	1	Process/Combined-Process/prestreamz-driver.dat

Locating Values

Using Delimiters and Regular Expressions

Example 1, Output of PhazeComp

Output of PhazeComp application looks in such way:

```
Final Variable Definitions
==========================

 Var Num    Reg Var                  Variable                      Current       Initial
----------  ---------  -----------------------------------------  -------------  -----------
     1                 SG-Factor                                  3.08200e-01
     2                 Global-Shape                               7.00000e-01
     3                 Global-Average                             2.60000e+02
     4                 Global-Bound                               9.50000e+01
     5                 C1-CPLUS-FAC                               1.80000e-01
     6                 CoreLab-Average                            2.51750e+02
     7                 19930806-Hycal-12HZ-18-6-13-W2M-Average    2.41877e+02
     8                 20080518-Hycal-Average                     2.57000e+02
     9                 19810108-CL-14-26-5-13-W2M-Average         2.73000e+02
    10                 19930325-Hycal-Average                     2.65652e+02
    11                 19930228-DBR-Average                       2.33192e+02
    12                 CoreLab-fg                                 6.59010e-02
    13                 19930806-Hycal-12HZ-18-6-13-W2M-fg         1.05121e-01
    14                 20080518-Hycal-fg                          6.79000e-02
    15                 19810108-CL-14-26-5-13-W2M-fg              6.10758e-02
    16                 19930325-Hycal-fg                          3.52918e-01
    17                 19930228-DBR-fg                            3.12911e-01
    18          1      MC4-fg-crit                                8.60178e-01    8.68650e-01
    19                    TB-C30P                               1.47000e+03
    20                    TC-C30P                               1.78300e+03
    21                    PC-C30P                               1.60000e+02
    22                    CO2-CPLUS-CHUEU-EXP                   1.00000e+00
    23          2         CO2-CPLUS-FAC                         0.00000e+00    1.00000e+00
    24          3         CO2-CPLUS-INC                         7.50503e-02    1.00000e-01
    25          4         BIP-CO2-C30P-INC                      3.24955e-02    0.00000e+00
```

Problem of locating values of variables which are based in fourth column "Current" is related to possibility of presence of numbers in "Reg Var" column. Then with appearing a number in the column, the position of token can shifted and you get wrong value of your variable.

Solution

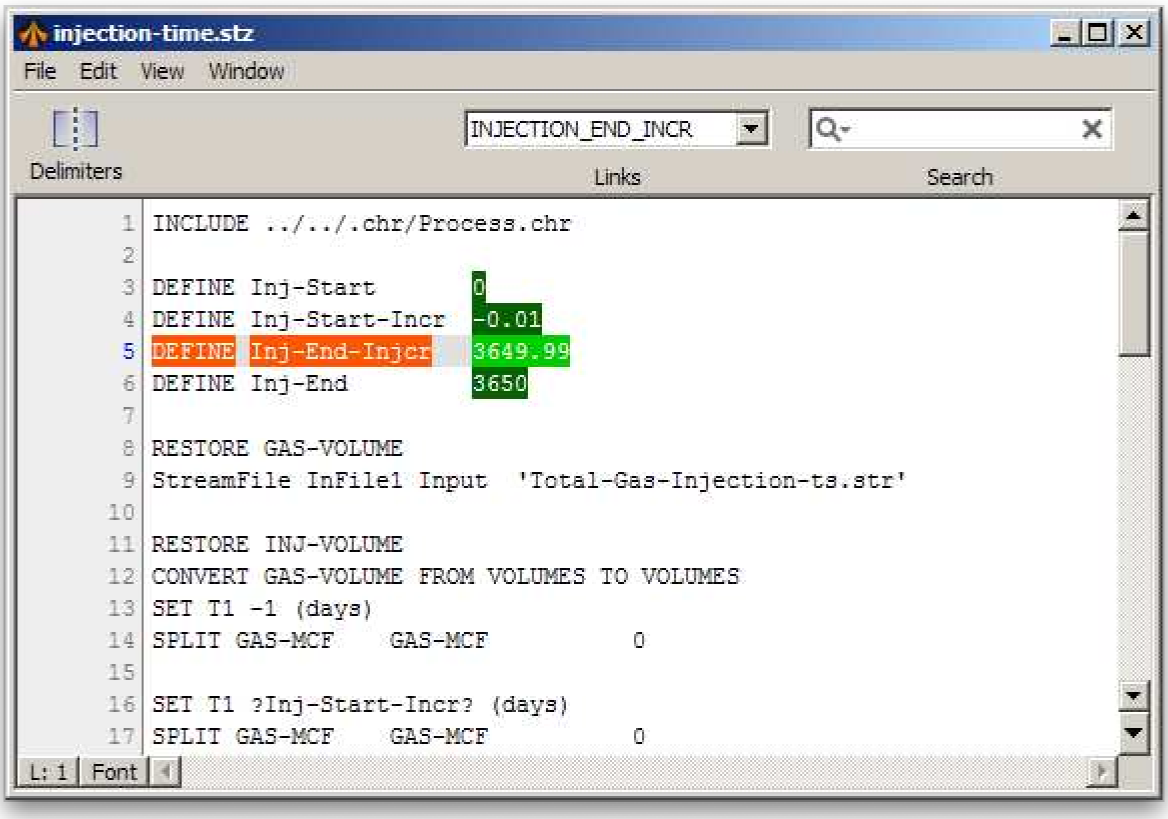
We need to avoid of considering numbers in "Reg Var" as tokens. For this you need to change delimiters regular expression. Because value of variable is definitely larger than 2 digits, we can just ignore all tokens with length less than 3 symbols. For this, in delimiters dialog, change TOKEN REGEXP

`[^$delimiters]+` to `[^$delimiters]{3,}`

It means that instead "+" (which means at least one symbol and more) linkz will look for "{3,}" (which means at least three symbols and more)

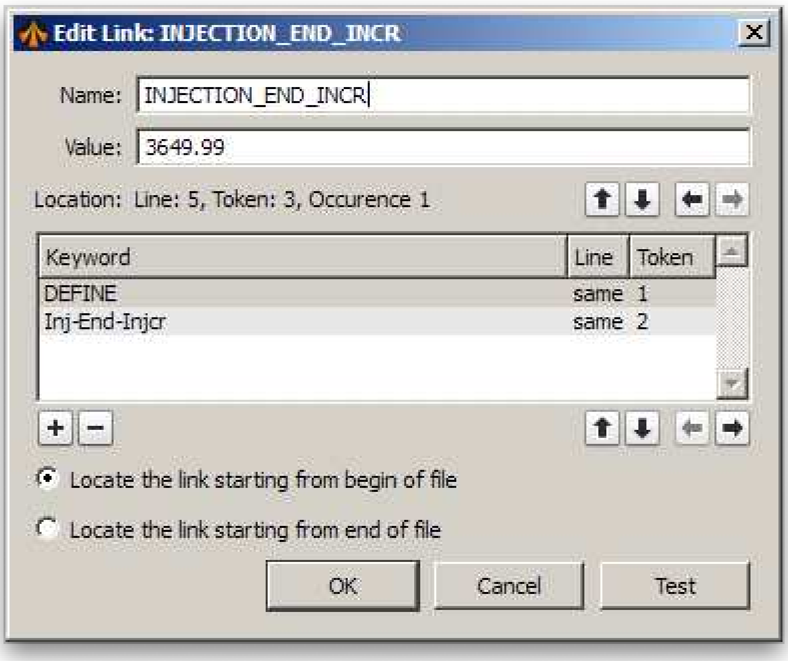
Then first two columns will be ignored and tokens are counted from "Variable" column.

Linkz File Viewer



```
INCLUDE ../../.chr/Process.chr

DEFINE Inj-Start        0
DEFINE Inj-Start-Incr   -0.01
DEFINE Inj-End-Injcr    3649.99
DEFINE Inj-End          3650

RESTORE GAS-VOLUME
StreamFile InFile1 Input  'Total-Gas-Injection-ts.str'

RESTORE INJ-VOLUME
CONVERT GAS-VOLUME FROM VOLUMES TO VOLUMES
SET T1 -1 (days)
SPLIT GAS-MCF    GAS-MCF        0

SET T1 ?Inj-Start-Incr? (days)
SPLIT GAS-MCF    GAS-MCF        0
```

Defining new Link and edit existing one



Name: INJECTION_END_INCR

Value: 3649.99

Location: Line: 5, Token: 3, Occurence 1

Keyword	Line	Token
DEFINE	same	1
Inj-End-Injcr	same	2

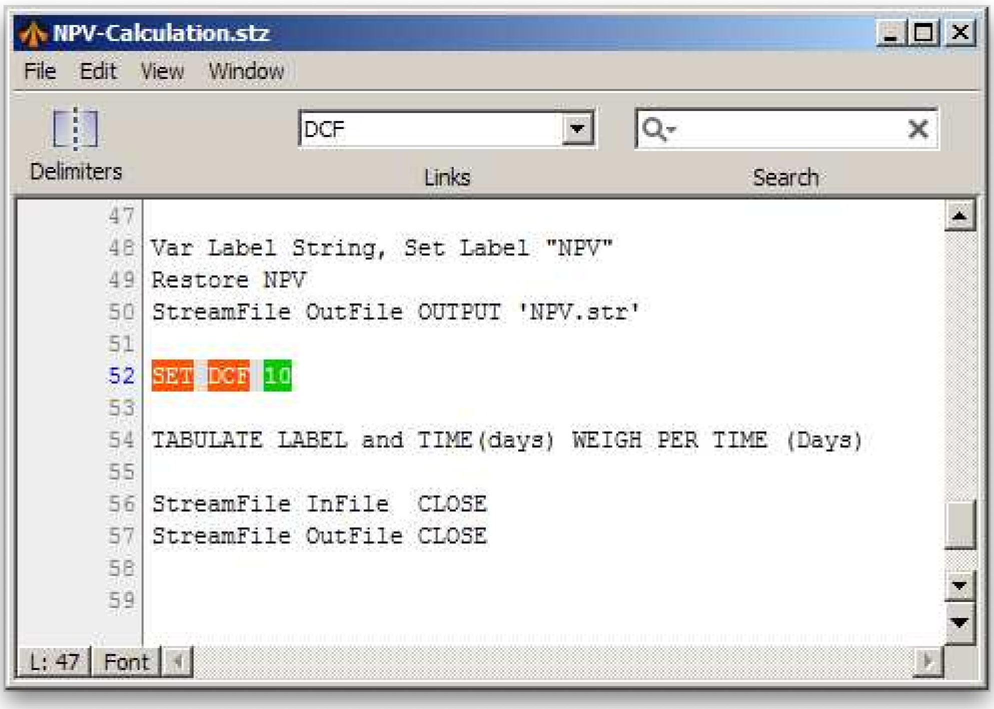
(●) Locate the link starting from begin of file

( ) Locate the link starting from end of file

Locating Links in files

This page explains how Linkz can locate and watch for values in files

Linkz File Viewer



```
Var Label String, Set Label "NPV"
Restore NPV
StreamFile OutFile OUTPUT 'NPV.str'

SET DCF 10

TABULATE LABEL and TIME(days) WEIGH PER TIME (Days)

StreamFile InFile  CLOSE
StreamFile OutFile CLOSE
```

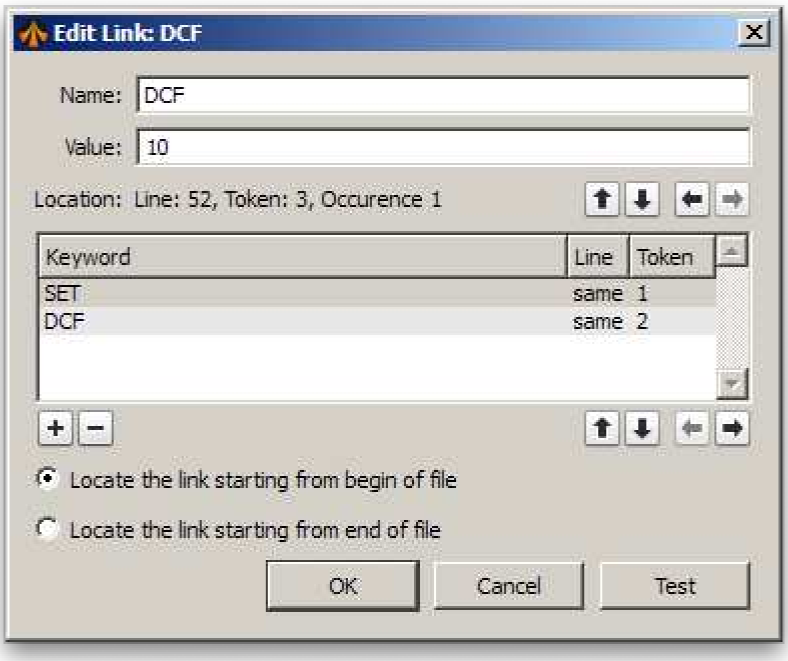
The file viewer window provides good visualization of links locating. All link values are highlighted with green color to indicate places where Linkz is watching for values. If you click mouse on the green area of link to select it, the file viewer will highlight keywords used for locating the value. These keywords can be located around the value on above or below lines.

When Linkz searches for the value of link it looks for such "fingerprints" - value with surrounding keywords. Of course it is not necessary to have these keywords unique, and often it is not possible to find such unique definition in case of repeating some blocks of similar text in file. In such situation Linkz counts occurrences of matches and choices defined occurrence.

Also there is a way to start the value lookup from end of file. Then Linkz counts occurrences of matches from end. It can be extremely useful for files with iterations - file is compound of blocks of almost same content and only last block contains values that should be linked. Because the amount of these blocks is not same on each execution it makes sense to start the value search from end of file.

Let's have a closer look at the link definition

Edit Link Dialog



Name: DCF

Value: 10

Location: Line: 52, Token: 3, Occurence 1

Keyword	Line	Token
SET	same	1
DCF	same	2

[✓ Locate the link starting from begin of file]

Locate the link starting from end of file

First you can see the link name and value. Link name is editable - so you can rename your link. Below there is an indication of location - line number, index of token in the line, occurrence and buttons with arrows to adjust the position of value (they can be useful if after file modification linkz found wrong token and you need to adjust line or token index) The list of keyword has also similar buttons for adjusting positions of keywords. With plus and minus buttons you can add additional keywords or remove existing. By default just added keyword is the first token in line. So after adding you move it with arrow buttons to position where your keyword is located. Finally you have option enable the link lookup from the end of file. With *Test* button you can test your link - the you can see updated occurrence.

We recommend to use at least 2-3 keywords for better definition of location and to have occurrence equal to 1 - which mean that it is first found location from begin or end of file.

If you work with huge file and your value is located at the end of file, then it is recommended to use *Locate the link starting from end of file*. Otherwise it has to parse whole file from the begin to find the location. In case of 100MB it means minutes and if you are using the link in some operations in your project it can slow down the execution.

Interaction with 3rd party software

In the same way as executing a Microsoft Excel from within Pipe-It, 3rd party applications like Eclipse7, Hysys8, VIP9, OLGA10 and others may be executed via the scripter command and a .bat file.

1. Eclipse is trademark of Schlumberger Ltd

2. Hysys is treademark of AspenTech Inc

3. VIP is trade mark Landmark Graphics Corp.

4. OLGA is the trademark of Scandpower Technology AS

See Also:

- Interacting Excel
- Plots on Canvas in Pipe-It projects and using R language

Interacting with Microsoft Excel

The Pipe-It created resource files (*.str files) are tab de-limitated and can readily be opened in a spreadsheet program like Microsoft Excel. The Pipe-It scripter command can be set to execute a MS-DOS .bat file with the following command:

```
start /w Excel.exe ts.str
```

This DOS command will open the ts.str file located under the current folder file in Microsoft Excel. The /w will make the process stop (pause Pipe-It) until Excel is closed. This causes Excel to behave as it is part of Pipe-It since the execution continues as soon as the instance of Excel is closed.

Any other program may be executed the same way as this. It is possible to edit a driver file from the software by starting the project with a process executing a text editor. The text editor could open the driver file and one could edit it and save, then close the editor. After this is done the project will continue, using the newly edited driver file.

Plots on Canvas in Pipe-It projects and using "R" language

Using Item properties dialog or with drag-and-drop you can easily attach images to your canvas items. Pipe-It uses OS file watchers, so if any of your image is changed, it is reloaded and appropriate items, which have attached the image, are updated.

It opens very powerful way to generate graphics and diagrams from your project scripts and show result images directly on canvas. You can use different utils or languages to generate them - they should be just part of your project (just scripters) to run with project and update appropriate graphics files.

One of advantages is possibility to use "R" language. This is very powerful language that used for for statistical computing and graphics. For example few screenshots that illustrates possibilities at their webpage:

- http://www.r-project.org/screenshots/screenshots.html
- http://addictedtor.free.fr/graphiques/thumbs.php?sort=votes

As example we can study how to make plots for some *STR* file. Imagine that you have some Resource with attached *STR* file:

This file: *TUNE.str* contains:

```
Streamz        1
Note        "Streamfile manufactured from user-specified data in file:RNB-Profiles-P10-P50.x
Note        "Case:P50"
Char        "BO+W"
```

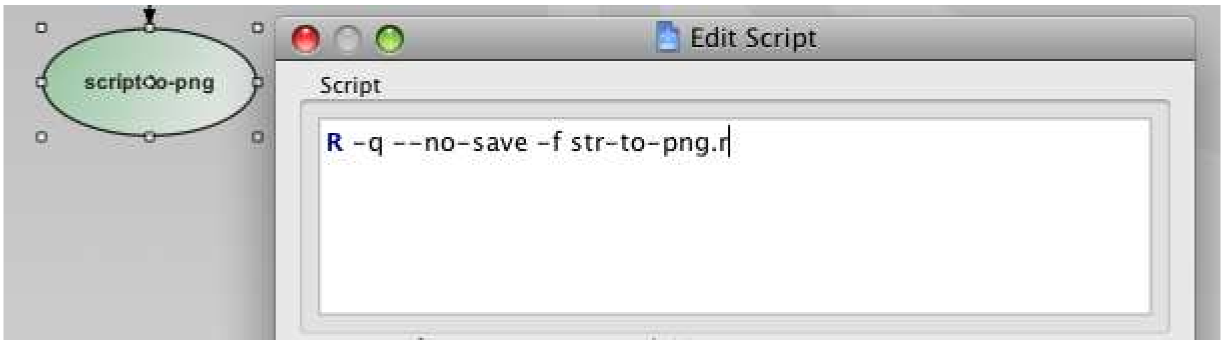
```
Variable        Year        Integer
Variable        Group        String

Data

Set        Group        TUNE

Year        Amounts  SO        SG        SW
2006        271552        555555550        13781
2007        150029        1736142619        6788
2008        26812        4347876676        1239
2009        178500        5451284722        0
2010        109233        6447013889        0
2011        40767        6385229167        0
2012        151500        1478472222        6122
2013        135000        1305000000        5510
2014        15000        145000000        612
```

There is a table that can be easily loaded with "R" language. For generating graphic "Year" --> "Amounts", only 4 lines of "R" code is required:

The script file: *str-to-png.r* contains:

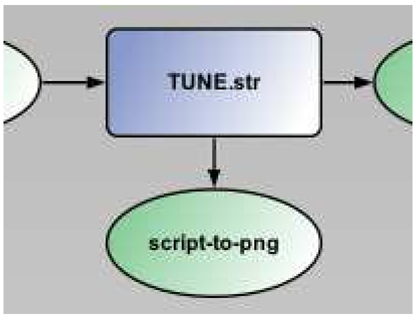
```
png("TUNE.str.png")
inp <- read.table("TUNE.str", sep="\t", skip=11, h=T)
plot(inp[[1]], inp[[2]], type="b")
dev.off()
```

Now you need to create simple script that call this "R" script file and generates png

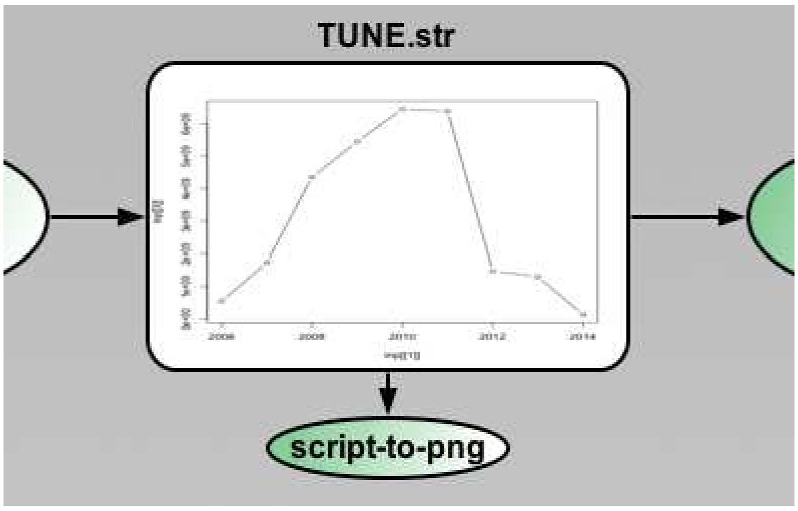


```
R -q --no-save -f str-to-png.r
```

(You need to have "R" in your *PATH* environment variable)



When you run the project you get *TUNE.str.png*. Which you can attach as background directly to your Resource file.



Then the plot will be generated each time your resource got new data and is completed during project run and background of the Resource will be updated automatically.

Actually you can attach the image to any other item - "script-to-png" or canvas background or background of composite or annotation - it does not matter, Pipe-It watch if the image is changed and updates appropriate items on canvas. You can create several "monitors" to watch how some diagrams are changed during run etc.

See also:

- http://www.r-project.org/
- http://www.r-project.org/screenshots/screenshots.html
- http://addictedtor.free.fr/graphiques/thumbs.php?sort=votes

Graphics and Logos related to Petrostreamz and Pipe-It

Petrostreamz Logo:

PETROSTREAMZ

Streamz Documentation

Streamz

Streamz is a generic program to convert fluid streams from one characterization to another. A characterization is a description of the number and names of components making up the stream, and optionally their molecular weights and other properties. Streamz is controlled by an input file which names the characterizations, their properties, and defines conversions among them. This file allows the user to specify opening of stream files, containing the streams in particular characterizations, and also commands for filtering, combining and copying streams from one file to another. The program automatically invokes conversions among the differing characterizations as required. Typically the program commands are generated by the pre-processors or modules, ensuring the syntax to be correct.

1. File Format for Extended Black-Oil PVT Data
2. Getting Started With Streamz
3. Streamz Reference Manual

4. Streamz Tabulation Reference

File format for extended black-oil PVT data

This document describes the file format for exchange of pvt data between a PVT program and the Streamz engine of Pipe-It.

Purpose:

File format for data needed by Pipe-It software to create tables for conversion of black-oil (BO) to compositional streams. Recommended extension of the file is “`boz`”.

Source:

An EOS based PVT model or software that generates black-oil PVT tables.

Description:

The .boz file created using the format described in this note is generated at the same time that an EOS-based PVT program generates a black-oil PVT table.

When the black-oil PVT table is generated a “depletion-type” experiment is simulated – e.g. constant composition expansion, differential liberation test, or constant volume depletion test. The black-oil PVT table contains a number of saturated PVT data and, usually, some undersaturated data.

The “extended” black-oil information written to the .boz file uses only information from the saturated conditions during the depletion experiment. In particular, the data needed for each “saturated” pressure are:

1. solution oil-gas ratio, r_s or R_v, for the reservoir gas phase;
2. molar density factor, $C_{\blacksquare g}$, of the reservoir gas phase, for converting surface oil volume to an “equivalent” surface gas volume;
3. the equilibrium gas composition of the reservoir gas phase;
4. inverse solution gas-oil ratio, $1/R_s$, for the reservoir oil phase;
5. molar density factor, $C_{\blacksquare o}$, of the reservoir oil phase, for converting surface oil volume to an “equivalent” surface gas volume;
6. the equilibrium oil composition of the reservoir oil phase.

Content:

A. Header Section

Occurs fully before the “Extended black-oil PVT data” section (see the example), and consists of the reserved keywords:

1. `TITLE` (optional), followed by a string within single quotes, typically identifying the source of the file.
2. `CHAR` (required), followed by a characterization name. If it contains embedded spaces, the name should be within single quotes.

3. `EOS` (required), followed by one of the recognized names identifying the equation of state. Recognized EOS names are:

- `RK` Original Redlich-Kwong EOS
- `SRK` Soave-Redlich-Kwong EOS
- `PR77` Original Peng-Robinson EOS
- `PR` Modified Peng-Robinson EOS (default)

1. The full property table (required) for the EOS characterization, consisting of:
    1. A row of headings of the following properties:
        - COMP (component name; max. 8 alphanumeric characters)
        - MW (molecular weight)
        - TC (critical temperature)
        - PC (critical pressure)
        - AF (acentric factor)
        - VS (volume shift or translation factor)
        - AMOD (A-parameter multiplier)
        - BMOD (B-parameter multiplier)
    2. A row of units for PC and TC. The units allowed for TC are R, K, F, C, and for PC are ATM, PSI, BAR, KPA, MPA, TORR (absolute pressures are indicated by these keywords as shown, or with an added A - e.g. PSIA; gauge pressures are indicated by an added G - e.g. PSIG). None of the other properties have units associated with them. The units should line up with their respective headings. The meaning of line up is described at the end of this document.
    3. Rows of actual property data. These should line up with their respective headings. The meaning of line up is described at the end of this document. A blank line or the keywords BIPS or END completes a property table.
    4. Binary interaction parameter tables (required), initiated using the BIPS keyword followed by a similarly formatted table. The headings following the BIPS keyword are the names of the components specified in the property table. The first column of each subsequent row also contains the names of components. Corresponding BIP data are entered at the intersection of the respective column and row headings. These data should also line up. A blank line or the word END completes a binary interaction parameter table.
    5. Optional "comments" using the semi-colon (;). Any characters to the right of a semi-colon will be ignored.

B. Extended black-oil PVT data Section

Starts after the "Header" section ends, and consists of:

1. The keyword UNIT, followed by either FIELD or METRIC, specifying the units in which the extended black-oil PVT data has been printed.

2. The keyword ID, followed by a contiguous (no blanks) string or "token" uniquely identifying the table that follows this identifier. This “token” should be not more than 20 characters long; the first table in the example uses CVD1_200_PVTG. The token may be as simple as a unique integer associated with the table. This may be followed on the same line by an optional string, within single quotes, typically giving a description of the source and contents of the identified table. The length can be 132 characters. The first table in the example uses: `'CVD Experiment 1, Feed: 200, Gas Phase'` as a descriptor.

3. Optional lines of comments using semi-colons, to provide a table heading for the identified table. This is ignored completely by the program but may provide important information to the user (e.g. units used).

4. A table of extended PVT data containing, in order, the depletion pressure, solution oil-gas ratio, the molar density, and the phase molar compositions (all defined later). One

table associated with a unique ID should be generated by the PVT program for each phase in each experiment and consist of extended PVT data at all the saturated pressure nodes specified in the experiment. All data for a particular pressure should be on a single line. One table consists of multiple lines of such data for all the pressures associated with this ID.

Definitions:

The following terms are used in the description above:

1. **Pressure** for a given stage of depletion in the experiment (only saturated values should be used).

2. **Oil-gas ratio** is the volumetric ratio of surface oil to surface gas, produced from a reservoir gas phase (r_s or R_v) or a reservoir oil phase ($1/R_s$) , at the specified pressure. If the phase is an oil, the inverse of the traditional solution gas-oil ratio should be printed.

3. **Molar density factor** is a conversion from a surface oil volume to an "equivalent" surface gas volume, defined as:

$$C_{\bar{o}x} = k \cdot \frac{\gamma_{\bar{o}x}}{M_{\bar{o}x}}$$

where:

- ■ : Subscript denoting surface oil
- x : Subscript denoting either o (for reservoir oil) or g (for reservoir gas).
- γ : Specific gravity of surface oil from the relevant phase
- M : Molecular weight of surface oil from the relevant phase.
- k : 132883.29907 in FIELD units (psia, STB/Mscf, and scf/STB). 23667.52637 in METRIC units (bara, Sm3/Sm3, and Sm3/Sm3)[1]

1. **Phase compositions** are mole fractions of the components of the phase at the depletion pressure.

Data format:

The numeric data on the file can be separated with spaces or commas, and supports normal Fortran input format. All input shown in examples on one line, should not extend to multiple lines. The lines are read-in as strings, so there are no field width limitations. Any numeric data, separated by delimiters listed above, can use the width required by the precision. Internally these numeric data are converted to double precision. We suggest using E format with six significant digits.

Example:

Example of the file containing the extended black-oil table, adapted from an output from the in-house Pera-PVTx program.

Getting Started with Streamz

Introduction

Pipe-It provides the user with a set of tools to manage the conversion of information among a multitude of models handling fluid streams. This software is developed to be very flexible to use. It can perform automated, multi-step conversions using batch scripts on possibly millions of streams. The package consists of a GUI for visually piping together a project, running external models to generate or import fluid streams, powered by the core program Streamz. This manual will get the user up and running with the Streamz program using a typical data set and explains the use of each command as they are encountered.

Conventions used in this document

Any reference to "Streamz" in the text refers to the Streamz program. To distinguish between different kinds of text, we use a few typographical conventions:

- Fixed-pitch Courier bold font is used for Streamz keywords mentioned in the text. Required characters will be indicated by upper case; optional trailing characters by lower case or an asterisk. Optional trailing characters may be replaced by anything (or nothing). For example:

TITle**,** CONVert**,** CONVersion**,** CONV***,** COPY*'*

- User-specified data to Streamz keywords are in *italics*:

TITle *title*

- Portions of an actual data-set used in this document are in Courier font, and enclosed within a box:

```
title 'Example conversion of Black-Oil streams'
subtitle 'Converted to 6 component EOS'
```

Definitions

Keywords:

- Command – primary keyword that introduces instructions.
- Sub-command – sub-level keyword that introduces further instructions.
- Option – sub-level keyword that sets parameters.

- Arguments: user-specified input to a keyword.

Introducing Streamz

Streamz is a generic program to convert *fluid streams* from one characterization to another. A fluid stream is any collection of data containing information about the amounts of the constituents of a petroleum fluid, and any other associated information like origin, pressure, temperature, etc. A characterization is a definition of the names of components making up the fluid stream and, in most cases, their molecular weights. Exceptions include black-oil streams, which do not have associated molecular weights. Equation-of-state (EOS) characterizations may also include critical properties and binary interaction parameters.

The user controls a run of Streamz via one or more nested *driver* files, the first of which is the *Primary Input* file. This file (through possible inclusion of other driver files) names the fluid characterizations, describes their properties, and defines conversions between them. It allows the user to specify the opening of *stream* files containing the streams for particular characterizations, and also allows instructions for *filtering*, *combining* and *copying* streams from one file to another. This interaction of Steamz with various files is depicted in Figure 1. Instructions are in the form of commands, sub-commands, options and arguments. All keywords and arguments are **case insensitive** for recognition purposes, but retain their user-specified cases otherwise (in particular for arguments such as title strings and file names). The program automatically invokes conversions among the required characterizations. Typically, various pre-processors generate the program commands, thereby ensuring that the syntax is correct.

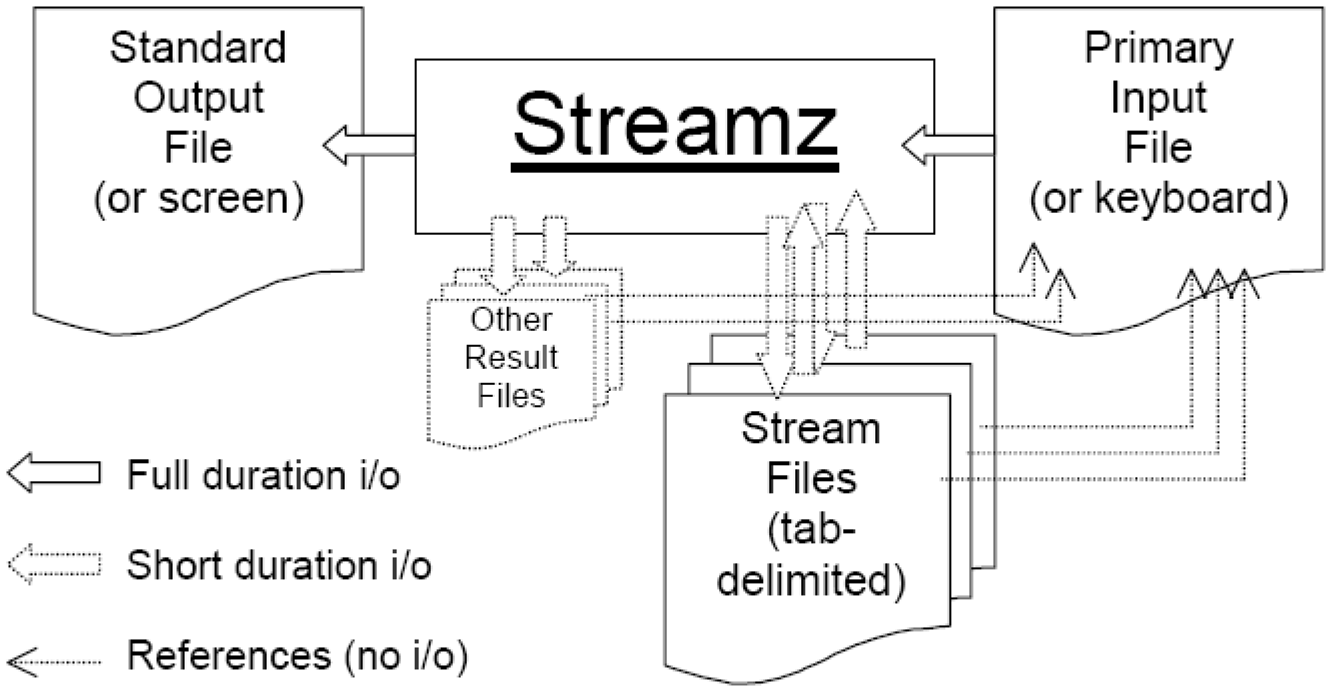

Standard Output File (or screen)
Streamz
Primary Input File (or keyboard)
Other Result Files
Stream Files (tab-delimited)
Full duration i/o
Short duration i/o
References (no i/o)

Figure 1: Streamz flow and I/O

Invoking Streamz

Streamz can be invoked from the operating system's command line using the program name **Streamz**. Up to two command-line arguments can follow the **Streamz** command. The first argument is the name of the Primary Input file, which contains the instructions and keywords to the Streamz program. The second argument is the name of the Standard Output file, to which the program will write various messages about its execution. Path information included with the file names will be considered relative to the current working directory from which the **Streamz** command was issued. Streamz normally requires additional input from one or more stream files and can generate other "result" files, but these can only be specified from within the Primary Input file.

```
Streamz InputFile OutputFile
```

Invoking the program with only the first argument results in a prompt to the user (in the form of a simple text prompt or a standard file dialog box, depending on the operating system) to supply the name of the Standard Output file. If the prompt is canceled, there will be no Standard Output file. If "@" (without the quotes) is given in response to the file dialog box, the Standard Output will be redirected to the screen. Invoking the program with no arguments results in prompts for both the Primary Input and Standard Output files, in that order.

Other methods of invoking the program are:

1. Double clicking the icon for the program (or it's short-cut, alias, or link). This is the same as invoking Streamz without command-line arguments, where the program prompts for both the Primary Input and Standard Output files.
2. Dragging an input file and dropping it on the program's icon (Windows). This is the same as invoking Streamz with only the first command-line argument, where the program prompts for the Standard Output file.
3. Running a batch file or script which contains information for the normal command-line invocation of the program.
4. Invoking the program in any manner without naming a Primary Input file (i.e., canceling the prompt). The program can then be run *interactively*, supplying the commands and keywords to Streamz from the command line, one at a time. The program will run the commands as if they came from an input file. An **EOF** command signals the end of the input.

A Typical Data-Set

This section steps through the "hows" and "whys" of a typical Primary Input file to Streamz. This input file contains a typical usage of the Streamz program containing the most used keywords and their options. This data set is designed to give the first time user a template to start creating his/her own input files that allow using the program straight away. This section will list the full data set and then step through it line-by-line (or portion-by-portion), explaining the syntax and usage of each keyword and option.

Description

The data set is set up to first convert streams, in form of surface oil and gas volumes, from a popular black-oil reservoir simulator (Eclipse 100) into 6-component molar-rate streams. Next, a conversion to 6-component mass-rate streams is performed. The 6-component molar-rate streams are then converted to 17-component molar-rate streams required by a process simulator.

The Data-Set

```
Title 'A data-set for a rich gas condensate field, Opitz'
TITLE 'Black-oil to EOS-6 Conversion'

CHAR 'Opitz BO'
NAME
SO
SG

STREAMFILE IN1:  INPUT "OpitzBO.str"
CHAR 'Opitz EOS-6'
NAME           MW
C1N2             18.640
C2-C6            58.890
```

```
C7P1           112.420
C7P2           179.980
C7P3           310.000
C7P3           480.000

CONVERT 'Opitz BO' from VOLUMES to MOLES
SET        PRES        (bar)        422.073
SPLIT        SO        X1         3.62E-03         8.28E-01         1.01E+00
                                9.36E-01         1.28E+00         2.15E-01
SPLIT        SG        X1         3.84E-02         4.52E-03         1.06E-03
                                2.37E-04        -5.15E-04        -2.35E-04
SET        PRES        (bar)        400
SPLIT        SO        X1        -1.44E-02         8.74E-01         1.06E+00
                                9.68E-01         1.28E+00         1.76E-01
SPLIT        SG        X1         3.84E-02         4.45E-03         9.89E-04
                                1.89E-04        -5.24E-04        -1.75E-04
SET        PRES        (bar)        300
SPLIT        SO        X1        -9.77E-02         1.03E+00         1.24E+00
                                1.07E+00         1.22E+00         9.83E-02
SPLIT        SG        X1         3.85E-02         4.21E-03         7.10E-04
                                3.84E-05        -4.13E-04        -5.18E-05
SET        PRES        (bar)        200
SPLIT        SO        X1        -1.77E-01         1.13E+00         1.42E+00
                                1.14E+00         1.12E+00         7.78E-02
SPLIT        SG        X1         3.84E-02         4.05E-03         4.52E-04
                               -3.97E-05        -2.32E-04        -1.89E-05
SET        PRES        (bar)        100
SPLIT        SO        X1        -1.98E-01         1.09E+00         1.58E+00
                                1.16E+00         1.05E+00         7.17E-02
SPLIT        SG        X1         3.82E-02         4.12E-03         2.57E-04
                               -5.58E-05        -1.10E-04        -7.73E-06
SET        PRES        (bar)        50
SPLIT        SO        X1        -1.27E-01         8.11E-01         1.62E+00
                                1.20E+00         1.07E+00         7.26E-02
SPLIT        SG        X1         3.79E-02         4.52E-03         2.37E-04
                               -6.83E-05        -1.00E-04        -6.87E-06


STREAMFILE OUT1:  OUTPUT 'OpitzEOS6z.str'

COPY

STREAMFILE IN1:  CLOSE
STREAMFILE OUT1:  CLOSE

TITLE 'Opitz EOS-6 to EOS-17 Conversion'
TITLE '(kg-moles/day of EOS-6 to kg-moles/hr of EOS-17)'
TITLE '(also to kg/hr of EOS-6)'

STREAMFILE  IN1: INPUT  'OpitzEOS6z.str'
STREAMFILE  OUT1: OUTPUT 'OpitzEOS6w.str'

CONVERT 'Opitz EOS-6' to MASS
```

```
PROPERTIES 'Opitz EOS-17'
NAME     MW     FULLNAME
CO2             'CARBON DIOXIDE'
N2              'NITROGEN      '
C1              'METHANE       '
C2              'ETHANE        '
C3              'PROPANE       '
IC4             'ISOBUTANE     '
C4              'N-BUTANE      '
IC5             '2-METHYL BUTANE'
C5              'N-PENTANE      '
C6     84.198   'HEXANES        '
C7     97.464
C8     111.251
C9     125.799
C10    139.157
CN1    180.000
CN2    310.000
CN3    480.000

CONVERT 'Opitz EOS-6' from MOLES to MOLES, conserving MASS

GAMMA X3 C7, FILE GAM1
SHAPE 1.570, AVERAGE 1.0, BOUND 0.662693471, ORIGIN 1.0

SET PRES 423 bar
SPLIT X1 CO2   0.03514 0.00462 0.84342 0.11682
SPLIT X2 C3    0.50204 0.07055 0.19427 0.05624 0.08078 0.09611

SET PRES 373.3 bar
SPLIT X1 CO2   0.03459 0.00485 0.84757 0.11299
SPLIT X2 C3    0.51820 0.07066 0.19165 0.05460 0.07709 0.08779

SET PRES 318.2 bar
SPLIT X1 CO2   0.03397 0.00492 0.85170 0.10941
SPLIT X2 C3    0.53182 0.07159 0.19091 0.05227 0.07273 0.08068

SET PRES 263 bar
SPLIT X1 CO2   0.03396 0.00492 0.85388 0.10725
SPLIT X2 C3    0.54201 0.07101 0.18935 0.05089 0.07101 0.07574

SET PRES 207.8 bar
SPLIT X1 CO2   0.03378 0.00495 0.85473 0.10653
SPLIT X2 C3    0.54935 0.07134 0.19025 0.04875 0.06897 0.07134

SET PRES 152.7 bar
SPLIT X1 CO2   0.03429 0.00480 0.85325 0.10766
SPLIT X2 C3    0.55334 0.07151 0.19109 0.04924 0.06800 0.06682

SET PRES 97.5 bar
SPLIT X1 CO2   0.03509 0.00469 0.84823 0.11198
SPLIT X2 C3    0.55519 0.07285 0.18874 0.05077 0.06843 0.06402
```

```
SET PRES 49.3 bar
SPLIT X1 CO2   0.03604 0.00432 0.83710 0.12254
SPLIT X2 C3    0.55296 0.07289 0.19728 0.04956 0.06706 0.06025

STREAMFILE OUT2:  OUTPUT 'OpitzEOS17z.str'

GAMMAFILE  GAM1:  OPEN 'OpitzEOS6.gam'
NEW*SFILE  NWS1:  OPEN 'OpitzEOS17.nws', USER AAZ

COPY, SCALING_BY 0.041666667 ;(convert daily to hourly production)
```

Stepping through the data-set

This section steps through the Primary Input file presented in the previous section.

```
Title 'A data-set for a rich gas condensate field, Opitz'
TITLE 'Opitz BO to EOS-6 Conversion'
```

The purpose of the **TITle** command is to print a boxed title to the standard output file. The quoted string following the **TITle** keyword is centered within a box made up of asterisk (*) characters. The box expands to accommodate the full length of the string. This may be used to visually separate different tasks being run from the same input file. A sub-command recognized within the context of **TITle** is **SUBTitle** (also given by a subsequent **TITle** keyword, as illustrated above). Each of these sub-commands prints its line of text within the same box as the primary **TITle** text. A use of **SUBTitle** after another keyword has been used (i.e., outside the context of the **TITle** command) will result in an error.

```
CHAR 'Opitz BO'
NAME
SO
SG
```

The **CHARacterization** command, or its alias **PROPerties**, names a fluid characterization. In the present example, the characterization name "Opitz BO" is defined.

The **COMPonents** or **NAMEs** sub-keyword triggers the tabular input of an EOS property table, which defines the names and properties (such as molecular weights and critical parameters) of the components that make up the characterization. Because a black-oil characterization is being defined in this example, the property table consists of only the component names. In this case, the names given to the two components are "SO" and "SG" (Surface Oil and Surface Gas). A detailed discussion of the property table is included later.

```
STREAMFILE INP1:  INPUT 'OpitzBO.str'
```

The **STREAMFile** keyword initiates the opening of a stream file for either input or output. The first argument to the keyword is a *file_nickname* (INP1 used here). The sub-command **INPut** directs that an existing file is to be opened for input. Its argument is either the actual file name ("OpitzBO.str" in our example case) or else the keyword **PROMpt**, in case the user wants to be prompted interactively for the actual file name. If the file name is entered here, and contains blanks or unusual punctuation, it should be quoted. It may contain OS specific path directives if the file is in another directory, relative to the current input file. The information in the file being opened should correspond to the "current" characterization. The current characterization is the one just defined (or made "current" with the **RESTore** command, followed by the appropriate *char_name*). In our case it is the "Opitz BO" characterization with two components. The stream data in the file being opened with this **STREAMFile** command should contain the quantity information for two components only.

```
CHAR 'Opitz EOS-6'
NAME           MW
X1               18.640
X2               58.890
X3              112.420
CN1             179.980
CN2             310.000
CN3             480.000
```

Here we have the definition of a second characterization using the **CHAR*** command. We need two defined characterizations to convert from one to the other. This is an EOS characterization named "Opitz EOS-6" with 6 components. The **NAMEs** sub-keyword triggers tabular input of an EOS property table and defines the properties of the components that make up this characterization. This tabular input scheme is very flexible, allowing any of the component properties (including their **NAMEs**) to be input in any order. Each property is identified by a heading keyword. For example, **MW*** (anything beginning with **MW**) indicates molecular weight. The only constraint of the tabular input scheme is that any entries in the table that belong to a particular heading (i.e., property) should *line up* with the heading. This allows unknown entries to be left blank without misaligning the rest of the table. The specific rules for *lining up* the table are described in Appendix B. The name of each component is listed under the **COMPonents** (or **NAMEs**) heading. The value of any other property corresponding to that component is entered in the same row, *lined up* under the heading corresponding to the desired property. In our example case, the molecular weight of X3 is 112.420. This input scheme is designed to enable *cut-and-paste* from any other data file or spreadsheet with minimal editing. The example requires only the **NAME*** and **MW*** properties but input of a full EOS property table is also allowed for forward compatibility.

```
CONVERT 'Opitz BO' from VOLUMES to MOLES
```

The **CONVert** keyword initiates the definition of a conversion procedure to convert streams corresponding to an *input* characterization into streams corresponding to an *output* characterization. The first argument to the keyword is the name of the input characterization (in this case "Opitz BO"). This must have been previously defined. The output characterization is the "current" characterization (the one defined last or made "current" by the **RESTore** command). The **FROM** option defines the input units expected by the conversion procedure. Conversion will be possible only for input streams specified in these (or compatible) units. Four units are currently understood, namely **MASSes**, **MOLEs**, **VOLUMEs**, and **AMOUNTs**. The **FROM** units should match those in the corresponding input stream files (**MASSes** and **MOLEs** are compatible if the component molecular weights have been defined). The actual streams may be in more specific, dimensional units, which may also denote rates, concentrations, fluxes, etc. For example, kgmol, lbmol/day, gmole/cc, or lbmol/ft<sup>2</sup>/sec would all fall under the category of **MOLEs**. The true, dimensional units are not relevant to the program, but should be kept track of by the user, so as not to confuse lbmol/day with kgmol/hr, for example. We use **VOLUMEs** here because we know that the input streams are in Sm^3/D (i.e. volumetric rates). The **TO** option specifies the units of the output streams. A **CONserve** option to the **CONVert** command will be discussed later in this section. If the **FROM** units are not specified, they will default to the **TO** units, the **CONserved** units, or **MOLEs**, in that order of preference. Then, if the **TO** or **CONserved** units have not been specified, they will default to the **FROM** units.

```
SET         PRES         (bar)        422.073
SPLIT         SO         X1        3.62E-03        8.28E-01        1.01E+00
                         9.36E-01        1.28E+00        2.15E-01
SPLIT         SG         X1        3.84E-02        4.52E-03        1.06E-03
                         2.37E-04        -5.15E-04        -2.35E-04
.
.
.

SET         PRES         (bar)         50
SPLIT         SO         X1        -1.27E-01        8.11E-01        1.62E+00
```

```
                       1.20E+00        1.07E+00        7.26E-02
SPLIT        SG        X1        3.79E-02        4.52E-03        2.37E-04
                       -6.83E-05        -1.00E-04        -6.87E-06
```

One way to convert an input stream to an output stream is through a set of *split factors*. Each split factor specifies the portion of a given input component that partitions into a given output component. Each component (input or output) may have several split factors associated with it and the split factors may be functions of one or more *control variables*. Streamz handles the input of these split factors by means of the **SET** and **SPLIT** (or **DELUMP**) keywords, which are discussed in this section. Examples are shown in the above extract, which contains only a portion of the relevant part of the data-set (missing portions are replaced by dots).

The **SET** sub-command is known within the context of the **CONVert** command and is used to specify control variables and set their values. Here we designate the previously defined pressure variable "PRES" (defined in our input stream file by the **VARiable** command, to be discussed later) as our one control variable and set its value initially to 422.073 bar (variables of pressure, temperature, time or distance need to be assigned units—in this case, **BAR**). Our split factors therefore become piecewise linear functions of pressure, starting with those at 422.073 bar. If we had specified additional control variables, the split factors would become piecewise linear functions of those variables as well. The first argument to the **SET** sub-command is the name of the primary control variable, followed by its value and units (if applicable), in either order (the parentheses shown here are optional and are actually ignored). Additional control variables, along with their values and units, could be given as additional arguments, as long as only one input line is used for the entire set of control variables. If the split factors are known to be constant, there is no need for the **SET** keyword.

The **SPLIT** (or its alias **DELUMP**) sub-command is one of the two methods Streamz uses to convert streams (the other being **GAMMA** distribution modeling). This specifies the split factors for conversion of a single input component to one or many output components. A split factor is the fraction of a component in the input stream that goes into a specified component of the output stream. The first argument to this keyword is always the name of the input component. That's followed by a series of *doublets*, each consisting of an output component name and its split factor, in either order. Either element of each doublet may be omitted, however. If the component is omitted, it defaults to the one following that of the previous doublet (with the first doublet defaulting to the first component). If a split factor is omitted, it defaults to 1. The doublets continue until a keyword that is not a component name is encountered. Thus, at 422.073 bar in our example, SO will split into six output components, starting with component X1. The same applies to SG. Since we are **CONVerting FROM VOLUMEs TO MOLEs**, each **VOLUME** of SO will split into 3.62E 03 **MOLEs** of X1, 8.28E 01 **MOLEs** of X2, 1.01E+00 **MOLEs** of X3, 9.36E 01 **MOLEs** of CN1, 1.28E+00 **MOLEs** of CN2, and 2.15E 01 **MOLEs** of CN3. Similarly, each **VOLUME** of SG will split into 3.84E 02 **MOLEs** of X1, 4.52E 03 **MOLEs** of X2, 1.06E 03 **MOLEs** of X3, 2.37E 04 **MOLEs** of CN1, 5.15E 04 **MOLEs** of CN2, and 2.35E 04 **MOLEs** of CN3. Note that this conversion assumes that the actual input **VOLUMEs** will have units of Sm^3 (for both SO and SG) and that the **MOLEs** it produces on output will actually be kgmols. Also note that negative split factors are normal for this type of process-dependent, black-oil to compositional conversion. Here, SG/SO ratios of 915 or more would result in negative EOS mole fractions, but for the process of interest, input streams that are saturated at 422.073 bar should never have SG/SO ratios that high. That being said, the design of process-dependent conversions is beyond the scope of this document.

Split factors are defined for a range of values of the dependent variable. In our example, the streams from a reservoir simulator are expected to be saturated at pressures from 422 bar to 50 bar. Each table uses the **SET** keyword to set the value of the pressure control variable, and uses the **SPLIT** keyword to define the split factors associated with each pressure "node". If a stream (in a stream file being converted) has this associated variable at an intermediate value, linear interpolation is used to calculate the split factors for conversion.

```
STREAMFILE OUT1:  OUTPUT 'OpitzEOS6z.str'
```

The **STREAMFile** keyword has already been discussed for input files. Here it is used with the **OUTput** option, which takes either the name of the file to be created ("OpitzEOS6z.str" in this case) or else the keyword **PROMpt**, in case the user wants to be prompted interactively for the new file name. Because the "Opitz EOS-6" characterization is "current", the streams written to this file will correspond to this characterization.

```
COPY
```

The **COPY** command initiates a read & write operation from all open input stream files to all open output stream files (one of each, in this case). Because the input and output files correspond to different characterizations, each stream needs to be converted before output. The program invokes the relevant conversion definition (in this case split factors are used) automatically. Because the **SPLIT** command is used with the **SET** option, and because the streams in the input stream file have a defined pressure variable named PRES, the split factors will be interpolated (if necessary) and then used to calculate the output streams. For advanced usage, the **COPY** command allows conditional processing (using the **IF** option to select criteria specified by previous **FILTer** commands), variable **WEIGHting** (and **OVERing**), stream **NORMalization**, constant **SCALing**, and selective outputting (using the **TO** option to specify the *file_nicknames* of only the files to which output is desired).

```
STREAMFILE INP1:  CLOSE
STREAMFILE OUT1:  CLOSE
```

The **STREAMFile** command is used with the **CLOSe** option to close the files associated with the supplied *file_nicknames*.

```
TITLE 'Opitz EOS-6 to EOS-17 Conversion'
TITLE '(kg-moles/day of EOS-6 to kg-moles/hr of EOS-17)'
TITLE '(also to kg/hr of EOS-6)'
```

Three **TITle** commands are used. The first is interpreted as the *primary* **TITle** command and the next two are interpreted as **SUBTitle** sub-commands. The result is the output of all three lines, in a single box, to the Standard Output file. This visually marks the start of the next task being run from the same file.

```
STREAMFILE  INP1: INPUT  'OpitzEOS6z.str'
STREAMFILE  OUT1: OUTPUT 'OpitzEOS6w.str'
```

Two stream files are associated with *file_nicknames* ("INP1" and OUT1") and opened. The first ("OpitzEOS6z.str") is the same as that used for **OUTput** in the previous task. It is being used for **INPut** now, allowing intermediate results from the same run of Streamz to be used for subsequent conversions. The second is a new **OUTput** file. Since both **STREAMFile** commands are issued one after the other, both will be associated with the same, "current" characterization ("Opitz EOS-6"). As we will soon see, this is exactly what is required.

```
CONVERT 'Opitz EOS-6' to MASS
```

This **CONVert** command defines a conversion from the "Opitz EOS-6" characterization to the "current" (also "Opitz EOS-6") characterization. Conversions from any characterization to "itself" are always defined, as long as the **FROM** and **TO** units are compatible. The only reason for specifying this type of self-conversion is to force a change of units (**TO MASSes**, in this case) for each stream processed. The **FROM** units aren't specified, so by the rules previously outlined, they are assumed to equal the specified **TO** units, **MASSes**. The **CONVersion FROM MASSes** (of "Opitz EOS-6") **TO MASSes** (of "Opitz EOS-6") is trivial (and even if non-trivial **SPLIT** factors were entered, they'd be ignored). It will obviously work for input streams given in **MASSes**, but it will also work for input streams given in **MOLEs**. That's because the "Opitz EOS-6" characterization includes the molecular weights of all its components, making **MOLEs** and **MASSes** completely compatible (i.e., internally convertible) for this characterization. In either case, all output streams will be given in **MASSes**. The next **COPY** command will automatically invoke this **CONVersion TO MASSes** when it copies the streams from INP1 to OUT1, since both of these files are associated with the "Opitz EOS-6" characterization.

```
PROPERTIES 'Opitz EOS-17'
NAME     MW     FULLNAME
CO2             'CARBON DIOXIDE'
N2              'NITROGEN      '
```

```
.
.
.
C6      84.198   'HEXANES        '
.
.
.
CN2    310.000
CN3    480.000
```

We now define a new, 17-component characterization. Reproduced above is only a portion of the original data-set. Note the use of the **PROPerties** keyword instead of **CHAR***. They both mean the same. Note also the use of a new property (**FULLname**). This allows names of components containing embedded spaces, which might be required in some usages. The **FULLname** property might also be used to associate a remark with the component. Currently this property is used for output to an in-house process simulator format.

```
CONVERT 'Opitz EOS-6' from MOLES to MOLES, conserving MASS
```

This command defines the conversion from the "Opitz EOS-6" characterization to the current ("Opitz EOS-17") characterization. The conversion is defined from molar streams to molar streams. Note the use of the **CONserving** option. This allows the user to specify the quantities to conserve during the conversion (**MASSes**, in this case). Use of the **CONserve** option has two effects. First, the split factors will be checked for possible material balance errors; if they will not conserve the requested quantities, warnings will be issued. Second, it determines the way Gamma distribution modeling is performed. Gamma modeling can conserve either moles or mass, but generally not both. Moles are conserved by default, but the option to **CONserve MASSes** can be used instead. Any combination of **FROM**, **TO**, and **CONserved** units may be specified, but the **CONserve** option has an effect only if the **CONserved** units are compatible with both the **FROM** and **TO** units. Otherwise, it is ignored.

```
GAMMA X3 C7, FILE GAM1
SHAPE 1.570, AVERAGE 1.0, BOUND 0.662693471, ORIGIN 1.0
```

The **GAMMA** option is another possible method of defining the conversion between characterizations (the first being the **SPLIT** option). It can be used just by itself, if the characterization contains only heavy components for which a continuous distribution is a good approximation (e.g., heptanes plus, C_{7+}, or decanes plus, C_{10+}). Or it can be used for a heavy-end subset of the components, with split factors being used for the remaining, lighter components. It expects 2 mandatory arguments in the form of the names of one component each of the input and output characterizations. These are the lightest components in each which are requested to participate in the Gamma modeling. The molecular weights and the amounts of all input components as heavy as, or heavier than, that specified would be used to calculate a gamma distribution model. The model, and the molecular weights of all output components as heavy as, or heavier than, that specified, would then be used to calculate the amounts of the output stream. In the example, we specify that X3 and heavier components (MW-wise) should be fit to a Gamma distribution model. We also specify that C7 and heavier components of the output streams will receive amounts based on the calculated model.

Without going into the mathematical details here, the Gamma model describes a continuous molar distribution as a function of molecular weight. The function itself is defined for molecular weights from an *origin* value to infinity, but only the molecular weights greater than or equal to a *boundary* value (greater than or equal to the *origin* value) are considered for the molar distribution. The molar distribution has an *average* MW and the function has a particular *shape*, which can (a) decay exponentially from a finite value at the origin MW, (b) decay faster than exponentially from an infinite value at the origin MW, or (c) decay slower than exponentially after rising from zero at the origin MW and going through a maximum. Shapes of type (b) are typical for gas condensates and shapes of type (c) are typical for heavy oils, while shape (a) falls in-between. The distribution is described by four parameters: one for the shape and three others for the origin, boundary, and average molecular weights.

The calculation of the model is essentially the determination of the four model parameters by means of regression. The user has control over the regression by specifying the starting values and the upper & lower bounds of these parameters. The four model parameters are specified by optional sub-keywords known within the context of the **CONVert** command. These sub-keywords are **SHAPE**, **BOUNDary**, **AVErage**, and **ORIGin** (or **ZERO***). The exponential shape (a) has a **SHAPE** parameter of 1. Shapes of type (b) have **SHAPE** parameters less than 1 (typically no less than 0.4) and shapes of type (c) have **SHAPE** parameters greater than 1 (typically no greater than 5). The model's average MW is given by the product of the **AVErage** parameter (typically around 1) and the calculated average MW of the portion of the input stream being modeled. The model's boundary MW is given by the product of the **BOUNDary** parameter (between 0 and 1) and the MW of the input component specified as the first argument to the **GAMMA** keyword. The model's origin MW is given by the product of the **ORIGin** parameter (between 0 and 1) and the model's boundary MW.

Each of the four parameter keywords may be entered with zero to three numerical arguments. If a particular parameter keyword is entered without any arguments (or if it is not entered at all), its default initial value and bounds will be used during regression. If it is entered with at least one argument, the first argument is taken as the parameter's initial value, the minimum argument is taken as the lower bound for regression, and the maximum argument is taken as the upper bound. If the initial value and both bounds turn out to be equal (e.g., when only one argument is entered), then that single fixed value will be used for the model and it will not be altered by regression. In our case we have fixed all the parameters. They were calculated previously by fitting a sample of the fluid to the Gamma distribution model. See Appendix A for an example use of starting values and upper & lower bounds for the parameters.

An optional sub-keyword within the context of the **GAMMA** keyword is **FILE**. This specifies the nickname of the file to which gamma model results are written. An actual file needs to be opened with the **GAMMAFile** command and associated with this nickname before the results will actually be written. We use the *file_nickname* "GAM1" in this example.

```
SET PRES 423 bar
SPLIT X1 CO2   0.03514 0.00462 0.84342 0.11682
SPLIT X2 C3    0.50204 0.07055 0.19427 0.05624 0.08078 0.09611
.
.
.
SET PRES 49.3 bar
SPLIT X1 CO2   0.03604 0.00432 0.83710 0.12254
SPLIT X2 C3    0.55296 0.07289 0.19728 0.04956 0.06706 0.06025
```

Again we have reproduced only a portion of the relevant part of the data-set. The current conversion definition uses a combination of **GAMMA** and **SPLIT** methods. While the X3-plus to C7-plus conversion is covered by the **GAMMA** command, pressure-dependent split factors are specified for the conversion of the other components. X1 of input streams splits into C02 and the next 3 components (based on order of definition in **CHAR*** command) of the output streams. X2 splits into C3 and the next 5 components. The splitting is a function of pressure where interpolation is used if required.

```
STREAMFILE OUT2:  OUTPUT 'OpitzEOS17z.str'
```

This specifies the *file_nickname* and the actual disk file where streams corresponding to the "current" characterization ("Opitz EOS-17") will be written.

```
GAMMAFILE  GAM1:  OPEN 'OpitzEOS6.gam'
NEW*SFILE  NWS1:  OPEN 'OpitzEOS17.nws', USER AAZ
```

The primary **GAMMAfile** command **OPENs** (also possible with the **FILE** or **OUTput** keywords), **CLOSes**, or **PROMpts** for a file where the results of gamma modeling are to be written. It first associates a *file_nickname* (GAM1, in this case) with this file. This *file_nickname* must be used when **CLOSing** the file or when specifically directing output to it (with the **FILE** option of the **GAMMA** sub-command of the **CONVert** command). The **OPEN*** sub-keyword takes the name of the file to be created ("OpitzEOS6.gam" in this case) as its argument. Gamma files are typically used for a small number of streams (unlike in the present case).

The primary **NEWSfile** (or **NEW*Sfile**) command provides temporary functionality to create output stream files in an in-house process simulator format. It **OPENs** (also possible with the **FILE** or **OUTput** keywords), **CLOSes**, or **PROMpts** for a file of this type. It first associates a *file_nickname* (NWS1, in this case) with this file. This *file_nickname* must be used when **CLOSing** the file or when specifically directing output to it (with the **TO** option of the **COPY**, **TABUlate**, or **WRITE** commands). The **OPEN*** sub-keyword takes the name of the file to be created ("OpitzEOS17.nws" in this case) as its argument. An additional sub-keyword (required by the proprietary process simulator format) is **USER**, which takes the initials of the user as its argument ("AAZ" in the example). The **NEWSfile** command is likely to be eliminated from Streamz in a future version.

```
COPY, SCALING_BY 0.041666667 ;(convert daily to hourly production)
```

The **COPY** command copies the streams from all open input stream files to all open output stream files (or a subset specified by the **TO** option), converting the streams as needed. The **SCALing** option multiplies the output streams by a constant factor. This can be used for conversion from one set of units (e.g. moles/day) to another (moles/hour), for example. Other options are illustrated in Appendix A and detailed in the Keyword Reference Manual.

A Typical Stream file

This section discusses the format of a typical stream file used by Streamz. It contains the most used stream file keywords and is designed to give a first-time user a template to start creating his/her own stream files and start using this program straight away. Since it is absurd to reproduce a complete stream file, because they are usually huge, we extract only a portion for the purpose of discussion. The next section will step through it line-by-line (or portion-by-portion) explaining the syntax and usage of each keyword.

Description

The first stream file (#1) is basically the first input stream file used by the data set discussed in the previous section. It contains surface oil and gas volumetric rates from a popular black-oil reservoir simulator (Eclipse 100). The rates are given for each well for each connection. For a particular connection, all the data for all time-steps are written, before moving on to the next connection. This is just one way the results might be organized. In fact this stream file was produced by one of the pre-processors of the PetroStream Management software suite. The example file contains stream data for 2 connections for 2 wells. The time steps are truncated to show only a sample. In reality there could be hundreds of time-steps for each connection (themselves numbering from 10 to over 100) for each well. Such stream files can be huge.

The second stream file (#2) is the output of the first conversion from the black-oil streams to the 6-component molar streams. We look at it to understand how an output stream file from Streamz looks, how it can be used as an input stream file, and how it differs from a black-oil stream file.

An important point to note is that stream files are *tab-delimited*. This means that each individual piece of information (i.e. each record) is separated from the next by a *tab* character. It is important that such files are not edited in a text editor that removes the tabs and converts them to spaces. The reason for use of tab-delimited files is the ease of import into Microsoft Excel for further manipulation. It is possible to drag-n-drop such files into MS Excel and have them opened with all data in the correct cells.

Stream Reference Manual

Currently this is available as PDF only

PSM-StreamzManual.pdf

Streamz TABULATE command reference

(This supersedes the TABULATE entry in the Streamz Reference Manual)

```
TABUlate [(var1|dom1) units [AND (var2|dom2) units ...]] [IF fltr]

    [FROM infile1 [AND infile2 ...]] [TO outfile1 [AND outfile2 ...]]
    [WEIGHt [OVER|PER|BY] (var4|dom4) units [AND (var5|dom5) units ...]]
    [OVER (var6|dom6) units [AND (var7|dom7) units ...]]
    [PER (var8|dom8) units [AND (var9|dom9) units ...]]
    [DISPlay (var10|dom10) units [AND (var11|dom11) units ...]]
    [(COLLATe|ORDER*|ACCRUe) [dom12 units [val1 [val2 ...]

        [STEP units step1 [step2 ...]]]]]
```

PER and WEIGHt PER are new options that apply to the COPY, COMBine, TOTAL, TABUlate, PROCess, and REALLOCate commands. PER is very similar to OVER, except for when it gets applied. For output streams that are created from just a single input stream (as is always the case with the COPY, PROCess, and REALLOCate commands), PER and OVER give identical results. When an output stream is composed of several constituent streams, however (as can be the case with the COMBine, TOTAL, or TABUlate commands), the effects might be different (although not necessarily). OVER is applied once for each constituent stream (you can remember this as "over and over"), whereas PER is applied just once "per" output stream. Generally, OVER will give you an average constituent rate of some sort, whereas PER will give you a combined average rate, which might be different. Detailed explanations, along with the actual formulas, are given in the associated document "Weighting.pdf" (Weighting.pdf). DISPlay (alias SHOW*) is another new option for the TABUlate command. It can be used to display the output values of variables other than those tabulated and collated. Variables that are simply DISPlayed will have no effect on how the tabulations or collations are performed. They will just be output whenever they can be determined unambiguously. Specifically, an ordinary variable will be shown whenever its value is the same for all constituents of an output stream, while domain variables will be adjusted to show the encompassing domains for each output stream.

The most powerful new feature, though, is the COLLATe (alias GATHER*) option and the related options ORDER* (alias REORDER*) and ACCRUe (alias INTEGRAte). As you can see, each one of these options has several options of its own. Some examples:

1. COLLATE
2. COLLATE time days
3. COLLATE time years 0 1 2 3 4 5 d.
4. COLLATE time months 0 1 2 3 STEP months 3*3 4*6 2*12

Explanations:

1. Collate all of the streams for which the tabulated variables match.
2. Same as (a), but also track and output the overall range of a domain.
3. Same as (a), but divide the collated streams into two or more domain ranges specified by explicit collation points.
4. Same as (c), but allow one or more collation points to be specified explicitly, with the remainder specified by a list of interval steps.

Let's assume "TABULATE WELL" is the primary command and that your input files include streams from 10 unique wells (satisfying any specified filter). Here are the results you'd get from the above options:

1. Ten total output streams (one for each well), where each stream would contain the name of the well and the sum of all the input stream amounts associated with the corresponding well. That sum could also be WEIGHted to convert rates to cumulatives, OVERed or PERed to convert cumulatives to average rates, or WEIGHted OVER or WEIGHted PER to convert rates to average rates.
2. Same as (a), except the total time range for each well (in days) would also be shown for each output stream.
3. As many as 70 total output streams (up to 7 for each well). For each well, you would get the cumulative or average amounts (depending on the WEIGHt, OVER and PER options) for time ranges that might include (but only if non-empty): minimum to 0 years, 0-1 years, 1-2 years, 2-3 years, 3-4 years, 4-5 years, and 5 to maximum years.
4. As many as 140 total output streams (up to 14 for each well). For each well, you would get the cumulative or average amounts (depending on the WEIGHt, OVER and PER options) for time ranges that might include (but only if non-empty): minimum to 0 months, 0-1 months, 1-2 months, 2-3 months, 3-6 months, 6-9 months, 9-12 months, 12-18 months, 18-24 months, 24-30 months, 30-36 months, 36-48 months, 48-60 months, and 60 to maximum months.

Now let's assume that "TABULATE WELL AND TIME (MONTHS)" is the primary command. The results of cases (b), (c), and (d) remain essentially the same, except the times will always be output by months, regardless of the units given for the collation points. Case (a) will behave quite differently, however. It will now collate all the streams that have unique combinations of WELL, starting TIME, and ending TIME. That could be any number of streams (as few as 10 or as many as the total number of input streams).

As you can see, you'll have a lot of flexibility at your disposal. As for the specific question about reporting cumulative molar productions, let's assume that you simply want monthly cumulatives for each well over a 10-year period (for example). If your input streams are already in terms of cumulatives, you would say:

TABULATE WELL, COLLATE TIME MONTH 0, STEP MONTHS 120*1

On the other hand, if your input streams are in terms of daily rates, you would simply add "WEIGHT TIME DAYS" to the above command. Then, if you decided you wanted average daily rates for each well over those monthly time periods, you would just add "OVER TIME DAYS" or (more likely) "PER TIME DAYS" to the previous command, depending on how your input data were arranged and the type of average you wanted (see Weighting.pdf for more information).

If you replace the COLLATe keyword with ORDER* (leaving all other input the same), the output streams will be sorted according to the following criteria:

Highest priority: Output stream units (in case your conversions lead to differences). Amounts will come first, then Volumes, Moles, and finally Masses.

Next priorities: Tabulated variables (except those belonging to the ORDERed domain, if any). Priority will be assigned in the order the variables (or domains) were specified by the TABUlate command (with lower domain variables taking priority over their upper counterparts). The variables will be sorted into ascending order (numerically or alphabetically), with undefined variables coming last.

Final priority: The ORDERed domain, which will be sorted into ascending order.

DISPlayed variables will have no effect on the sorting.

The final option uses the keyword ACCRUe. If you use this in place of the ORDER* keyword, each set of ordered results (having the same tabulated variables aside from the ordered domain) will be further combined into running totals before being output. All of the other previous rules for the ORDER option (as well as for the WEIGHT, OVER and PER options) still apply.

PSM GUI Manual

Currently this is available as PDF only

PSM Toolkit Manual

Currently this is available as PDF only