

# **PETROLEUM REFINING LIBRARY (PRL) FOR ANYLOGIC**

## **USER MANUAL**

[www.prlsim.com](http://www.prlsim.com)

version 1.1  
26.07.2025

# Contents

1 INTRODUCTION .....	3
1.1 What is Petroleum Refining Library? .....	3
1.2 Petroleum Refining Library license .....	3
1.3 Documentation license .....	3
1.4 About this manual.....	3
1.5 Acknowledgements.....	3
2 PETROLEUM REFINING LIBRARY STRUCTURE.....	4
2.1 Petroleum Refining Library structure.....	4
2.2 FREE TO TRY version.....	7
3 COMPATIBILITY .....	8
4 FIVE STEPS TO RUN PETROLEUM REFINING LIBRARY .....	8
5 USING PETROLEUM REFINING LIBRARY .....	9
5.1 Source.....	9
5.2 Plant .....	10
5.3 Control.....	11
5.4 SeparatorLight .....	12
5.5 MixerLight.....	12
5.6 Separator .....	13
5.7 Mixer .....	13
5.8 ShipmentNode .....	13
5.9 ReservoirPark.....	14
5.9.1 RpFlowing .....	15
5.9.2 RpAccumulative .....	15
5.10 Optimizer .....	16
5.11 GasOwnNeeds.....	16
5.12 Dispose .....	16
5.13 FlowQuota .....	17
5.14 ProductMixer .....	17
5.15 LoadingRack.....	18
4 AUXILIARY DATABASE .....	18
4.1 Database references .....	19
4.2 Database tables .....	20
5 CREATING AND EXECUTING THE PRODUCTS PRODUCTION “REQUESTS” .....	28
6 PETROLEUM REFINING LIBRARY ENUMS .....	29
7 RECIPES .....	29
8 TYPES OF MEASURING .....	29
9 FAQ .....	29

# 1 INTRODUCTION

## 1.1 What is Petroleum Refining Library?

**Petroleum Refining Library (PRL)** is an additional palette library for Anylogic ([www.anylogic.com](http://www.anylogic.com)). It is based on Anylogic Fluid Library and helps to build a different kind of imitation models for Oil & Gas industry. Like Anylogic, PRL fully supports Java.

Base PRL features:

- building a different kind of imitation models like “digital twins” in Oil & Gas industry;
- performing an accurate “balance” calculations of incoming flows and resulting products;
- make detailed optimization flows calculations within Oil & Gas refining;
- taking into account all unique features of specific Oil & Gas companies for maximum accurate forecasting;
- setting and solving any tasks of the current processing management in Oil & Gas industry.

Editions of PRL:

- free edition – “free to try” version with limited performance and license;
- commercial edition – high-performance version of PRL with business-friendly license;
- advanced commercial edition – commercial edition of PRL with full source library code.

Free Edition of PRL has a corresponding warning on every page, and has a limit on model time duration (max 45 days). Commercial Edition is a heavily optimized version of PRL, containing all library components without any restrictions.

There is also an advanced commercial PRL license, under which you’ll receive full source code.

## 1.2 Petroleum Refining Library license

The PRL free edition is distributed under a license that favors non-commercial usage and operates under a “free to try” principle.

The PRL commercial edition and advanced commercial edition are distributed under a license that is friendly to commercial users. A copy of the commercial license can be found [here](#)

## 1.3 Documentation license

This reference manual is licensed under BSD-like documentation license:

Copyright 2025 by Petroleum Refining project. All rights reserved.

THIS DOCUMENTATION IS PROVIDED BY THE PETROLEUM REFINING PROJECT “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PRL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 1.4 About this manual

This manual provides general information about PRL. It contains a description of every PRL agent and some examples. To know more, please visit our [website](#), or [YouTube channel](#).

On our [website](#) you can also found a Java API that describes all available PRL functions.

## 1.5 Acknowledgements

PRL was not possible without contributions of the following open-source projects:

- **IpSolve** – mixed Integer Linear Programming (MILP) solver. IpSolve solves pure linear, (mixed) integer/binary, semicont and special ordered sets (SOS) models;
- **java ilp** - a java interface to integer linear programming (ILP) solvers.



You **MUST** add these libraries to your Anylogic model before using PRL. Working versions of these libraries are always located in the **\lib\** subfolder

Jar files and class folders required to build the model:

Location
javailp-1.2a.jar
lpsolve-5.5.2.0.jar
lpsolve55.jar

Buttons: + (Add), - (Remove)

## 2 PETROLEUM REFINING LIBRARY STRUCTURE

### 2.1 Petroleum Refining Library structure

The *PrlConstants* class contains *final static* constants for PRL. Full description can be found in PRL API.

All PRL agents extend from the *PrObject* agent. They can be divided into two categories: agents with interface (such as *Plant* or *RpAccumulative*), which extend from *PrGUIObject* (*PrGUIObject* extends from *PrObject*), and agents without interface (such as *Source* or *Separator*), which extend directly from *PrObject*.

All PRL agents have an inner *Constant* class (accessed via the *public constant* variable) that contains local additional Agent settings.

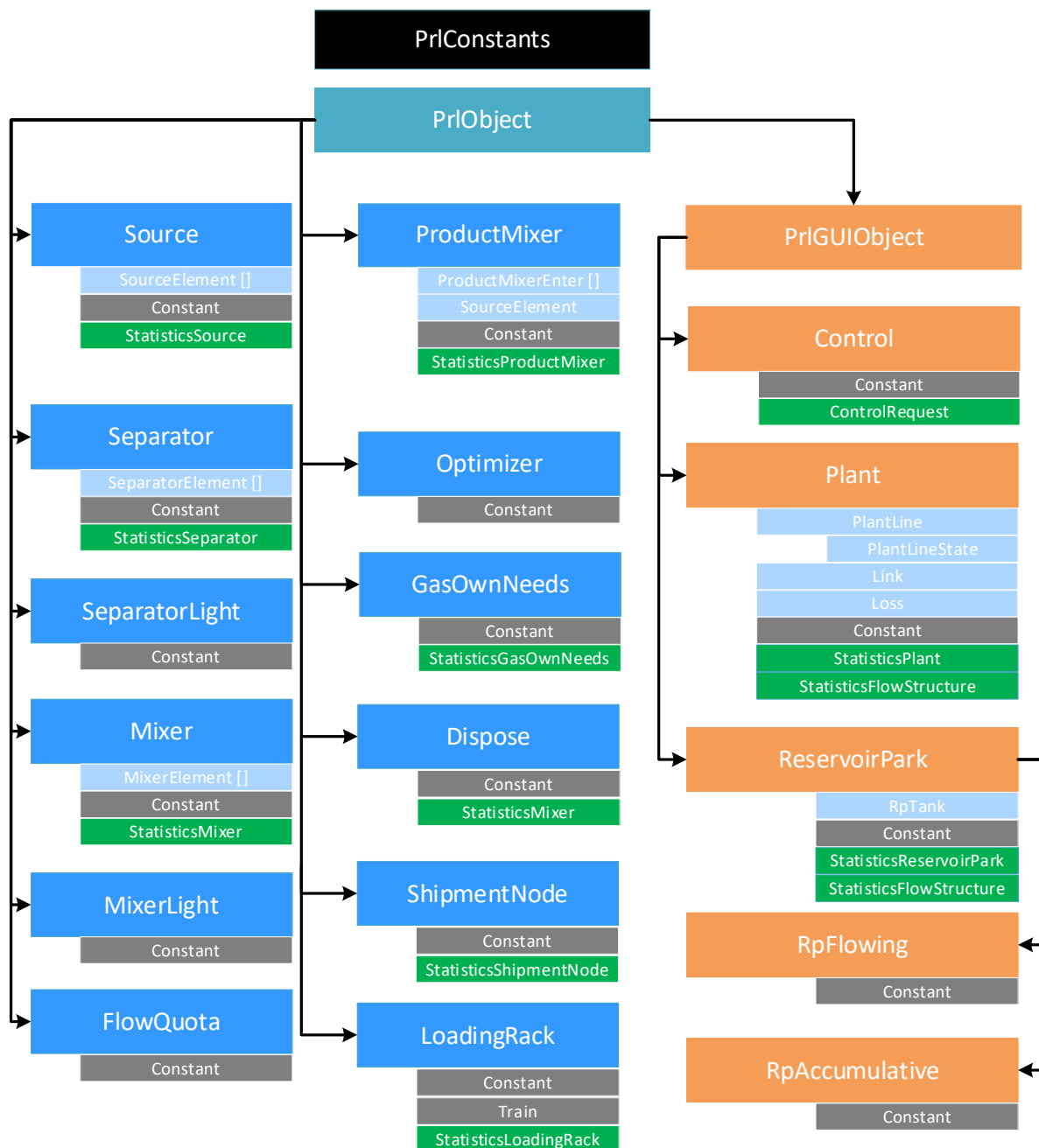
Many agents also contain an inner *StatisticsAgentName* class (accessed via the *public statistics* variable) to access various statistical indicators (enter or exit flow rates, flows mass, etc.). Full description can be found in PRL API.

Some agents also contains inner agents. For example, *Source* agent contains an inner *SourceElement* agent for modeling incoming flow from a single source (oil or gas field), *Plant* agent contains an inner *PlantLine* agent for working with the single line.

Some parameters values can be set in \*.ini files. The name of each \*.ini file and it's location are formed according to the following rule: the directory name and the name of the file, created by the agent class name (for example:

***BaseConstant.INI\_DIRECTORY\_PATH + Source.class.getSimpleName() + ".ini"***

will search \*.ini file in: "*Vni\Source.ini*"). To access a specific agent in the \*.ini file, create a section with its system name (in square brackets). For example: *[sourceAgent1]*.



## MAIN PETROLEUM REFINING LIBRARY AGENTS (AVAILABLE IN ANYLOGIC PALETTE)



**Source** is an advanced container that allows to create various types of entry flows (oil, unstable natural gas liquid, etc.). Source agents enable to set daily or monthly flow plans from the attached database, facilitate flow smoothing for monthly values, and provide many other functions



**Plant** is an agent, that allows for the configuration of the flow-splitting process by various types of processing plants in the Oil & Gas industry (such as rectification, catalytic reforming, etc.). The Plant agent contains a wide variety of settings that enable the establishment of basic operating conditions for almost any kind Plant in Oil & Gas



**RpFlowing** is an advanced park agent that extends all Reservoir park agent properties. RpFlowing is characterized by continuous inflows and outflows that fill and unload some parallel connected tank (called "maps"), RpFlowing is usually used for "smoothing" outgoing flow products



**RpAccumulative** is advanced park agent, that extends all Reservoir Park agent properties. RpAccumulative contains many options, which are necessary for "cumulative" reservoir parks. This agent implements procedures of accumulation, passportization (compounding), and shipment of products. In addition, RpAccumulative contains built-in mechanisms for tracking product shipment plan with the ShipmentNode agent



**ShipmentNode** is an agent that allows to set plans for pumping products and monitor the degree of their implementation during modeling. This agent contains a variety of settings that allow to create different options for pumping products (daily, monthly) etc.



**Control** is an agent that collects all requests from RpAccumulative and ShipmentNode agents and tracks their completion level. Control agent allows to make an effective request management because it contains some Java Interface elements that enable setting a different rules for processing requests



**SeparatorLight** is an advanced separator (split) agent that allows to split the input flow into two outputs. In comparison with the AnyLogic FluidSplit, this agent enables to direct the entire flow to ONLY ONE output, set the input flow speed limit, and implement many other functions



**MixerLight** is an advanced mixer (merge) agent that allows to combine two entry flows into one exit flow. In comparison with the AnyLogic FluidMerge, this agent enables to set an exit flow speed limit, and implement many other functions



**Separator** is an advanced separator (split) agent that allows for splitting an incoming flow into any number of exits. In comparison with the AnyLogic FluidSplit, this agent allows for directing the entire flow to ONLY ONE output, setting as many exits as the user wants, establishing a user-defined separation factor, entering a speed limit, and implement many other functions



**Mixer** is an advanced mixing (merging) agent that allows for merging of any number of input flows into one output. In comparison with the AnyLogic FluidMerge, this agent allows users to set as many input flows as desired, establishes an input speed limit, and implement many other functions



**ProductMixer** is an advanced product mixing agent, that allows for the optimal mixing of input flows and product additives to create a desired product (such as petrol, kerosene, etc.). It is known that the creation of most products is accompanied by a number of requirements (such as octane number, sulfur content, saturated vapor pressure, aromatics, etc.). This agent takes these requirements into account and mixes incoming flows in an optimal way to ensure their simultaneous fulfillment



**Optimizer** is an agent, that allows to set up and solve different kinds of linear programming (LP) problems (for example, splitting the input flow between plants and plant lines with the objective of minimizing residuals). The Optimizer agent can solve two types of LP problems.

The first type is "parametric", where user has a ready-made IpSolve file (\*.lp) with specific parameters. In this case, the Optimizer will automatically replace the parametric texts in the \*.lp file with numeric values, solve the LP problem, and save it with a new name. This type of task is useful for complex LP problems when the user needs to see all the equations directly, but it is quite slow.

The second type allows the user to create own LP problem equations and solve it directly, by using the internal Optimizer agent Java ILP Interface



**GasOwnNeeds** is an agent, that allows to calculate the amount of gas for own needs (GON) and distribute it among Plants. As usual, the GON flow depends on the season, list of Plants, their modes, and load levels



**FlowQuota** is an agent that allow to set periodic plans for pumping products. For example, set a plan for the maximum monthly (or daily) pumping. Required to account for the maximum capacity of the pumping equipment



**LoadingRack** controls the loading of refined products into railway tanks/trucks at a loading rack, managing train schedules, loading operations, wagon assignments, product quantities, supply/removal durations, passportization, and overseeing the state machine for the loading workflow (supply/load/remove)



**Dispose** is an agent, that allows to calculate the amount of disposed flows. In comparison with AnyLogic FluidDispose, this agent enables the collection of the same type of flows into one agent, saves the necessary statistics, and destroys them (for example, collecting all gas flows per flare)

## ADDITIONAL PETROLEUM REFINING LIBRARY AGENTS (NOT AVAILABLE IN ANYLOGIC PALETTE)

**PrlConstants** is an internal PRL class, that contains all base constants. Since all variables in the *PrlConstants* class are *final* and *static*, user can call them directly. For more information, please refer to the PRL API



```
String unit = PrlConstants.TONS_UNIT; //Return ton(s)
```

**PrlObject** is the base agent for all PRL agents. It contains basic methods that are available for use in a model. For more information, please refer to the PRL API

**PrlGUIObject** is the base agent for all PRL agents with an interface. It extends *PrlObject* agent and contains basic interface elements

**SourceElement** is an advanced source agent, that allows to create a various types of incoming flows (such as oil, unstable natural gas liquid, etc.). *SourceElement* is located in the *Source* and *ProductMixer* (as an additive) agents

**Link** is an agent that allows to accept a flow on *FluidEnter* and pass it to *FluidExit*. Required for internal work with PRL flows

**Loss** is an agent that allows to destroy the incoming flows. Required for working with loss flows in *Plant* and *ReservoirPark* agents

**Constant** is an internal PRL class that contains a list of constants for a specific type of agent. This class also includes a built-in *StringBuilder*. All constants are accessed via the attached agent's internal *public* variable *constant*. For more information, please refer to the PRL API



*// ReInitialization a constant*

```
self.constant = self.new Constant().builder().MENU_IF_USE_DIRECT_FLOW(false).build();
```

**ProductMixerEnter** is an agent that allows for the separation of the *ProductMixer* entry flow for mixing and residuals. *ProductMixerEnter* is located within the *ProductMixer* agent

**PlantLine** is an agent, that allows to configure *Plant* line settings. *PlantLine* is located in the *Plant* agent

**PlantLineState** is an agent that represent one of line plant state

**RpTank** is an agent that allows to model tank for OIL and Gas refining

**MixerElement** is an agent, that allows to mix to flows to one

**SeparatorElement** is an agent, that allows to separate one flows to two

**PlantLineState** is an agent that allows for the configuration of plant line states, including "temporary" states. *PlantLineState* is located within the *PlantLine* agent



*Many Plants in Oil & Gas refining have a temporary states. For example, if plant is used to make aviation kerosene, it must be in a temporary mode for a day before switching to aviation kerosene production mode*

**ReservoirPark** is an agent that allows to configure the basic options for *ReservoirPark*. *ReservoirPark* is a base class for *RpFlowing* and *RpAccumulative* agents

**ControlRequest** is an internal PRL class which contains one shipment request for *A-D priorities* from *RpAccumulative* and *ShipmentNode* agents in the *Control* agent

**StatisticsAgentType** is an internal PRL class, that contains various statistics information about the agent. As usual, this class includes *Enter*, *Exit*, and *Database* subclasses. For *Reservoir Park*, it also contains a *Remains* subclass, and for *ProductMixer*, it includes a *Constrain* subclass. All methods are accessed via the internal *public* variable, *statistics*. For more information, please refer to the PRL API

**StatisticsFlowStructure** is an internal PRL class, that allows for the calculation of a current flow structure at the entrance of a *Plant* or *Reservoir Park*. Flow structure is an important indicator, as it affects the values of the splitting recipes

## 2.2 FREE TO TRY version

PRL comes in three versions: one "free to try" version and two commercial versions.

The "free to try" version contains the following restrictions:

- limited set of PRL agents;
- message "THIS IS THE FREE VERSION TO TRY; PLEASE BUY A FULL VERSION" on every PRL agent page;
- maximum simulation duration of 45 days.

The "free to try" version is necessary for familiarizing oneself with the basic capabilities of PRL and solving simple business problems. To build full-fledged models, please purchase the commercial version.

### 3 COMPATIBILITY

This PRL version is compatible with every modern version of Windows (Vista, 7, 10, 11, etc.). PRL was developed in AnyLogic 8 and will be updated along with the base AnyLogic product. Stay tuned for updates on our [website](#)

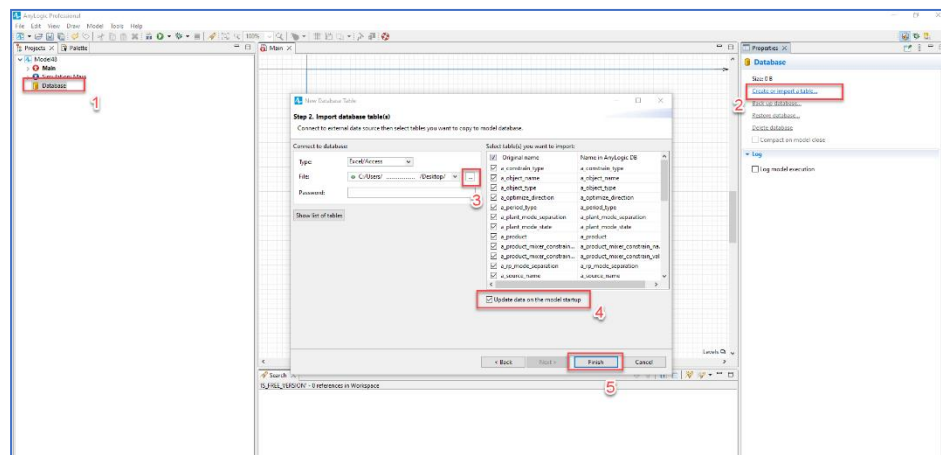
### 4 FIVE STEPS TO RUN PETROLEUM REFINING LIBRARY

**Must-read!** To successfully work with PRL in AnyLogic, you **MUST** follow these steps:

1. Please click on **Palettes -> Manage Libraries -> Add**. Add PRL library to your model.
2. Add a PRL database to your model. You can find the **DB\_Refining.accdb** database in the PRL folder.



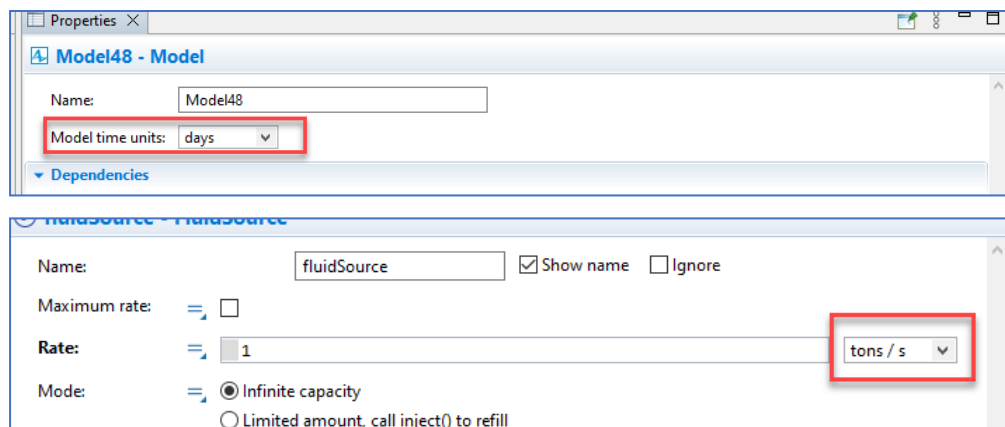
At default PRL database is based on Microsoft Access



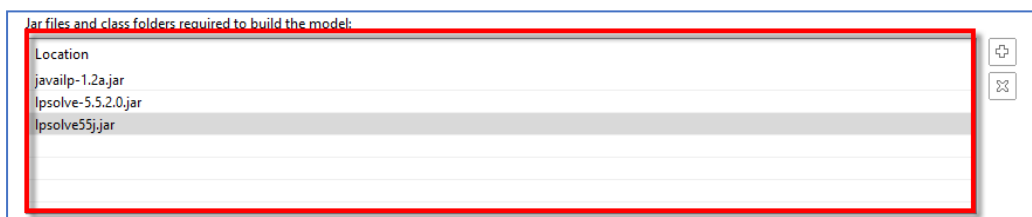
- 3 Set Anylogic model time units to **days** and all Source rates (if you have it) to **mass (tons/s or kilograms/s)**



To ensure the **law of conservation mass**, all flows **MUST** be specified in units of mass. It is also possible to show flows values in other measure units (for example speed), by using special PRL classes and methods



- 4 Add required PRL external \*.jar libraries to your model (you can found them in lib\ subfolder)





5 Remember that ALL *FluidExit* and *FluidEnter* Anylogic elements that you are using with PRL library MUST allow for highlight the related flow product name

FLUID BLOCK NAME EXAMPLE	STATUS	NOTE
fluidExit_UNGL	OK	UNGL is <i>unstable natural gas liquid</i> . This product can be found in the built-in database. The separation symbol "_" is correct
fluidExit1_UNGL, fluidExit2_UNGL	OK	UNGL is <i>unstable natural gas liquid</i> . This product can be found in build in database. The separation symbol "_" is correct. <i>FluidExit</i> names differ in the first part
fluidExit1_UNGL, fluidExit1_WMW	OK	UNGL is <i>unstable natural gas liquid</i> and WMW is <i>water-methanol mixture</i> . These products can be found in build-in database. The separation symbol "_" is correct
fluidExit_OGCM	Wrong	OGCM is <i>oil and gas condensate mixture</i> . However, if OGCM can't be found in the built-in database, PRL shows a mistake
fluidExit	Wrong	Separation symbol "_" is not found. There is no product with the name <i>fluidExit</i> in the built-in database. PRL will try to search missing <i>fluidExit</i> product in database
fluidExit1_UNGL, fluidExit1_UNGL	Wrong	There are two identical FluidExit names. Anylogic will not allow you to create them 😊
fluidExit!UNGL	Wrong	Separation symbol "_" is incorrect and PRL will try to search missing <i>fluidExit!UNGL</i> product in the built-in database
fluidExitUNGL	Wrong	Separation symbol "-" is not found and PRL will try to search missing <i>fluidExitUNGL</i> product in the built-in database
fluidExit_ZZZ	Wrong	What is <b>ZZZ</b> ? If this "product" can't be found in build in database, PRL shows a mistake. But separation symbol "_" is correct



Our practice shows that it is most convenient to name connections as *fl\_ExitAgent\_EnterAgent\_ProductName* (for example: *fl\_PlantDK\_RpKerosine\_Kerosine*)

6 (optional) Update the built-in AnyLogic fluid library solver. After each event in your model, AnyLogic will automatically call an external Apache LP solver to solve the LP problem of maximum graph flow. Extensive use of this library has demonstrated the *instability of the default AnyLogic settings* for the fluid library, particularly when solving problems for large systems, such as Oil & Gaz. We recommend to update the built-in AnyLogic solver for improved productivity and to reduce error messages like "*No Feasible Solution*". Please contact [support@prlsim.com](mailto:support@prlsim.com) for more information and to get a corrected solver

## 5 USING PETROLEUM REFINING LIBRARY

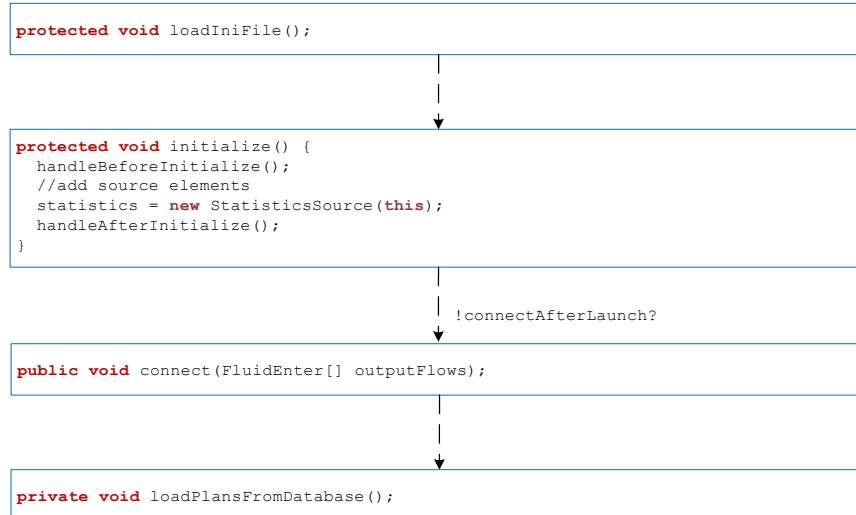
### 5.1 Source

**Source** is an advanced container that allows to create various types of entry flows (oil, unstable natural gas liquid, etc.). *Source* agents enable to set daily or monthly flow plans from the attached database, facilitate flow smoothing for monthly values, and provide many other functions.

Each *Source* agent contains arrays of *SourceElement* agents. *SourceElement* agent can generate an oil or condensate flow in accordance with a given plan. The execution of the plan is carried out by the *SourceElement* automatically, which calculates the instantaneous pumping rate at the time the plan was set.

Each *Source* agent contains an internal class *Statistics* with accessible via *public* variable *statistics*. This class contains many methods for collecting and analyzing data from the *Source* agent. Look PRL API for more information.

## KEY STEPS OF INITIALIZATION



## HITS AND TIPS

1 The size of the *exit[]* array must be equal to *SourceElement.size()* or equal to *one*. In the latter case, all exit flows will be automatically mixed into one

2 All *FluidEnter* blocks, associated with the *Source* agent, must have a naming structure that allows identification of flow product types

3 The *onSpeedChange()* event is triggered with an atomic delay. This is necessary to ensure that all speed changes been completed before the event is called

4 Processing plants are usually connected to groups of sources, rather than just one. It is rare for each individual source to have its own dedicated pipeline leading to the plant. Typically, groups of sources mix their flows before they enter the processing plants. In this scenario, the result flow that has already been mixed comes to the processing plant. That's why all sources in the *Source* agent are not individually set up. If an individual flow reaches the processing plant it should be identified in the database by a unique group number

5 Each *Source* agent allows to set of two types of plans: *plan* and *plan2*. The *plan2* variable is auxiliary and depends on the value of the *plan* variable. For example, if the *plan* variable represents the mass of unstable natural gas liquid, then the *plan2* variable may determine the equivalent volume of gas production or the value of the gas condensate factor

6 PRL will automatically create statistics tables in the AnyLogic database for each *SourceElement* agent, according to the following template: *PrlConstants.DB\_STATS\_PREFIX* + "source\_" + *sourceId*

7 To change agent constants, use the *build()* method in the inner Constant class



// Change the Interval (in hours) between successive smoothing updates

```
source.constant = source.new Constant().builder().SMOOTHING_DURATION (20).build();
```









## 5.2 Plant

**Plant** is the major agent that allows the configuration of flow-splitting processes by various types of oil and gas processing plants (such as rectification, catalytic reforming and others). The *Plant agent* contains a variety of settings to establish basic operating conditions for any type of oil and gas plant.

## KEY STEPS OF INITIALIZATION



## ICONS

ICON	INDEX
	0
	1
	2
	3
	4
	5
	6
	7

### 5.3 Control

**Control** is an agent that collects all requests from [RpAccumulative](#) and [ShipmentNode](#) agents and tracks their completion level. Control agent allows to make an effective request management because it contains JAVA Interface elements that enable setting different rules for processing requests.



Each model can contain no more than one Control agent!

Control agent contains two type of views:

- requests for products pumping to fulfill month plans, received from *ShippingNode agents*;
- requests to fulfill production pumping plans, received *RpAccumulative* from *ShipmentNode* (priority **A**), as well as additional tank loading, due to large pumping plans for the next month (priority **B**), additional tank loading to the minimum level (priority **C**), as well as additional mass for entering the fleet by user's request (priority **D**).

Product shipment requests (priority **A**) received from *ShippingNode* and *RpAccumulative* agent are not identical. *ShippingNode* agent generates a "direct" request to shipping a certain amount (mass) of product, whereas *RpAccumulate* takes into account the current tanks product residue, losses, and additions.

## 5.4 SeparatorLight

**SeparatorLight** is an advanced separator (split) agent that allows to split the input flow into two outputs. In comparison with the standard Anylogic FluidSplit, this agent enables to direct the entire flow to only ONE output, set the input flow speed limit, and implement many other functions

### KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();
```

```
handleAfterInitialize();
```

### HITS AND TIPS

1 All split and merge functions are split (merge) flows by setting a new fraction value only on one exit (enter) *SeparatorElement* (*MixerElement*) agent. The second exit (enter) automatically equates to 1.0. This significantly improves performance, as it requires only a single recalculation of the internal AnyLogic max flow LP problem

2 If the *useIniSettings* parameter is true, then agent will try to load *fraction1* and *speedLimit* parameters from \*.ini file to use them as start values

3 When specifying the separation (mixing) factors, normalization is not needed. It will be performed automatically

## 5.5 MixerLight

**MixerLight** is an advanced mixer (merge) agent that allows to combine two entry flows into one exit flow. In comparison with the standard Anylogic FluidMerge, this agent enables to setting an exit flow speed limit, and implement many other functions

### KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();
```

```
handleAfterInitialize();
```

### HITS AND TIPS

1 All split and merge functions are split (merge) flows by setting a new fraction value only on one exit (enter) *SeparatorElement* (*MixerElement*) agent. The second exit (enter) automatically equates to 1.0. This significantly improves performance, as it requires only a single recalculation of the internal AnyLogic max flow LP problem

2 If the *useIniSettings* parameter is true, then agent will try to load *fraction1* and *speedLimit* parameters from \*.ini file to use them as start values

3 When specifying the separation (mixing) factors, normalization is not needed. It will be performed automatically

## 5.6 Separator

**Separator** is an advanced separator (split) agent that allows for splitting an incoming flow into any number of exits flows. In comparison with the standard Anylogic FluidSplit, this agent allows for directing the entire flow to only ONE output, setting as many exits as necessary, establishing a user-defined separation factor, entering a speed limit, and implement many other functions



*The Separator (Mixer) agent does not require an unique flow product names at the exit (enter) of the agent. If more than one exit (enter) flow has the same product name, they will be added to the Separator (Mixer) in the order in which they are found in the incoming (out coming) flows array*

### KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();

!connectAfterLaunch

public void initialize(inputFlow, outputFlows) {
    statistics = new StatisticsSeparator(this)
    handleAfterInitialize();
}
```

### HITS AND TIPS

- 1 All split and merge functions are split (merge) flows by setting a new fraction value only on one exit (enter). The second exit (enter) automatically equates to 1.0. This significantly improves performance, as it requires only a single recalculation of the internal AnyLogic LP max flow problem
- 2 When specifying the separation (mixing) factors, normalization is not needed. It will be performed automatically

## 5.7 Mixer

**Mixer** is an advanced mixing (merging) agent that allows for merging of any number of input flows into one output. In comparison with the standard Anylogic FluidMerge, this agent allows users to set as many input flows as desired, establishes an input speed limit, and implement many other functions



*The Separator (Mixer) agent does not require an unique flow product names at the exit (enter) of the agent. If more than one exit (enter) flow has the same product name, they will be added to the Separator (Mixer) in the order in which they are found in the incoming (out coming) flows array*

### KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();

!connectAfterLaunch

public void initialize(inputFlow, outputFlows) {
    statistics = new StatisticsMixer(this)
    handleAfterInitialize();
}
```

### HITS AND TIPS

- 1 All split and merge functions are split (merge) flows by setting a new fraction value only on one exit (enter). The second exit (enter) automatically equates to 1.0. This significantly improves performance, as it requires only a single recalculation of the internal AnyLogic max flow LP problem
- 2 When specifying the separation (mixing) factors, normalization is not needed. It will be performed automatically

## 5.8 ShipmentNode

**ShipmentNode** is an agent that allows to set plans for pumping products and monitor the degree of their implementation. This agent contains a variety of settings that allow to create different options for pumping products (daily, monthly)



Each *ShipmentNode* agent has a strict rule: one *ShipmentNode* agent for one product

## KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();
```

```
protected void initialize() {  
    mixer.initialize(inputFlows, fEnter);  
    statistics = new StatisticsShipmentNode(this);  
    handleAfterInitialize();  
}
```

```
eLoadPlans.restart(); //Load plans from the DataBase
```

## 5.9 ReservoirPark

**ReservoirPark** is an agent, that allows to configure the basic options for ReservoirPark. This agent contains a set of basic properties and methods that are used in [RpAccumulative](#) and [RpFlowing](#) agents.

## KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();
```

```
protected void initialize() {  
    statistics = new StatisticsReservoirPark(this, false);  
    statisticsFlowStructure = new StatisticsFlowStructure(this);  
}
```









```
handleAfterInitialize();
```

```
//Update enter speed mix recipe  
protected void applyInputFlowMixing();
```

```
protected void loadTanksFromDatabase();
```

```
// Create events for repair  
protected void scheduleTankRepairs();
```

## ICONS

ICON	INDEX
	0
	1
	2
	3
	4
	5
	6
	7

### 5.9.1 RpFlowing

**RpFlowing** is an advanced park agent, that extends all ReservoirPark agent properties. RpFlowing is characterized by continuous inflows and outflows that fill and unload parallel connected tank (called "maps"), RpFlowing is used for "smoothing" outgoing flow products.

RpFlowing can operate in two different "smoothing" modes, depending on the *autoAdjustOutflowSpeed* value.

**Mode 1:** The out speed ( $V_{out}(t)$ ) at each moment is determined automatically by the out speed at the previous moment ( $V_{out}(t-1)$ ) and the maximum out speed change per step ( $\Delta V_{out}$ ). It also takes into account restrictions on the maximum ( $V_{max}$ ) and minimum ( $V_{min}$ ) values of out speed. This mode is usually used when RpFlowing is set at the end of the scheme, and the outflow goes to a pipeline. In this case, the minimum and maximum pumping speeds are determined by the performance of the pumping equipment, and the maximum speed change per simulation step is dictated by the need for the tanks to "breathe," as well as the inadmissibility of excessive hydrodynamic effects on the receiving gas pipeline.

**Mode 2:** RpFlowing is located within the scheme and also performs the function of "smoothing" the outgoing flow. However, in this case, the out speed flow is entirely determined by the restrictions imposed by subsequent nodes (for example, the total capacity of the plants located behind RpFlowing).

RpFlowing is characterized by the following state diagram (named *sStates*):

**sInitial** is the starting state. It is used for the time = 0 of establishing connections between all RpFlowing components.

**sDirectFlow** is a direct flow mode. In this mode, the entrance of the RpFlowing is directly connected to the exit. RpTank simulations are excluded from the calculation.

**sFlowCorrection** is a state in which the incoming flow is not equal to the shipment flow or the tanks loading exceeds the "acceptable" range.

**sSteadyFlow** is a state in which the incoming flow is equal to the outgoing (with a given accuracy) and the tank loading is in an "acceptable" range.

**sPumpingDown** is a state in which the incoming flow is zero and the model consistently reduces the pumping speed to a minimum (and then zero) value.

**sOverflowProtection** is a state of suspension of reception when the filling level of the tank farm is close to critical.

The states **sSteadyFlow**, **sPumpingDown** and **sOverflowProtection** are available if RpFlowing is working in Mode 1. All settings for RpFlowing states are set via the built-in Constant class.

### 5.9.2 RpAccumulative

**RpAccumulative** is advanced park agent, that extends all Reservoir Park agent properties. RpAccumulative contains many options, which are necessary for "cumulative" reservoir parks. This agent implements procedures of accumulation, passportization (compounding), and shipment of products. In addition, RpAccumulative contains built-in mechanisms for tracking product shipment plans



You can connect one *RpAccumulative* to one or more *ShipmentNode* agents. In the second case, *RpAccumulative* will strive to consistently meet the shipping requirements of each of the *ShipmentNode* agents connected to it

#### HITS AND TIPS

1 This agent allows the creation of special requests for Pants to add additional mass to the Reservoir Park. Each request is characterized by two parameters: priority and amount. All requests are collected in the Control agent to manage their execution

2 If requests with priority **A** (*aPriorityShipmentPlan*) are included, then it is necessary to add link to one or more *ShipmentNodes* agents. Each *ShipmentNode* agent pump product to fulfill the plan and passes it to *RpAccumulative*. In turn, *RpAccumulative* forms its own request based on the *ShipmentNode*. However, in general, the masses of *ShipmentNode* and *RpAccumulative* queries do not match. This is due to the presence of residues in *RpAccumulative* tanks, losses, and the use of additional additives

3 *RpAccumulative* agent also contains *ReservoirReallocator* Class that allow to transfer tanks to or from "reserve". The concept of "tanks reserve" is used to account for the possible movement of reservoirs between different parks, when the same tank can be used in different parks during the simulation.

## 5.10 Optimizer

**Optimizer** is an agent, that allows to set up and solve different kinds of linear programming (LP) problems (for example, splitting the input flow between plants and plant lines with the objective of minimizing residuals). The Optimizer agent can solve two types of LP problems.

The first type is "parametric", where user has a ready-made [lpSolve](#) file (\*.lp) with specific parameters. In this case, the Optimizer will automatically replace the parametric texts in the \*.lp file with numeric values, solve the LP problem, and save it with a new name. This type of task is useful for complex LP problems when the user needs to see all the equations directly, but it is quite slow.

The second type allows the user to create own LP problem equations and solve it directly, by using the internal Optimizer agent via Java ILP Interface.

### HITS AND TIPS

1 In the case of parametric calculation, the Optimizer agent will try to find the source LP file in the folder `System.getProperty("user.dir") + "\\LP\\" + lpName + ".lp"`.

2 The optimizer agent contains public Java ILP variables, which help to formulate any LP problem equations without creating an external LP file:

```
/**Class SolverFactoryLpSolve of Java ILP*/  
public SolverFactoryLpSolve lpFactory = new SolverFactoryLpSolve();  
  
/**Class Result of Java ILP*/  
public Result lpResult;  
  
/**Class Problem of Java ILP*/  
public Problem lpProblem;
```

3 Please remember that parametric optimization is **quite slow**!

## 5.11 GasOwnNeeds

**GasOwnNeeds** is an agent, that allows to calculate the amount of gas for own needs (GON) and distribute it among Plants. As usual, the GON flow depends on the season, list of Plants, their modes and load levels

### KEY STEPS OF INITIALIZATION

```
public void initialize() {  
    onAfterInitialize();  
}
```

## 5.12 Dispose




**Dispose** is an agent, that allows to calculate the amount of disposed flows. In comparison with the standard Anylogic FluidDispose, this agent enables the collection of the same type of flows (for example, gas flows per flare) into one agent, saves the necessary statistics, and destroys flows

### KEY STEPS OF INITIALIZATION

```
statistics = mixer.statistics; //dispose statistics is a mixer statistics type
```

```
protected void initialize() {  
    mixer.initialize(inputFlows, fEnter);  
    handleAfterInitialize();  
}
```



ICON	INDEX
	0
	1
	2

### 5.13 FlowQuota

**FlowQuota** is an agent that allow to set periodic plans for pumping products. For example, set a plan for the maximum monthly (or daily) pumping. Required to account for the maximum capacity of the pumping equipment. In compare to the *dispose()* method available in the Fluid library *valve*, this agent also allows to make the constraint periodic (i.e. daily or monthly)

#### KEY STEPS OF INITIALIZATION

```
protected void loadIniFile();
```



```
handleAfterInitialize();
```

#### HITS AND TIPS

- 1 The default value of *quotaValue()*, is loaded from the \*.ini file into public temporary *parsedQuotaLimit* variable

### 5.14 ProductMixer

**ProductMixer** is an advanced product mixing agent, that allows for the optimal mixing of input flows and product additives to create a desired product (such as petrol, kerosene, etc.). It is known that the creation of most products is accompanied by a number of requirements (such as octane number, sulfur content, saturated vapor pressure, aromatics, etc.). This agent takes all these requirements into account and mixes incoming flows in an optimal way (assuming the additive of the requirements) to ensure their simultaneous fulfillment



ProductMixer try's to ensure all requirements at the assumption of additive of their values

#### KEY STEPS OF INITIALIZATION

```
constant = new Constant();
```



```
lpFactory = new SolverFactoryLpSolve();  
lpFactory.setParameter(Solver.VERBOSE, 0);
```



```
statistics = new StatisticsProductMixer(this);
```



```
protected void loadIniFile();
```



```
Protected void initialize() {  
    handleAfterInitialize();  
}
```

#### HITS AND TIPS

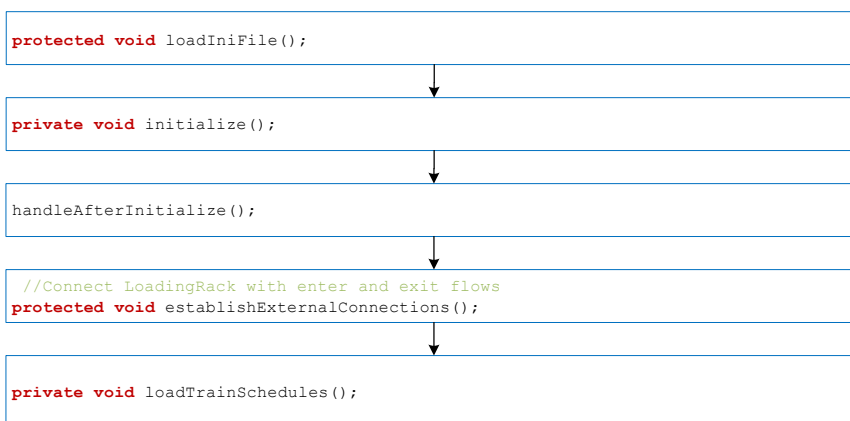
- 1 If the *isAutoCalculationEnabled* is true, then the ProductMixer agent makes an automatic formulation and solution of the LP problem. The objective function, which is the sum of the penalty functions with the corresponding coefficients, is used as the objective function

NAME	PENALTY	DESCRIPTION
<i>shouldMinimizeAdditive= true</i>		
FLOW_MASS_RESIDUAL_FIRST	1E6	Unrecognized remnants of first-priority flows
FLOW_PENALTY	1E3	Additive usage amount
FLOW_PENALTY_MIN	1	Unprocessed product balance
FLOW_MASS_RESIDUAL_SECOND	1	Unrecognized remnants of second-priority flows
<i>shouldMinimizeAdditive = false</i>		
FLOW_MASS_PRODUCT_TEXT	-1	Maximize the exit of result product

## 5.15 LoadingRack

**LoadingRack** controls the loading of refined products into railway tanks/trucks at a loading rack, managing train schedules, loading operations, wagon assignments, product quantities, supply/removal durations, passportization, and overseeing the state machine for the loading workflow (supply/load/remove)

### KEY STEPS OF INITIALIZATION



## 4 AUXILIARY DATABASE

A key part of PRL is an auxiliary database that contains all information about products, Reservoir Parks, Plants and other



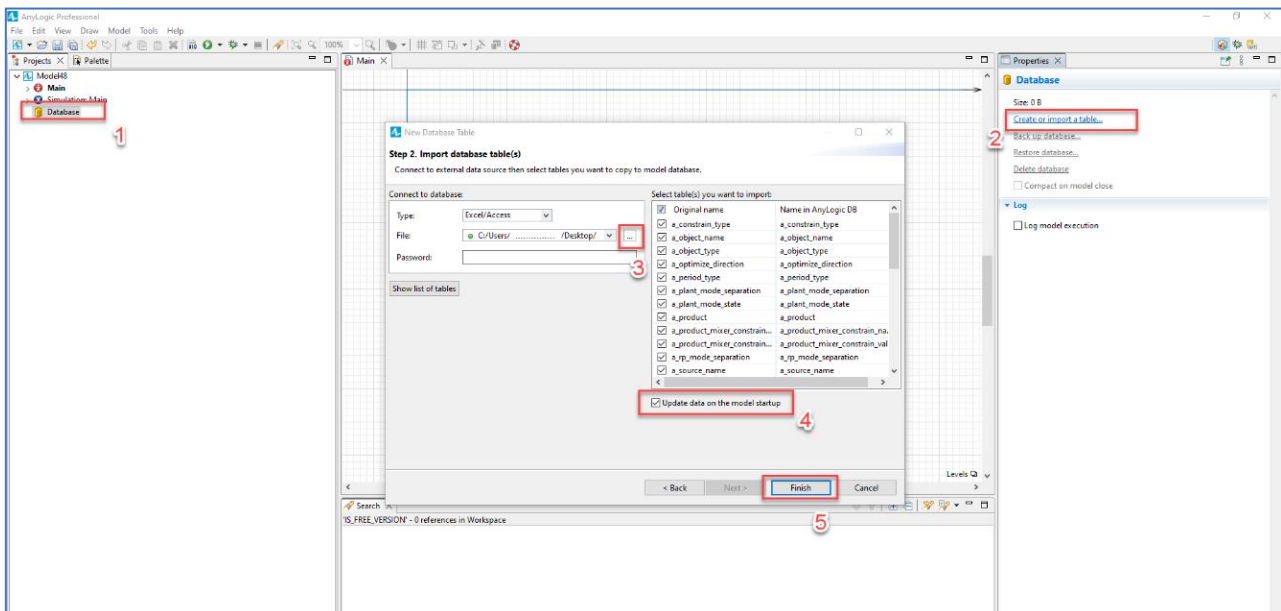
PRL automatically checks for database tables in the AnyLogic project every time it starts. If the specified tables are not found, PRL will return an error

To add an auxiliary database to your project:

1. In Anylogic please click on **Palettes -> Manage Libraries -> Add**. Add PRL to your project.
2. Add a PRL database to your model. You can find the **DB\_Refining.accdb** database in the PRL folder.



At present, PRL has a ready - made database templates in Microsoft Access



## 4.1 Database references

**a\_constrains\_types** – a reference, containing a list of basic units of measurement. Users can change the names

FIELD NAME	FIELD TYPE	DESCRIPTION
constrain_type_id (id)	Counter	-
name (Name)	Short text	-

**a\_loading\_racks\_states** – a reference, containing a list of basic stats of loading racks. Users can change the names

FIELD NAME	FIELD TYPE	DESCRIPTION
loading_racks_id (id)	Counter	-
name (Name)	Short text	-

**a\_object\_types** – a reference, containing a list of object types (Plant or Reservoir Park). Name field can be changed by the user

FIELD NAME	FIELD TYPE	DESCRIPTION
a_object_type (id)	Counter	-
name (Name)	Short text	-

**a\_optimize\_directions** – a reference containing a list of optimization limitations. The name field can be changed by the user. This table is used in the ProductMixer agent when creating and check the achievement of constraints

FIELD NAME	FIELD TYPE	DESCRIPTION
optimize_direction_id (id)	Counter	-
name (Name)	Short text	-

**a\_periods\_types** – a reference containing a list of planning durations. The name field can be changed by the user. This table is used in the Source agent when loading plans

FIELD NAME	FIELD TYPE	DESCRIPTION
period_type_id (id)	Counter	-
name (Name)	Short text	-

**a\_product\_mixer\_constrains\_names** – a reference, containing a list of limitations when ProductMixer agents attempts to create new types of products

FIELD NAME	FIELD TYPE	DESCRIPTION
constrain_name_id (id)	Counter	-
full_name (Full restriction name)	Short text	Full product name (optional)
short_name (Short name)	Short text	Short constrain name
unit (Unit)	Short text	Unit measure
description (Description)	Short text	-

**a\_product\_mixer\_constrains\_values** – a reference, containing a list of flow limitations values when ProductMixer agents tries to create new types of products

FIELD NAME	FIELD TYPE	DESCRIPTION
constrain_value_id (id)	Counter	-
constrain_name_id (Restriction name)	Number	Link to the restriction name
product_short_id (Product name)	Number	-
if_enabled (Enable)	boolean	If false then PRL will not find this record
limit_value (Limit)	Number	-
direction_id (Direction)	Number	-
type_id (Type)	Number	Type: mass or volume
density (Density)	Number	Density for moving to volume limits
group_number (Group number)	Number	Group of restrictions (for example for separation restrictions in winter and in summer)
description (Description)	Short text	-

**a\_seasons** – a reference containing a list of the seasons

FIELD NAME	FIELD TYPE	DESCRIPTION
season_type_id (id)	Counter	-
name (Name)	Short text	-

**a\_tank\_states** – a reference containing a list of the states of the ReservoirPark agents tanks

FIELD NAME	FIELD TYPE	DESCRIPTION
tank_state_id (id)	Counter	-
name (Name)	Short text	-

## 4.2 Database tables

**checkbox\_values** – table, containing a list of Plant agents checkbox

FIELD NAME	FIELD TYPE	DESCRIPTION
id (id)	Counter	-
object_id (Name)	Short text	-
label_name (Caption)	Short text	-
if_enabled (Enable)	boolean	-
description (Description)	Short text	-

**gas\_own\_needs** – table, containing settings to calc gas own needs

FIELD NAME	FIELD TYPE	DESCRIPTION
id (id)	Counter	-
plant_id (Name)	Number	-
gon_value (Value (ton(s)/tons(s) enter))	Number	-
if_enabled (Enable)	boolean	-
month_number (Month [1-12])	Number	-
description (Description)	Short text	-

**general** – table, containing the basic options of model

FIELD NAME	FIELD TYPE	DESCRIPTION
id (id)	Counter	-
program_id	Number	Id in the PRL library functions
description	Short text	-
val (value)	Number	-

**link\_values** – table, containing a list of links

FIELD NAME	FIELD TYPE	DESCRIPTION
link_id	Counter	-
object_id	Counter	-
if_enabled	boolean	-
parameter_value	Number	-
description	Short text	-

**listbox\_values** – table, containing a list of values to fill ListBox

FIELD NAME	FIELD TYPE	DESCRIPTION
object_list_id (id)	Counter	-
object_id (Name)	Number	-
parameter_value (Value)	Number	-
if_enabled (Enable)	boolean	-
description (Description)	Short text	-

**loading\_racks** – table, containing a list of the loading racks

FIELD NAME	FIELD TYPE	DESCRIPTION
rack_id (id)	Counter	-
rack_name (Name)	Short text	-
start_state (Start state)	Number	-
Initial_residue (Initial residue (ton(s)))	Number	-
If_enabled (Enable)	boolean	-
description (Description)	Short text	-

**a\_object\_name** – a reference containing a list of modeling objects (Plants, Reservoir Parks). Information about all objects must be entered in this table



All modeling objects (such as Plants or Reservoir Parks) have a *shortName* variable. PRL will automatically search for objects by their *shortName* in this table and load all object parameters from the database using *object\_id*

FIELD NAME	FIELD TYPE	DESCRIPTION
object_id (id)	Counter	-
short_name (Name)	Short text	This field is used when PRL tries to link the model with the database. Short names in this field MUST be equal to the shortNames field of the Plant and Reservoir Park agents.
full_name (Full name)	Short text	Optional describe field
if_enabled (Enable)	Boolean	If false then PRL will not find this record
type (Type)	Number	Type of object (Plant or Reservoir Park)
description (Description)	Short text	-

**plant\_capacity** – table, containing information about Plant agents lines capacity

FIELD NAME	FIELD TYPE	DESCRIPTION
mode_id (id)	Counter	-
plant_id (Name)	Number	-
mode_name (Mode name)	Short text	-
min_load (Min load (ton(s)/h))	Number	-
group_modes (Group modes)	Number	If the value in this field is the same for several modes, then the temporary mode will not be initiated
max_load (Max load (ton(s)/h))	Number	-
if_enabled (Enable)	boolean	-
if_mode_default (Default mode)	boolean	-
description (Description)	Short text	-

**plant\_lines** – table, containing information about Plant agents lines count

FIELD NAME	FIELD TYPE	DESCRIPTION
id (id)	Counter	-
plant_id (Name)	Number	-
If_enabled (Enable)	Short text	-
lines_count (Lines count (for plant))	Number	-
lines count (for plant)	Number	-

**plant\_modes\_cluster** - allows to set availability of Plant modes based on they relationships. For example, can be used to set the modes of Plant B witch be available if a specific mode of Plant A is selected

FIELD NAME	FIELD TYPE	DESCRIPTION
id	Counter	-
source_mode_id (Source mode)	Number	-
dependent_mode_id (Dependent mode)	Number	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-

**plant\_modes\_separation** – a reference contains Plant agents separation coefficients. These coefficients depend on the type of incoming product flow entering the Plant agent. More information about recipes can be found [here](#)

FIELD NAME	FIELD TYPE	DESCRIPTION
plant_modes_sep_id (id)	Counter	-
plant_id (Name)	Short text	Link to the Plant
mode_id (Mode name)	Number	Link to the Plant mode
product_id (Product name)	Number	Link to the Plant out product
in_flow_id (Enter product name)	Number	Link to the Plant in flow
separation_value (Separation value)	Number	Coefficients of separation
if_enabled (Enable)	Boolean	If false then PRL will not find this record
description (Description)	Short Text	-

**plant\_modes\_temporary** – a reference that contains the relationships between Plant agents states. It allows for the definition of temporary and permanent Plant modes. If a mode has a required temporary mode, then it must be obtained on one line

FIELD NAME	FIELD TYPE	DESCRIPTION
plant_mode_state_id (id)	Counter	-
plant_id (Name)	Number	Link to the Plant
temporary_id (Temporary mode name)	Number	Link to the Plant temporary mode
mode_id (Mode name)	Number	Link to the Plant permanent mode
if_enabled (Enable)	Number	If false then PRL will not find this record
temporary_duration (Temporary duration, hour(s))	Number	Temporary duration
description (Description)	Boolean	-

**plant\_repairs** – table, containing information about Plant agents repairs

FIELD NAME	FIELD TYPE	DESCRIPTION
plant_repair_id (id)	Counter	-
plant_id (Name)	Number	-
plant_line (Line)	Number	-
date_begin (Begin date)	Date and time	-
date_end (End date)	Date and time	-
if_enabled (Enable)	boolean	-
description (Description)	Short text	-

**product** – a reference containing a list of products



Field *short\_name* can't contain spaces or special separation character " \_ "

Field *short\_name* is case sensitive, that is, products names **Oil** and **oil** are interpret as different

FIELD NAME	FIELD TYPE	DESCRIPTION
product_id (id)	Number	-
short_name (Product name)	Short text	Short product (flow product) name
full_name (Full product name)	Short text	Full product name (optional)
hex_color (Product color (HEX))	Short text	Product fluid color
if_enabled (Enable)	Boolean	If false then PRL will not find this record
density (Density (g/cm3))	Number	Density for using in ProductMixer agent and check the volume restrictions
description (Description)	Short text	-

**product\_mixer\_flow\_constrains\_values** – table, containing information about the values of the limiting parameters of the flows, that are used in the ProductMixer agent

FIELD NAME	FIELD TYPE	DESCRIPTION
flow_constrain_value_id (id)	Counter	-
product_id (Product name)	Number	-
constrain_name_id (Constrain name)	Number	-
flow_constrain_value (Value)	Number	-

**rp\_accumulatives** – table, containing information about the structure of RpAccumulative agent



FIELD NAME	FIELD TYPE	DESCRIPTION
rp_accumulative_id (id)	Counter	-
rp_id (Name)	Number	-
tank_name (Tank name)	Short text	-
min_load (Min load (ton(s)))	Number	-
max_load (Max load (ton(s)))	Number	-
capacity (Capacity (ton(s)))	Number	-
residual (Capacity (ton(s)))	Number	-
condition (State)	Short text	-
if_enabled (Enable)	boolean	-
description (Description)	Short text	-

**rp\_flowling** – table, containing information about the structure of RpFlowling agent

FIELD NAME	FIELD TYPE	DESCRIPTION
rp_flowling_id (id)	Counter	-
rp_id (Name)	Number	-
tank_name (Map name)	Short text	-
min_load (Min map load (ton(s)))	Number	-
max_load (Max map load (ton(s)))	Number	-
capacity (Map capacity (ton(s)))	Number	-
residual (Residual (ton(s)))	Number	-
tank_count (Maps count)	Number	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-

**rp\_modes\_separation** – a reference containing a list of the main coefficients of separation in Reservoir Parks

FIELD NAME	FIELD TYPE	DESCRIPTION
rp_mode_separation_id (id)	Counter	-
rp_id (Name)	Number	Link to the Reservoir Park
product_id (Product name)	Number	Link to the Reservoir Park out product
separation_value (Separation value)	Number	Coefficients of separation
if_enabled (Enable)	boolean	If false then PRL will not find this record
description (Description)	Short text	-

**rp\_tanks\_repairs** – table, containing the repairs of tanks

FIELD NAME	FIELD TYPE	DESCRIPTION
rp_tank_repair_id (id)	Counter	-
rp_id (Name)	Number	-
date_begin (Date start)	Date time	-
date_end (Date end)	Date time	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-

**shipment\_exception** – table, containing a list of product shipment exclusion dates for ShipmentNode

FIELD NAME	FIELD TYPE	DESCRIPTION
rp_unloading_exception_id (id)	Counter	-
day_value (Day)	Number	-
month_value (Month)	Number	-
year_value (Year (optional))	Number	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-

**shipment\_plans** – table, containing product shipment plans

FIELD NAME	FIELD TYPE	DESCRIPTION
unload_production_plan_id (id)	Counter	-
product_id (Product name)	Number	-
plan_date (Period)	Date time	-
plan_value (Value (ton(s)))	Number	-
if_enabled (Enable)	Boolean	-
plan_period_type (Period type)	Number	-
description (Description)	Short text	-

**source\_names** – a reference containing a list of the Source agents

FIELD NAME	FIELD TYPE	DESCRIPTION
source_id (id)	Counter	-
name (Source name)	Short text	Source name
name2 (Source name 2)	Short text	Second Source name
product_id (Product name)	Number	Link to the Source out product
group_number (Group number)	Number	To combine sources whose flows are mixed before entering the Plant
if_enabled (Enable)	Boolean	If false then PRL will not find this record
description (Description)	Short text	-

**source\_plans** – table, containing a list of plans for incoming flows from Source agents

FIELD NAME	FIELD TYPE	DESCRIPTION
id (Id)	Counter	-
source_id (Name)	Number	-
period_start (Period)	Date time	-
primary_value (Period)	Number	-
if_enabled ()	Boolean	-
type (Period type)	Short text	-
secondary_plan (Secondary plan (ton(s)))	Number	-
description (Description)	Short text	-

**textbox\_values** – table, containing a list of text values

FIELD NAME	FIELD TYPE	DESCRIPTION
id (id)	Counter	-
object_id (Object name)	Number	-
label_name (Value)	Date time	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-

**train\_loading\_racks\_preparation**– table, containing a list of time to loading racks

FIELD NAME	FIELD TYPE	DESCRIPTION
rail_tanker_id	Counter	-
rail_tanker_id (Train name)	Number	-
if_enabled (Enable)	Boolean	-
description (Description)	Short text	-
Duration (Duration (h))	Number	-
State (State name)	Number	-

**train\_schedule** – table, containing a train arrival schedule

FIELD NAME	FIELD TYPE	DESCRIPTION
train_schedule_id (ID)	Counter	-
train_name (Train Name)	Number	-
arrival_date_time (Arrival date, time)	Date and time	-
rack_id (Loading rack)	Number	-
if_enabled (Enable)	Boolean	-
Description (Description)	Short text	-
expected_loading_time (Loading time optional (h))	Number	Does not participate in calculations. Contains the desired loading time of the train. Can be used to set, for example, the maximum pumping speed from a tank farm

**train\_wagon\_assignment** – table, containing a train wagon assignment

FIELD NAME	FIELD TYPE	DESCRIPTION
train_wagon_id (ID)	Counter	-
train_schedule_id (Train name)	Number	-
wagon_capacity (Wagon capacity (ton(s)))	Number	-
wagons_count (Wagons count)	Number	-
wagons_product_id (Wagon product)	Number	-
if_enabled (Enable)	Boolean	-
description (Description)	Short Text	-

## 5 CREATING AND EXECUTING THE PRODUCTS PRODUCTION “REQUESTS”

It is known that most of processing plants operate according to the same scheme: products that are produced at the Plants pass through Reservoir Parks before shipment. Every Reservoir Park perform accumulation, compounding (in some cases passportization) and product shipment. In this case, if you have an optimization shipment plan for all products (based on optimization planning tools such as Aspen Pims), then you can set an equivalent shipment plan of product for each of Reservoir Park.

The difficulty lies in that fact that Reservoir Parks of most processing Oil & Gas plants operate according to more complex scheme. Personnel often performs to make an additional operations for filling or unloading tanks. For example, set a large plan for the product shipment at next month. In this case it is necessary to fill tanks with additionally mass to create starting reserves to fulfill plans for the next month. Such actions can be effectively presented in the form of a demand (request) from Reservoir Parks to the Plans. Each request is characterized by required product, mass, it's priority (more on this below) and possibly other characteristics.

*RpAccumulative* agent contains **four** built-in types of product requests (requests are listed in order of decreasing their priority)<sup>1</sup>

REQUEAST TYPE	PRIORITY	VARIABLE DESCRIPTION (NAME)	DESCRIPTION
Shipment plan	<b>A</b>	A: Fulfillment shipment plan (aPriorityPlan)	Reflects most important requirement from the Reservoir Park to Plants: additional product mass to complete shipment plan at current month
Additional amount to complete the plans for the next month	<b>B</b>	B: Additional load to next month (bPriorityNextMonth) min plan to next month (tons) (bPriorityNextMonthLimit) additional mass (tons) (bPriorityNextMonthAddMass)	Reflects requirement for additional mass to fulfill a plan for the shipment of products in next month Depends on two parameters: bPriorityNextMonthLimit - minimum plan for shipment for the next month for creating a request bPriorityNextMonthAddMass - requirement for additional mass in Reservoir Park
Additional amount to loading an active tank to the minimum acceptable level	<b>C</b>	C: Active tank additional load min tank load to start proceed (tons) cPriorityLevelStartProceed	Reflects requirement for additional mass to loading an active tank to the minimum acceptable level. This level can be determined, for example, by the minimum level of the start of passportization
Additional user mass entrance	<b>D</b>	D: Additional user mass entrance dPriorityUserAmount	Reflects requirement for additional mass for some reason specified by user

Each *RpAccumulative* consistently generates requests to complete the four types of requests listed above. Note that the product shipment request (Shipment plan, priority A) is set to *RpAccumulative* agent by *ShipmentNode* agent. *ShipmentNode* agent allows to automatically load and monitor the implementation of plans for the shipment of any product. This allows to build more complex schemes for shipping products. In particular, "many-to-many" relationships can be used, when products from several *RpAccumulative* agents can be used to fulfill any plan, and, conversely, several types of products can be used to fulfill any one plan (for example, propane's different brands can be used to execute the propane-butane plan). If there was only one Reservoir Park, then all requests would be fulfilled in order of decreasing priority.

<sup>1</sup> In some cases, the unloading from the Reservoir park may slightly exceed the established shipment plan. This happens if between the Reservoir park and the shipment node there is a LoadingPack, on the railway track of which there is a train with a total capacity of wagons exceeding the established pumping plan

The task becomes much more complicated if alternative requests are received by Plants from different fleets “competing” for their limited capacity. In this case, many factors have to be taken into account, such as the level of current competition of plans, priority of requests, Reservoir Park load levels, minimizing switching between modes, and others. Control agent features can be used to solve such complex tasks

## 6 PETROLEUM REFINING LIBRARY ENUMS

**MeasurementType** – types of PRL measuring

**ReservoirParkRequest** – ReservoirPark agent requests types

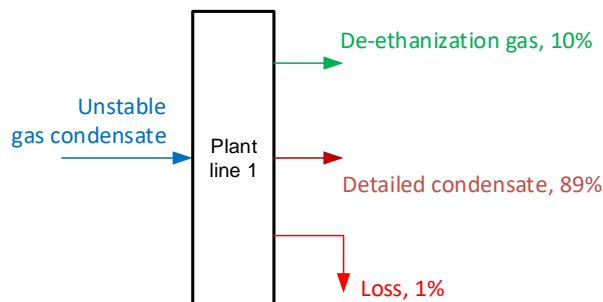
**ReservoirParkState** – ReservoirPark agent states

**ShipmentAlgorithm** – shipment algorithms

**RpTankState** – RpTank states

## 7 RECIPES

Recipes are the rules that relates the mass of the incoming flow to the outgoing flows of semi-products and products, as well as the flow of losses. In PRL recipes can be set for both in *Plant* and *ReservoirPark* agents. The recipes affect the output flows values and represent the separation coefficients. The example below shows how a given "recipe" for the process of deethanization of unstable gas condensate (10%, 89%, 1%) results in a simulation of separation on one of the Plant lines.



The recipe values for the PRL are the source data. To get recipe values, use specialized programs such as Aspen Hysys, “test runs”, or expert values

For every Plant agent, recipes are set in the plant\_modes\_separations database table. Each recipe depends on the mode of the Plant, as well as the type of incoming flow. The last condition allows to calculate the weighted average values of (mass) recipe values when more than one flows are sent to the Plant agent at the same time. In ReservoirPark, recipes are set in the rp\_modes\_separation table. Recipes for ReservoirPark agents reflect the possibility of separating the mixture entering the tank as a result of "sludge" or other mechanisms.



Plant and Reservoir Park agents recipes must have a loss flow. Otherwise, the error occurs

## 8 TYPES OF MEASURING

The PRL library allows to represent simulation model results in various units of measurement (for example, mass in tons(s), speed (tons(s)/h). The active model measurement unit is contained in *public static measType* variable. In addition, the PRL library allows the to change the active model measurement unit via the *public toggleMeasurementUnit()* method.

## 9 FAQ

1 How do I name FluidExit and FluidEnter block to highlight the product name?

**Answer:** every FluidExit and FluidEnter Anylogic block should allow PRL to identify a linked product flow name.

2 Are there any restrictions on the set date and time when the simulation model starts?

**Answer:** PRL does not contain any restrictions on the start date of the simulation. The initial simulation time should correspond to the beginning of the day (00:00:00)

3 What is the difference between the results obtained on PRL and Aspen Pims (or analogs)?

**Answer:** These systems solve two different problems. Aspen Pims allows to determine the optimal production plan that ensures, for example, the maximum total margin income. Meanwhile, it is known that the possibilities of analytical (in t.h. linear) mathematical optimization have quite serious limitations. In short, this is due to the inability to take into account all the features of processing plants in linear (and non-linear) equation systems. On the contrary, PRL allows you to take into account all key features as much as possible (and therefore check the optimal plans used for feasibility) to build a balance that is as close to reality as possible. Thus, these systems are not opposed, but complement each other.