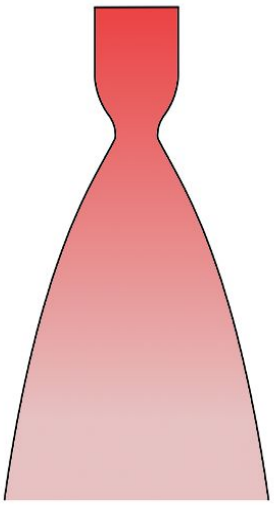
Rocket Propulsion Analysis



Version 2.2

User Manual

Cologne, Germany – 2014
www.propulsion-analysis.com

Table of Contents

Introduction....6
RPA editions....6
System requirements....7
Microsoft Windows....7
Apple Mac....7
Linux....7
Installation on Microsoft Windows....7
If you downloaded installation package....7
If you downloaded ZIP package....8
Installation on Apple Mac OS X....8
Installation on Linux....8
Running RPA....8
Graphical User Interface....8
Command-line utility....10
Scripting utility....11
Configuration Files....12
Engine Definition....12
Propellant Specification....14
Propellant Specification Screen....14
Component Properties....17
Components Database....18
Nozzle Flow Model....19
Nozzle Flow Model Screen....19
Nozzle conditions....20
Nozzle Shape and Efficiencies....21
Ambient condition....22
Throttle settings....22
Starting Analysis....23
Chamber Performance....24
Propellant properties....24
Thermodynamic properties....25
Performance....26
Altitude performance....28
Throttled performance....29
Nested Analysis....31
Propellant Analysis....32
Chamber Geometry....34
Design Parameters....34
Size and Geometry....35
Thermal Analysis....36
Heat Transfer Parameters....37
Thrust Chamber Cooling....38
Radiation Cooling....40
Regenerative Cooling....40

Coolant definition....41
Coaxial Shell Jacket Design....41
Tubular Wall Jacket Design....42
Channel Wall Jacket Design....43
Film Cooling....45
Thrust chamber with several cooling segments....45
Thermal Barrier Coating Layer....45
Thermal Analysis....46
Propellant Feed System (Cycle Analysis)....48
Design Parameters....48
Running Cycle Analysis....53
Operating Parameters....53
Cycle Performance....54
Engine Dry Weight....54
Thermodynamic Database Editor....55
Preferences....58
Input and Output Units....60
Scripting Utility....61
API Reference....61
Generic built-In Functions....62
Object File....64
Configuration API....65
Object ConfigFile....65
Object GeneralOptions....68
Object NozzleFlowOptions....70
Object CombustionChamberConditions....72
Object FreezingConditions....72
Object NozzleInletConditions....72
Object NozzleSectionConditions....73
Object EfficiencyFactors....74
Object AmbientConditions....75
Object ThrottlingConditions....76
Object Propellant....76
Object Component....77
Object EngineSize....77
Object ChamberGeometry....78
Thermo API....79
Object database....79
Object Species....80
Object Propellant....82
Object Mixture....84
Reaction API....86
Object Product....86
Object Reaction....89
Object Derivatives....91
Performance API....93

Object Chamber....94
Object NozzleSectionConditions....95
Object ChamberFr....97
Object NozzleSectionConditionsFr....99
Object Performance....100
Scripting examples....101
Performance - Example 1....101
Performance - Example 2....102
Performance - Example 3....103
Performance - Example 4....104
Performance - Example 5....106
Propellant....107
Mixture....108
Reaction....110
Reaction Products....111
Frozen Equilibrium....113
Nested Analysis....114
Propellant Analysis....115

Introduction

RPA is an acronym for Rocket Propulsion Analysis.

RPA is a rocket engine analysis tool for rocketry professionals, scientists, students and amateurs.

RPA is an easy-to-use multi-platform tool for the performance prediction of rocket engines. It features an intuitive graphical user interface with convenient grouping the input parameters and analysis results. RPA utilizes an expandable chemical species library based on NASA Glenn thermodynamic database, that includes data for numerous fuels and oxidizers, such as liquid hydrogen and oxygen, kerosene, hydrogen peroxide, MMH, and many others. With embedded species editor, the users may also easily define new propellant components, or import components from *PROPEP* or *CEA2* species databases.

By providing a few engine parameters such as combustion chamber pressure, used propellant components, and nozzle parameters, the program obtains chemical equilibrium composition of combustion products, determines its thermodynamic properties, and predicts the theoretical rocket performance. The results of calculation can also be used to design combustion chambers, gas generators and preburners of the liquid propellant rocket engines.

The calculation method is based on robust, proven and industry-accepted Gibbs free energy minimization approach to obtain the combustion composition, analysis of nozzle flows with shifting and frozen chemical equilibrium, and calculation of engine performance for a finite- and infinite-area combustion chambers.

RPA is written in C++ programming language using following libraries: Nokia Qt, Qwt, libconfig++.

The program was written by Alexander Ponomarenko. You can contact him by sending an email to: contact@propulsion-analysis.com

RPA editions

You can download three different versions of RPA from http://www.propulsion-analysis.com/downloads.htm: freeware Lite Edition, commercial Standard Edition v.1.x and commercial Standard Edition v.2.x.

System requirements and installation procedure are the same for both editions.

If you downloaded and used an evaluation copy of RPA Standard Edition with free 15-day trial period, you may purchase a license and get your personalized product key which converts this evaluation version of the product to a fully licensed version.

System requirements

Microsoft Windows

Operating Systems:

- Windows 2000 (32-bit or 64-bit Edition)
- Windows XP (32-bit or 64-bit Edition)
- Windows Vista (32-bit or 64-bit Edition)
- Windows 7 (32-bit or 64-bit Edition)

Any computer that runs with mentioned operating systems.

Apple Mac

- Mac OS X 10.5 or later
- Macintosh computer with an Intel x86 or x86-64 processor

Linux

RPA will not run without the following libraries:

- Glib 2.12 or higher
- X.Org 1.0 or higher

Installation on Microsoft Windows

RPA for Microsoft Windows is distributed in installation and ZIP packages both for x86 and x86-64 architectures.

RPA for Windows depends on Nokia Qt and MS VC++ 2008 SP1 run-time libraries. If your computer does not have it installed, please choose the package that includes all required components.

If you downloaded installation package

- Run installation executable file and follow the instructions the installer provides
- When done with the installation, you can delete the installer file to recover disk space
- The installer will create shortcuts for RPA executable on desktop and start menu, which you can use to start the application
- To uninstall the application run `Uninstall.exe` from the RPA installation directory

Note that in order to install the software from installation executable file you must have administrator rights. If you don't have administrative rights, you can still install the program from ZIP package.

If you downloaded ZIP package

- Extract files from the ZIP package into selected directory
- Start the program, executing the command `RPA.exe`
- To uninstall the application delete the RPA installation directory

Installation on Apple Mac OS X

RPA for Apple Mac OS X is distributed in installation package, containing universal binary for Intel x86 and x86-64 architectures.

RPA for Apple Mac OS X depends on Nokia Qt run-time libraries. If your computer does not have it installed, please choose the package that includes all required components.

To install the program

- Run installation package and follow the instructions the installer provides
- When done with the installation, you can delete the installer file to recover disk space
- The program `RPA.app` will be installed in Application directory
- To uninstall the application delete `RPA.app` from the Application directory

Installation on Linux

RPA for Linux is distributed in tar.gz packages both for x86 and x86-64 architectures.

RPA for Linux depends on Nokia Qt libraries. If your computer does not have it installed, please choose the package that includes all required components.

To install the program

- Extract files from the archived package into selected directory
- Start the program, executing the shell script `RPA.exe`
- To uninstall the application delete the RPA installation directory

Running RPA

Graphical User Interface

You start RPA either by clicking on an icon (on Desktop or in file browser), or typing `RPA.exe` on a command line.

Although command line arguments are not required when starting RPA, the available arguments are shown below:

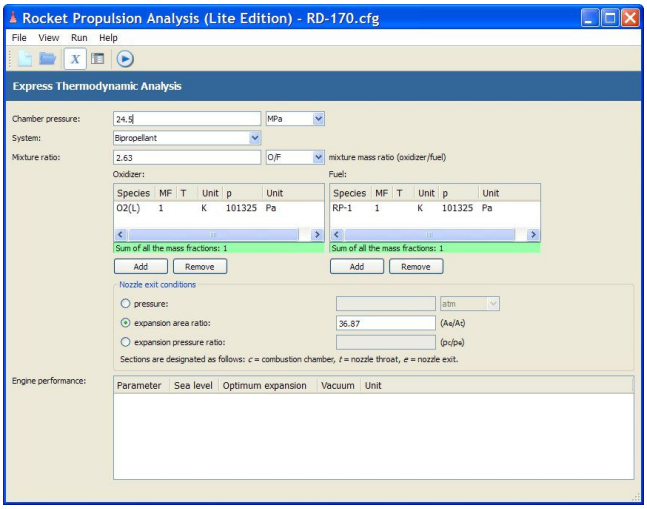
Option	Value	Description
`-t` or `--thermo`	`FILE`	Thermodynamics database. Default is resources/thermo.inp
`-ut` or `--usr_thermo`	`FILE`	User-defined thermodynamics database. Default is resources/usr_thermo.inp
`-p` or `--properties`	`FILE`	Properties database. Default is resources/properties.inp
`-up` or `--usr_properties`	`FILE`	User-defined properties database. Default is resources/properties.inp
`-i` or `--input`	`FILE`	Problem configuration file that has to be loaded. Default is last opened file.

Command line arguments must be in the command line that you use to start RPA.

See chapter Thermodynamic Database Editor for more information about database types.

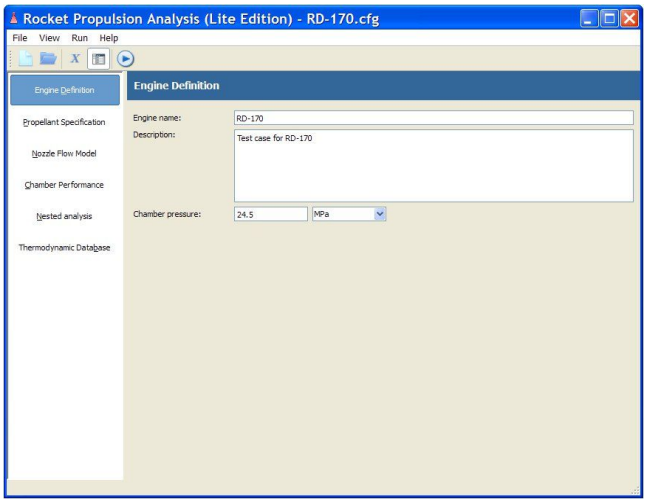
After starting the program, the RPA main window will appear. The main windows features menu bar, toolbar and working area, that can be used in two views: Express Analysis and Extended Analysis.

Express Analysis view is intended to keep the subset of input parameters and results on the same screen, and can be useful for quick analysis of the rocket engines, when theoretical performance is the only result that should be considered.

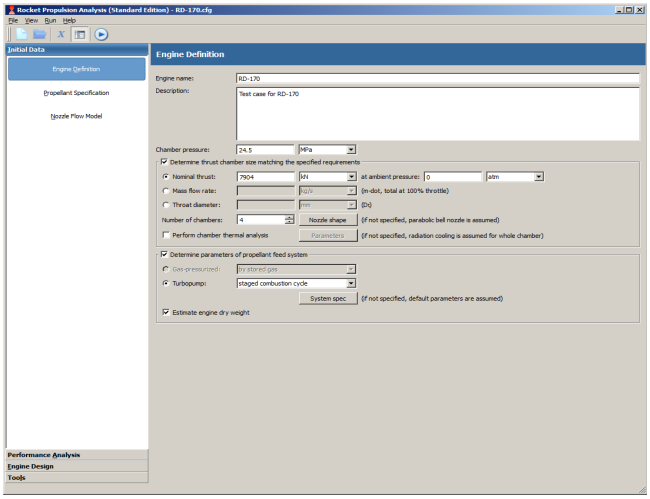


Express Analysis view

Extended Analysis view consists of several screens which conveniently group the input parameters and results. The desired screen can be activated by mouse click on corresponding button on the list at the left side of the main window. You can enlarge or narrow the list while the screens will be narrowed or enlarged, dragging the vertical bar between the list and the screens right or left.



Extended Analysis view (Lite Edition)



Extended Analysis view (Standard Edition)

In RPA Standard Edition, there are 4 different lists grouped in the following folders:

- **Initial Data** containing items Engine Definition, Propellant Specification, and Nozzle Flow Model,
- **Performance Analysis** containing items Chamber Performance, Nested Analysis, and Propwllant Analysis,
- **Engine Design** containing items Chamber Geometry, Thermal Analysis, and Propellant Feed System
- and **Tools** containing the item Thermodynamic Database.

Command-line utility

You start command-line utility by typing `rpac.exe` on a command line.

The available command-line arguments are shown below:

Option	Value	Description
`-t` or `--thermo`	`FILE`	Thermodynamics database. Default is resources/thermo.inp
`-ut` or `--usr_thermo`	`FILE`	User-defined thermodynamics database. Default is resources/usr_thermo.inp
`-p` or `--properties`	`FILE`	Properties database. Default is resources/properties.inp
`-up` or `--usr_properties`	`FILE`	User-defined properties database. Default is resources/properties.inp
`-i` or `--input`	`FILE`	Problem configuration file that has to be loaded. Default is last opened file.
`-o` or `--output`	`FILE_NAME_PREFIX`	Output file name prefix without extension. Default is "info".
`-opt` or `--optimize`		Find optimum propellant mixture ratio, bypassing the one

Option	Value	Description
		defined in input configuration file.
`-bau` or `--bau_units`		Print out results using britisj-american units.
`-pr` or `--performance`		Print out performance table
`-pt` or `--points`	`NUMBER`	Number of lines in altitude performance table

Upon completion, the command-line utility prints out the results in console window and writes it into the log file.

Scripting utility

Scripting utility is a tool that can be used to execute user's own problems.

Scripting utility can be started in either an interactive mode or a batch mode.

You start scripting utility by typing `rpas.exe` on a command line. The available command-line arguments are shown below:

The available command-line arguments are shown below:

Option	Value	Description
`-t` or `--thermo`	`FILE`	Thermodynamics database. Default is resources/thermo.inp
`-ut` or `--usr_thermo`	`FILE`	User-defined thermodynamics database. Default is resources/usr_thermo.inp
`-p` or `--properties`	`FILE`	Properties database. Default is resources/properties.inp
`-up` or `--usr_properties`	`FILE`	User-defined properties database. Default is resources/properties.inp
`-i` or `--input`	`FILE`	Script file.
`-o` or `--output`	`FILE_NAME_PREFIX`	Output file name prefix without extension. Default is "info".

Upon start up, scripting utility prints out the prompt `rpa>`, inviting you to type any valid command.

Type "exit" to stop the interactive interpreter. See Scripting Built-In Commands and Scripting API Reference to get more information about available commands.

To start scripting utility in the batch mode, specify the name of the script you want to execute as a command-line argument:

```
rpa.exe -i some_script.js
```

After completion, the scripting utility prints out the results in console window and writes it into the log file.

Configuration Files

The analysis problem input data is stored in the configuration file with extension `.cfg`. This is a specially formatted ASCII file, that can be viewed/edited in any ASCII text editor.

To start new analysis problem, create configuration file by clicking **File**, and then **New** in menu bar, or clicking icon **New** on the toolbar.

To continue with old analysis problem, load existing configuration file by clicking **File**, and then **Open** in menu bar, or clicking icon **Open** on the toolbar, and select the file to be opened. If you already have opened a configuration file, or have created new one, you will be asked to confirm the opening another configuration file.

To save the current analysis problem into the file, click **File**, and then **Save** in the menu bar. If the current analysis problem is new, you will be asked to specify the file name, or to select the existing file to be written. Otherwise the data will be saved into the same source file, that has been opened before.

You also can write the current analysis problem into another file, clicking **File**, and then **Save As...** in the menu bar.

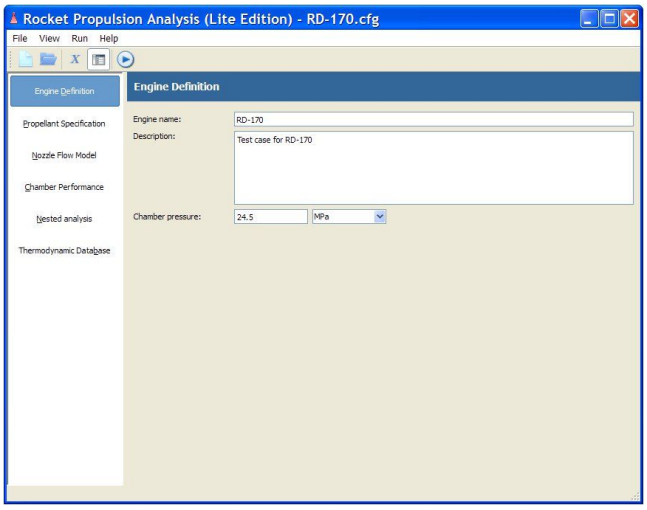
The last 10 used files are shown at the bottom of menu **File** in menu bar.

The last used configuration file is automatically opened at program start up.

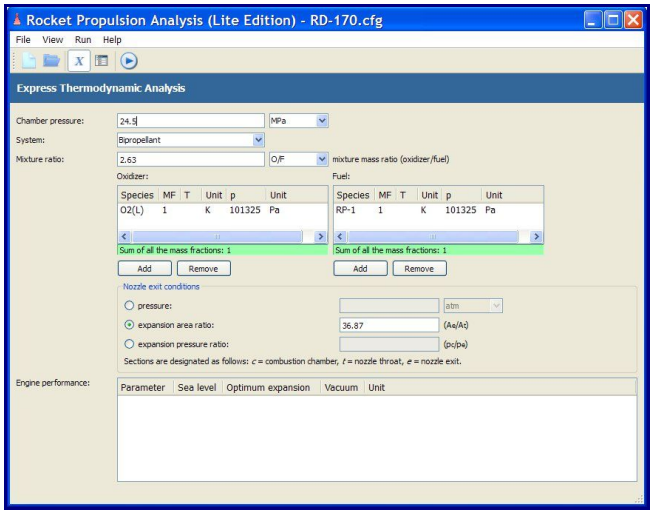
The program is shipped with a few example configuration files, located in directory examples.

Engine Definition

In the **RPA Lite Edition**, Engine Definition screen is used to define the engine name, the description and the combustion chamber pressure.



Engine Definition screen (Lite Edition)



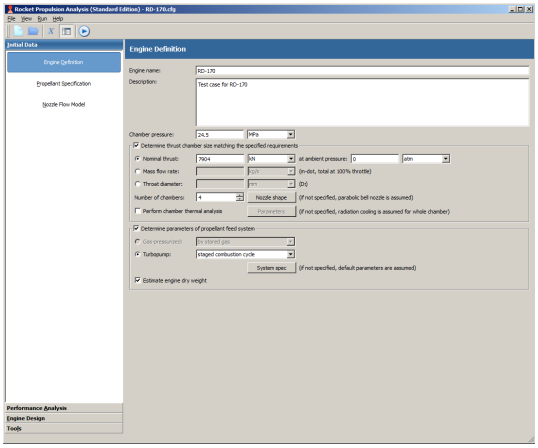
Engine Definition in Express Analysis view

Engine name and description are optional parameters, whereas the combustion chamber pressure is obligatory parameter. Note that engine name is used in results print out to identify the problem.

The pressure is an absolute pressure and can be entered using one of the following units: `MPa`, `atm`, `kg/sm`^{2}, `bar`, `psi`, `Pa`.

In the Express Analysis view the only parameter that is visible and can be changed is combustion chamber pressure.

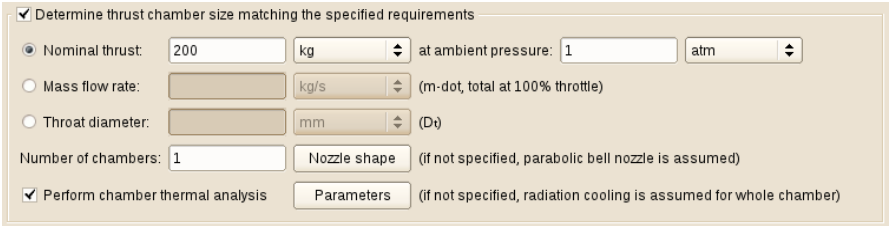
In the **RPA Standard Edition**, Engine Definition screen is also used to define the parameters for the combustion chamber and nozzle sizing, for switching on the flag for performing the thrust chamber thermal analysis, and to define the type of the engine cycle.



Engine Definition screen (Standard Edition)

The program can estimate the size of the combustion chamber and nozzle matching one of the following requirements:

- Nominal thrust at the certain ambient pressure
- Nominal mass flow rate
- Throat diameter



Chamber and nozzle sizing parameters

If specified number of chambers is greater than 1, the given thrust is a total engine thrust, and the given mass flow rate is a total mass flow rate.

If ambient pressure is not specified, it is assumed that the engine nominally operates in

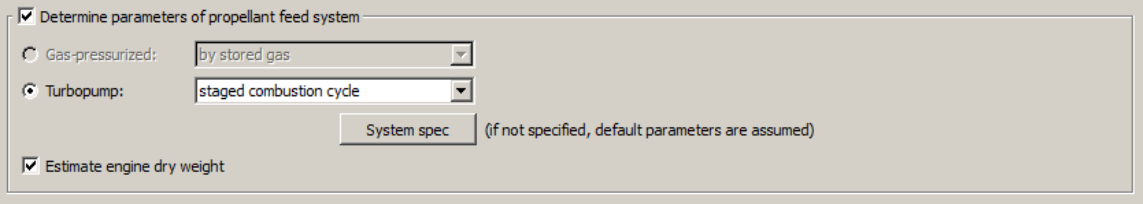
vacuum condition.

The user can specify the nozzle shape on the screen Nozzle Shape and Efficiencies, and sizing parameters on the screens Nozzle Conditions and Chamber Geometry.

If flag “Perform chamber thermal analysis” is switched on, the program will execute the thermal analysis. The user can specify additional heat transfer and chamber cooling parameters on the screens Heat Transfer Parameters and Thrust Chamber Cooling.

The program can perform engine cycle analysis, obtaining parameters of liquid propellant feed system and estimating engine weight for one of the following engine cycles:

- Gas generator cycle
- Staged combustion cycle
- Full flow staged combustion cycle



Engine cycle type

Additional parameters for cycle analysis can be specified on the screen Propellant Feed System.

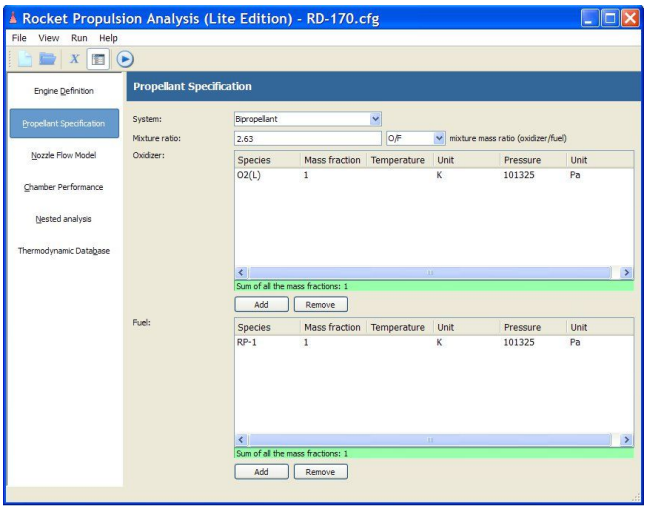
All parameters can be entered using either SI or American Customary units:

- thrust: `kN`, `kg`, `lbf`, `N`
- mass flow rate: `kg/s`, `lbm/s`
- throat diameter: `mm`, `in`, `m`, `ft`

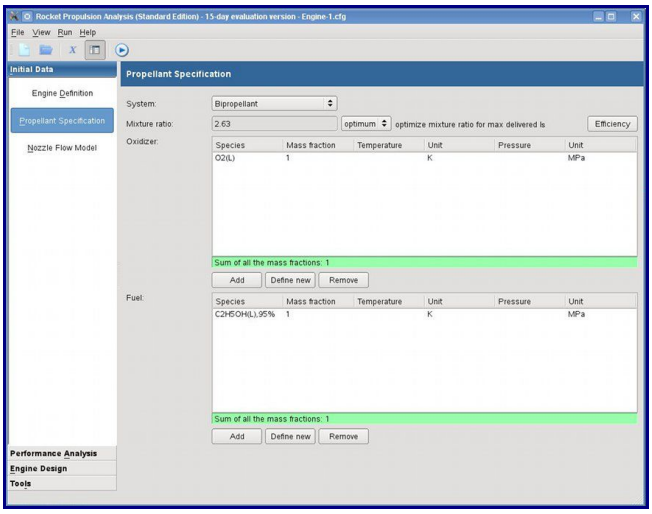
Propellant Specification

Propellant Specification Screen

Propellant Specification screen is intended to specify used propellant component/s.

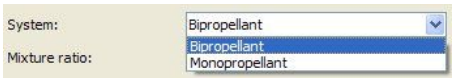


Propellant Specification screen (Lite Edition)



Propellant Specification screen (Standard Edition)

You can choose between bipropellant and monopropellant propulsion systems, selecting the corresponding item in the list box at the top of the screen:



Note: although you have the choice between bipropellant and monopropellant propulsion systems only, *there is a possibility to specify three (or more) propellant components*.

See section **How to...** (http://www.propulsion-analysis.com/howto/index.htm) on RPA web site for further details.

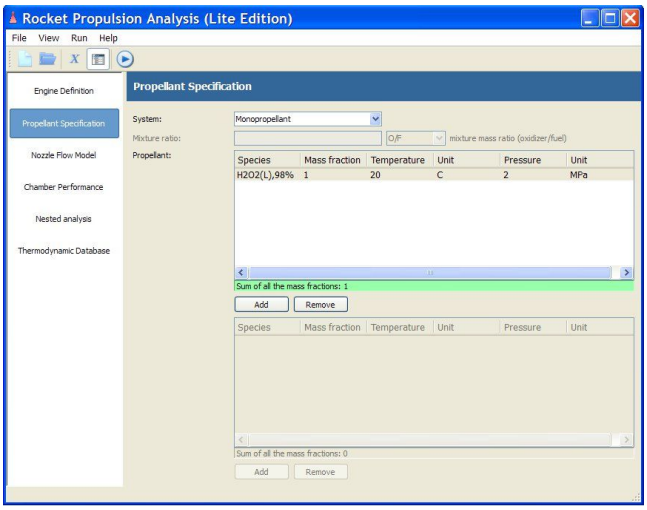
For bipropellant systems, the lists for both Oxidizer and Fuel are enabled (see figure *"Propellant Specification screen"* above), as well as the fields for specifying a mixture ratio.

The mixture ratio can be specified either as an O/F ratio (ratio of "oxidizer flow rate" to "fuel flow rate"), or as an oxidizer excess coefficient, given as ratio of desired O/F to stoichiometric O/F.

You can also select an item "optimum":

In this case the mixture ratio will be optimized for getting maximum specific impulse under given conditions. The found optimum O/F ratio will be displayed on the screen Chamber Performance.

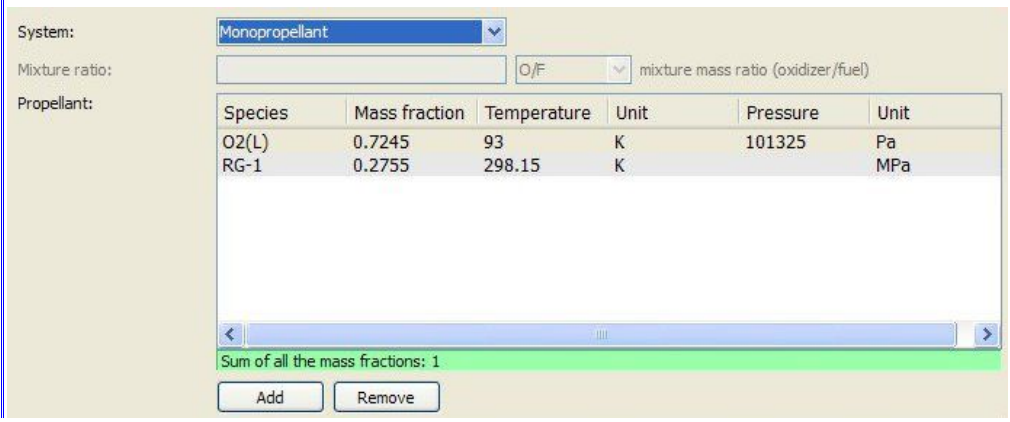
For monopropellant system, the single list of propellant components is enabled, whereas the Fuel list and fields for specifying mixture ratio are disabled.



Specifying monopropellant system

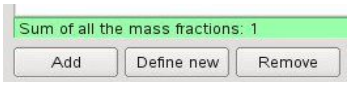
Note: for the thermodynamic calculation, the only difference between bipropellant system and monopropellant system that contains two species, is how the species mixture is defined. Actually, you can define any bipropellant system (as well as three- or more propellant systems) as a monopropellant system, specifying the proper mass fraction for each component on the list.

For instance, the following "monopropellant" configuration is equivalent to the bipropellant one with O/F=0.7245/0.2755=2.63:



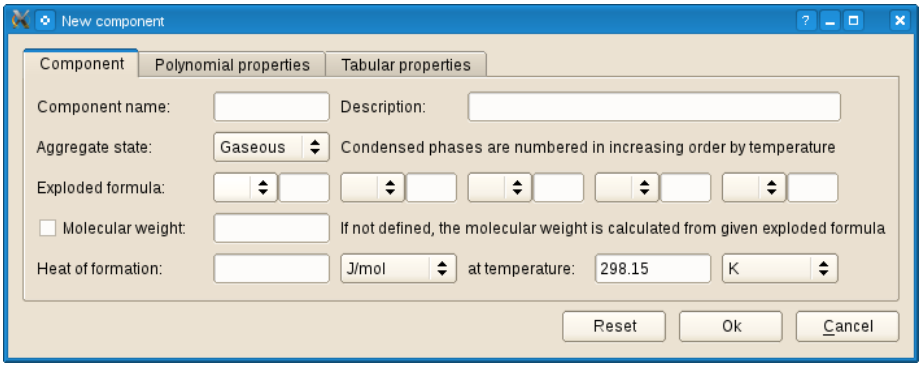
Each component list (Oxidizer, Fuel, or Propellant) contains one or more species, displayed on the single row. To add species to the list, click the button **Add** at the bottom of the corresponding list. To remove the selected species from the list, click the button **Remove**.

To add new species, click the button **Define new** (available in RPA Standard Edition only):



In the appeared dialog window, specify at least the name, the exploded chemical formula, and

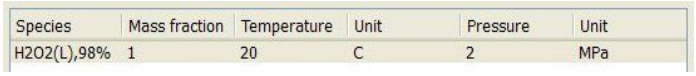
the heat of formation of new species:



Once the new species was defined, it is stored in the user thermodynamic database and permanent available for using in other problems.

Component Properties

The component on the list features 4 parameters: species name, mass fraction of the species in the component (for bipropellant systems) or propellant (for monopropellant systems), initial temperature of the species, and initial pressure of the species:



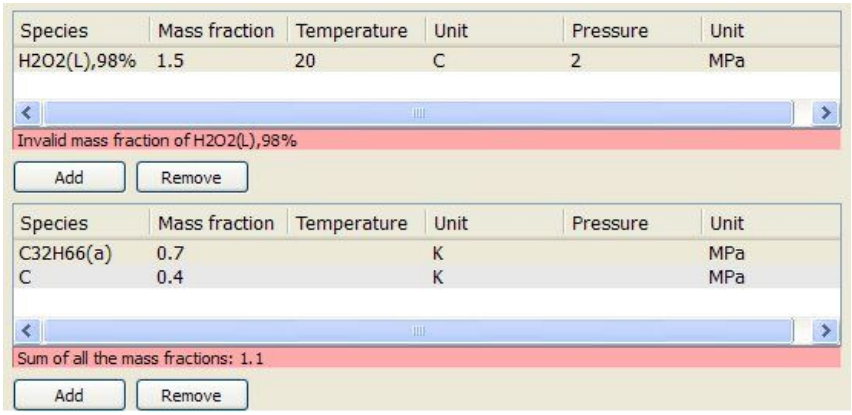
Species	Mass fraction	Temperature	Unit	Pressure	Unit
H2O2(L),98%	1	20	C	2	MPa

Initial species temperature and pressure are optional parameters. If not specified, the following default values will be assigned automatically:

- `p = 1 atm, T = 298.15 K` for non-cryogenic species
- `T =` [boiling point temperature] for cryogenic liquids

When composing component (for bipropellant systems) or propellant (for monopropellant systems) from several species, the sum of all the mass fractions of components on the same list has to be equal to 1. To change the mass fraction for the species, double-click on the corresponding cell, enter the new value and press Enter button (or click away).

Each list features the automatic mass fraction checker, that displays the current sum in the list footer. If the sum is correct, the background color of the footer is light-green, otherwise the color is light-red:



To specify the initial temperature and/or pressure of the species, double-click on the corresponding cell, enter the new value and then press Enter button (or click away):

To change the unit, double-click on the corresponding cell, select the desired unit on the list, and then press Enter button (or click away):

The temperature can be entered using one of the following units: `K`, `C`, `F`, `R`.

The pressure is an absolute pressure and can be entered using one of the following units: `MPa`, `atm`, `kg/sm`^{2}, `bar`, `psi`, `Pa`.

Note: the initial temperature and/or pressure can only be specified for the components which are supplied together with thermodynamic properties either in the polynomial form or in tabular form.

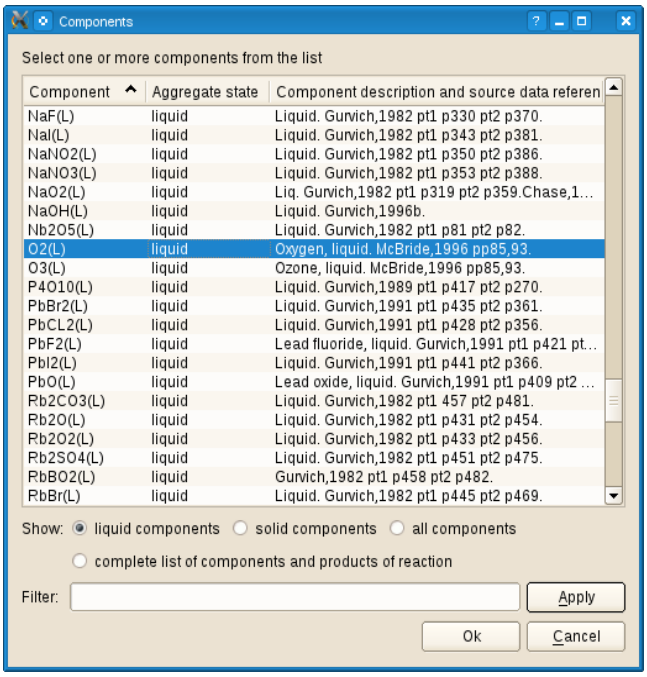
See chapter Thermodynamic Database Editor for further details.

Note: if the components change their temperature/pressure due to the work performed by components themselves, you should assign initial `(T,p)` which components had *before the work is performed*.

For instance, in staged combustion engine, a turbopump is powered by components, and the correct initial parameters correspond to components' conditions at the pump *inlets*, in opposite to the conditions at pump outlets or combustion chamber injector.

Components Database

After clicking on button **Add**, the dialog window *"Components"* appears. The content of the dialog window depends on the type of target list, where component will be added.



Components Database

When adding new component to the oxidizer list, the dialog window displays available oxidizers; when adding new component to the fuel list, the dialog window displays available fuels. For monopropellant systems, both oxidizers and fuels will be displayed in the dialog window.

You can filter the list in the dialog window, using a regular expression. The filter pattern is applied to both columns of the table.

Mark the check box *"Show complete list of available reactants and product of reaction"* if you want to see all species, including atomized and/or ionized products of reaction, or keep it unmarked if you want to see only possible propellant components.

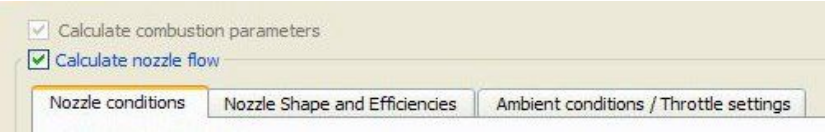
Select one or more species on the list and click the button **OK**. Click the button **Cancel** if you want to leave without adding any species.

Nozzle Flow Model

Nozzle Flow Model Screen

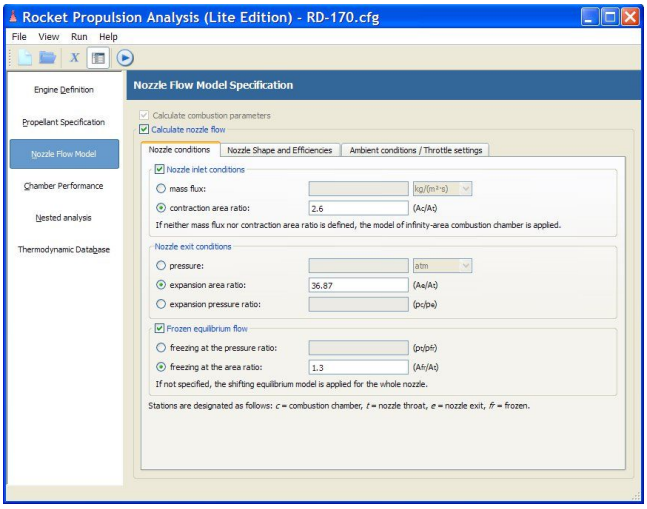
The program can either calculate the combustion parameters in the combustion chamber, or perform the complete engine performance analysis, calculating the flow through the nozzle.

Clear the check box *"Calculate nozzle flow"*, in order to choose the first possibility, or mark it to start the nozzle flow analysis:

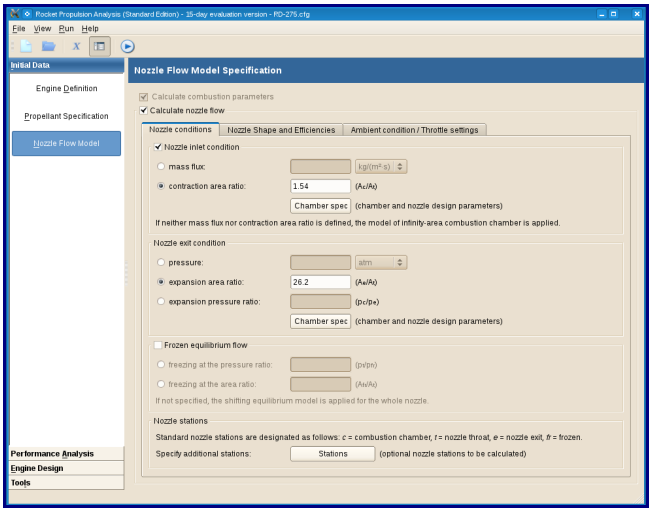


Nozzle conditions

If you are solving the nozzle flow problem, you have to define at least nozzle exit conditions, specifying one of three parameters: nozzle exit pressure, nozzle expansion area ratio, or nozzle expansion pressure ratio.



Nozzle conditions (Lite Edition)



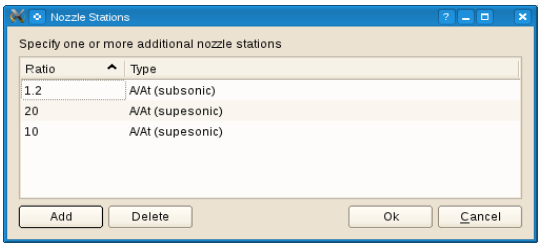
Nozzle conditions (Standard Edition)

The program can calculate the performance for a combustion chambers with finite or infinite cross section area. The default model assumes the infinite cross section area of the chamber. To switch to the finite-area model, mark the check box *"Nozzle inlet conditions"*, and then specify the chamber contraction area ratio `Ac/At` or mass flux in the combustion chamber.

If specified, the nozzle inlet and exit conditions are used for chamber and nozzle sizing, together with additional parameters on the screen Chamber Geometry (press the button **Chamber spec** to jump directly to the that screen).

The program can calculate the performance with respect to the shifting and frozen chemical equilibrium in the nozzle. The default model assumes the shifting chemical equilibrium in the whole nozzle. To switch to the frozen chemical equilibrium model, mark the check box *"Frozen equilibrium flow"*, and then specify the nozzle section, where application of this model should be started.

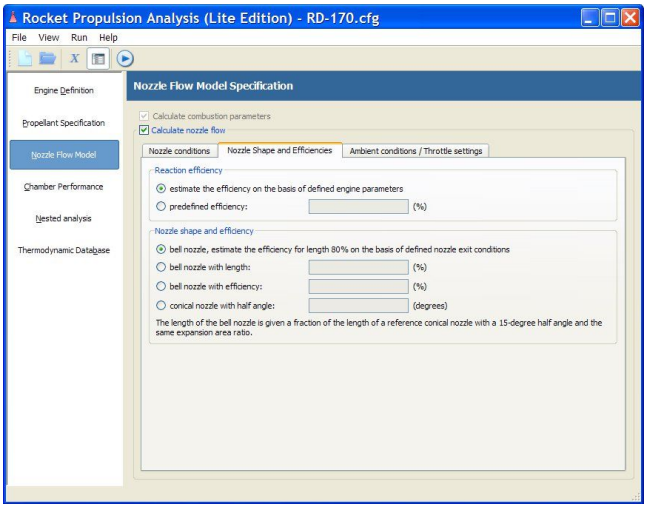
By default, the program calculates parameters at following nozzle stations: nozzle inlet, nozzle throat and nozzle exit. To specify additional nozzle stations, press the button **Stations** and specify one or more additional stations, defining them by area ratio `Ac/At` or by pressure ratio `pc/p`:



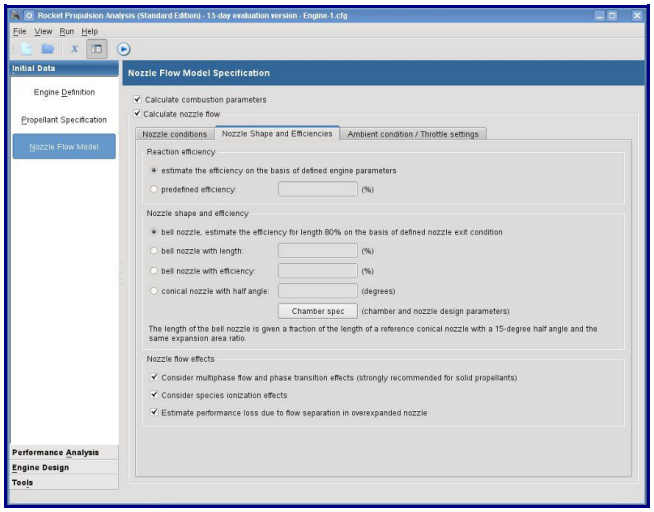
Nozzle Stations (Standard Edition)

Nozzle Shape and Efficiencies

The program calculates both theoretical and delivered engine performance. You can define correction factors on the screen *Nozzle Shape and Efficiencies*, otherwise the program estimates it on the basis of defined engine parameters, such as chamber pressure, propellant components and nozzle conditions.

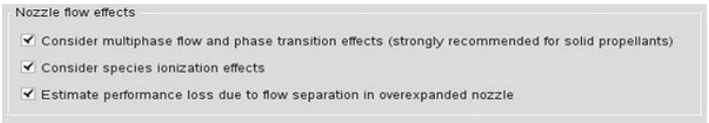


Nozzle Shape and Efficiencies (Lite Edition)



Nozzle Shape and Efficiencies (Standard Edition)

In RPA Standard Edition, you can also control the considered by the program nozzle flow effects:



Switch off the flag **Consider multiphase flow and phase transition** in order to suppress the calculation of multiphase flow effects. Note that for the most of solid propellant problems this flag should be switched on.

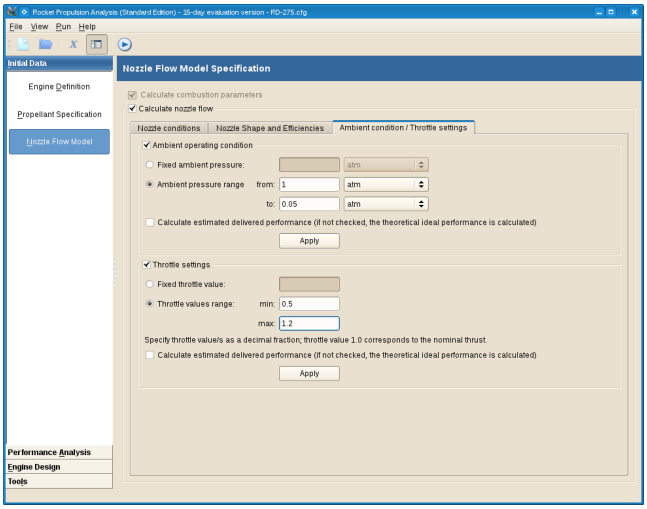
Switch off the flag **Consider species ionization effects** in order to to suppress the calculation of species ionization effects.

Switch off the flag **Estimate performance loss due to flow separation** in order to disable the additional calculation of flow separation effects.

Ambient condition

By default the program calculates performance of the rocket engine at the sea level conditions (`pa=1 atm`, or `14.7 psi`), optimum nozzle expansion (`pe=pa`), and vacuum conditions (`pa=0`).

To calculate the performance at desired ambient conditions, you can also explicitly specify either the specific ambient pressure or the range of ambient pressures given as high and low range values.



Ambient condition

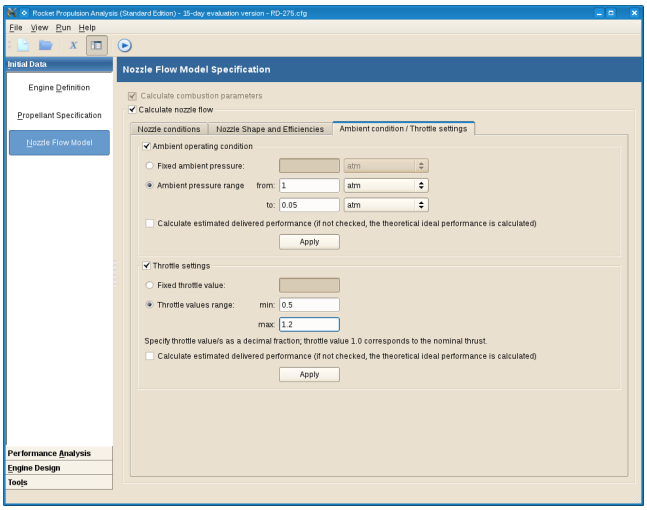
Switch on the flag “Calculate estimated delivered performance” in order to apply the performance corrections factors to results of calculation.

The pressure is an absolute pressure and can be entered using one of the following units: `MPa`, `atm`, `kg/sm`^{2}, `bar`, `psi`, `Pa`.

Throttle settings

By default the program calculates performance of the rocket engine assuming the propellant flow rate that correspond to nominal thrust.

To calculate the performance at desired throttle settings, you can also explicitly specify either the specific throttle value, or the range of throttle values given as high and low range values.



Throttle settings

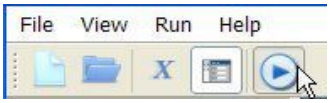
Switch on the flag “Calculate estimated delivered performance” in order to apply the performance corrections factors to results of calculation.

The throttle value is a ratio of propellant flow rate at desired throttle setting to flow rate that correspond to nominal thrust (100% flow rate).

Starting Analysis

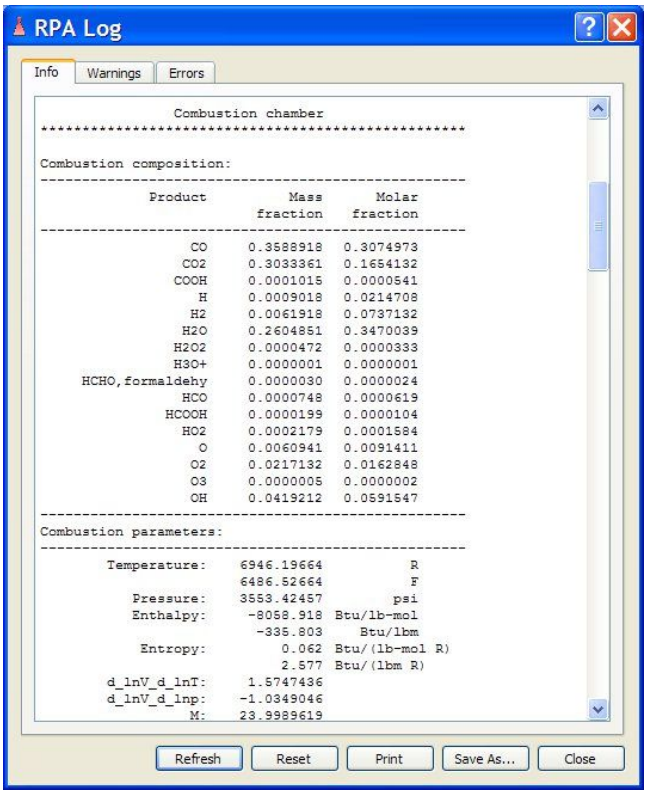
After specifying the initial data, or opening existing configuration file, you can start the analysis by clicking menu **Run**, and then **Start analysis** in menu bar:

or clicking icon **Start** on the main toolbar:



After successful finishing the analysis the program automatically switches to the screen Chamber Performance, that displays the calculated results.

You can also print out the results, clicking menu **Run**, and then **Print results**, or check the analysis log, clicking menu **Run**, and then **Show log** in menu bar. The analysis log as an ASCII file is also available in the user directory.



Analysis Log

Chamber Performance

The screen Chamber Performance consists of 4 tabs *Thermodynamic properties*, *Performance*, *Altitude performance*, and *Throttled performance*.

You can print out the results, clicking the button **Print**, or save the results as ASCII or HTML file, clicking the button **Save As...** at the right-bottom corner of the screen.

Propellant properties

The following propellant properties are available in the analysis log:

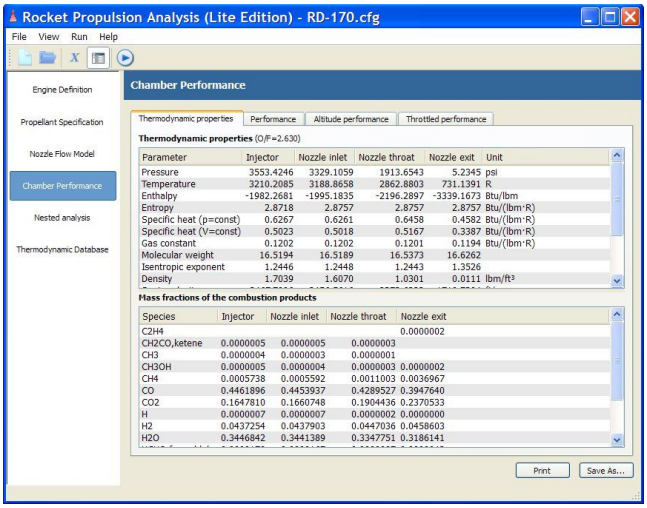
- propellant exploded formula,
- oxidizer excess coefficient,
- stoichiometric mixture ratio,
- propellant density (only if density values of all components are available).

```
Propellant:
------------------------------------------------
    Component  Temp.     Pressure        Mass
                [K]       [MPa]       fraction
------------------------------------------------
        H2(L)   20.3       0.1013      0.1428571
        O2(L)   90.2       0.1013      0.8571429
------------------------------------------------
Propellant exploded formula: (O)0.549 (H)1.451
              alpha:      0.7559833 (oxidizer excess coefficient)
                O/F:      6.0000000
              O/F 0:      7.9366827 (stoichiometric)
                rho:      932.86285      kg/m^3
```

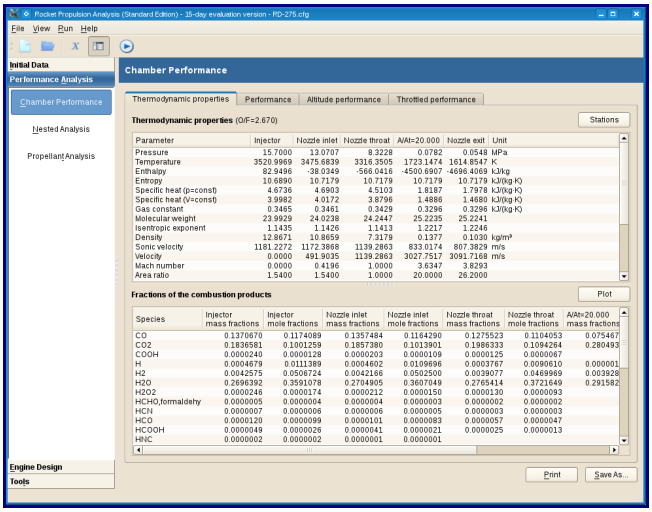
Propellant properties in the analysis log

Thermodynamic properties

The tab *Thermodynamic properties* displays the parameters of reaction products and their mass fractions at the chamber stations involved into the analysis.

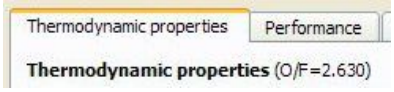


Thermodynamic properties (Lite Edition)



Thermodynamic properties (Standard Edition)

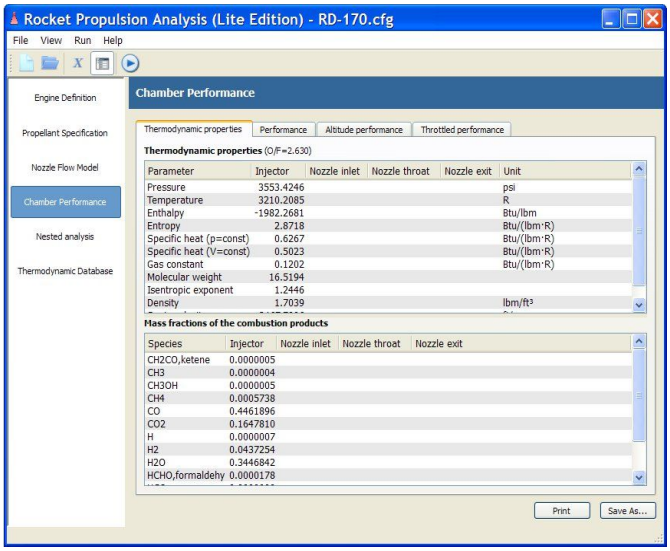
The current propellant components ratio (either explicitly defined on the screen Propellant Specification, or found by optimizer) is displayed in the header of the top table on the tab:



If the problem is configured to solve the parameters in the combustion chamber (see chapter Nozzle Flow Model), the only available nozzle station is *Injector* (see figure *Injector thermodynamic properties*), otherwise the available chamber stations are *Injector*, *Nozzle inlet*, *Nozzle throat* and *Nozzle exit* as well as additional stations, defined on screen *Nozzle Flow Model* (see figure *Thermodynamic properties above*).

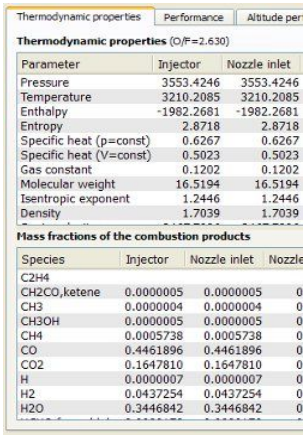
You can specify additional nozzle stations either on screen Nozzle Flow Model, or on screen

Thermodynamic properties, pressing the button **Stations**.



Injector thermodynamic properties

If the problem is configured to calculate the performance for the combustion chamber with infinite cross section area, the parameters for *Nozzle inlet* station are identical to that at *Injector* station:

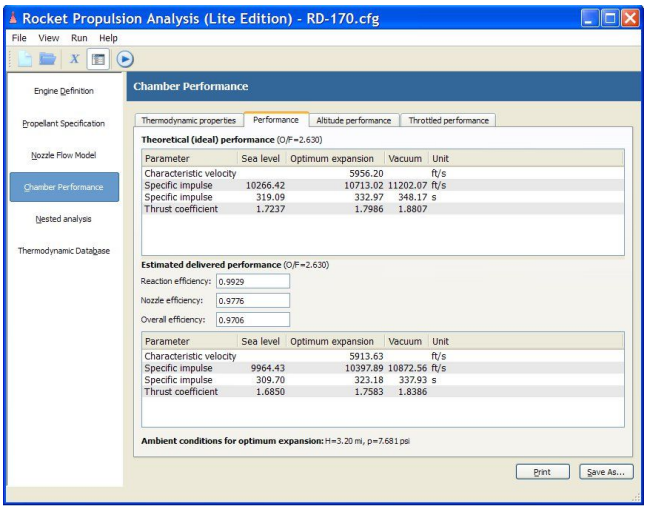


The thermodynamic parameters are displayed in the current units, defined in the Preferences Dialog.

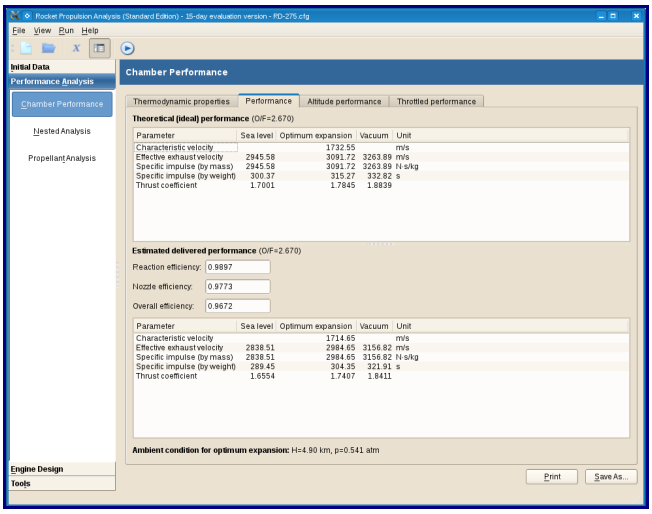
You can shrink or heighten the top table while the bottom table will be heightened or shrunk, dragging the horizontal bar between the tables down or up.

Performance

The tab *Performance* displays the theoretical (ideal) performance of the chamber, as well as its estimated delivered performance and the correction factors used to calculate the delivered performance.



Chamber performance (Lite Edition)



Chamber performance (Standard Edition)

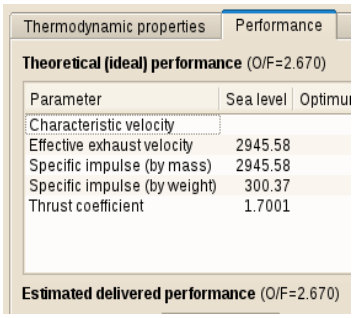
Note: if you are analyzing a pressure-fed-cycle or staged-combustion-cycle engine, the calculated performance is actually an *engine* performance.

If the thrust chamber sizing parameters have been configured (see screen Engine Definition), the program calculates the divergence thrust loss and displays it on the screen Thrust Chamber Size and Geometry:

```
    De =   58.44   mm
e_div = 0.00478   (divergence loss)
```

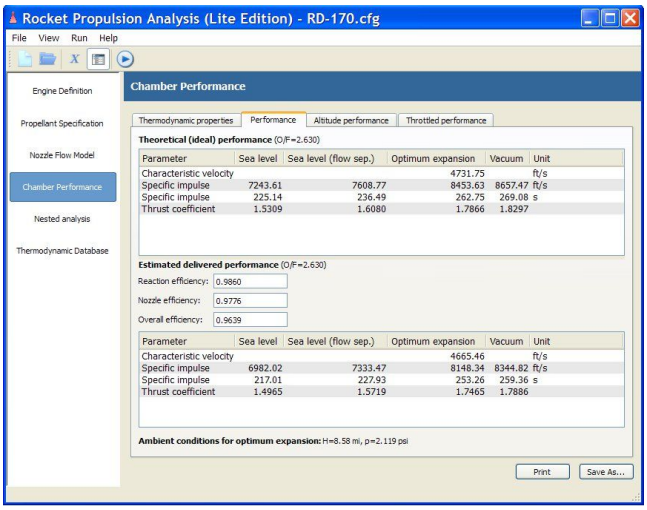
After the first program run, you can use the calculated value of divergence thrust loss as well the calculated value of friction thrust loss (see Thermal Analysis) to adjust the nozzle efficiency on the screen Nozzle Shape and Efficiencies.

The current propellant components ratio (either explicitly defined on the screen Propellant Specification, or found by optimizer) is displayed in the header of the both tables on the tab:



If the problem is configured to solve the combustion parameters at *Injector* station (see chapter Nozzle Flow Model), the performance parameters are not available, otherwise the tab displays calculated performance of the chamber at the sea level conditions (`pa=1 atm`, or `14.7 psi`), optimum nozzle expansion (`pe=pa`), and vacuum conditions (`pa=0`).

In case if flow separation occurs in the nozzle, the additional column displays the performance at sea level with respect to the flow separation.



Chamber performance with flow separation

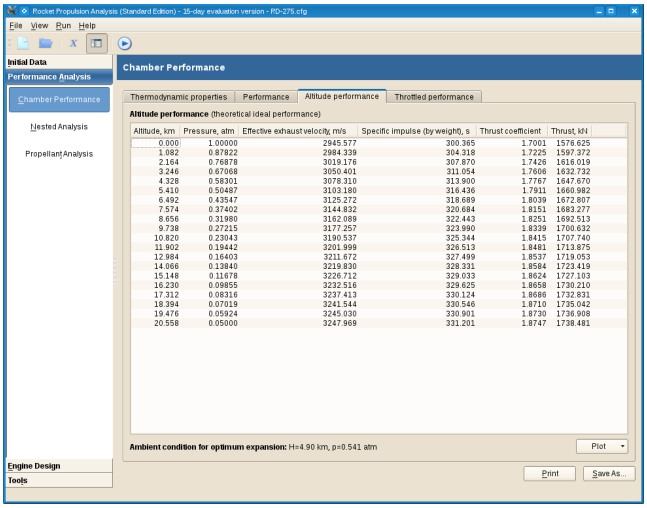
Ambient conditions for optimum nozzle expansion (altitude and ambient pressure) are displayed at the bottom of the tab:

The performance parameters are displayed in the current units, defined in the Preferences Dialog.

You can shrink or heighten the top table while the bottom table will be heightened or shrunk, dragging the horizontal bar between the tables down or up.

Altitude performance

The tab *Altitude performance* displays the performance of the chamber in the specified ambient conditions.



Altitude performance

By default, the program calculates ideal theoretical performance. If you want to calculate estimated delivered performance, switch on the flag “Calculate estimated delivered performance” on screen *Ambient Conditions*.

If the problem is configured to solve the combustion parameters at *Injector* station (see chapter Nozzle Flow Model), the performance parameters are not available, otherwise the tab displays calculated performance of the chamber at the defined altitude and ambient pressure.

You can plot the diagrams *"specific impulse vs. altitude"*, *"thrust coefficients vs. altitude"* or *"thrust vs. altitude"* clicking the button **Plot** at the bottom-right corner of the tab. The altitude of the optimum expansion, as well as the altitude of flow separation (if occurs) is shown on the diagram by corresponded vertical marker line/s.

Plot - altitude performance

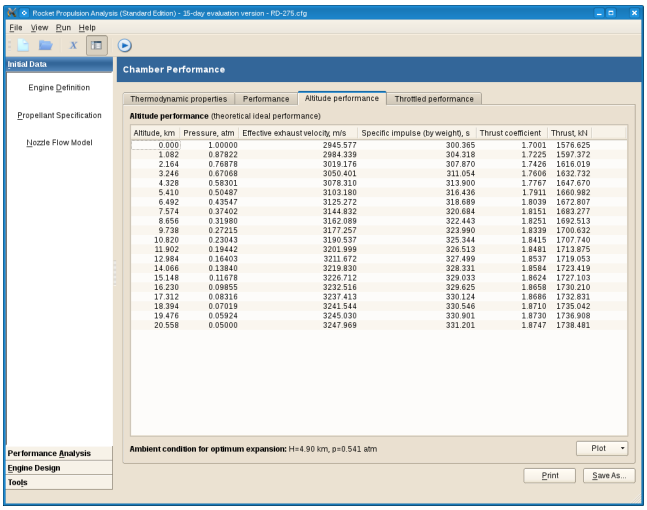
Plot - altitude performance with flow separation

Ambient conditions for optimum nozzle expansion (altitude and ambient pressure) are displayed at the bottom of the tab.

The performance parameters are displayed in the current units, defined in the Preferences Dialog.

Throttled performance

The tab *Throttled performance* displays the performance of the chamber at the specified throttle values.

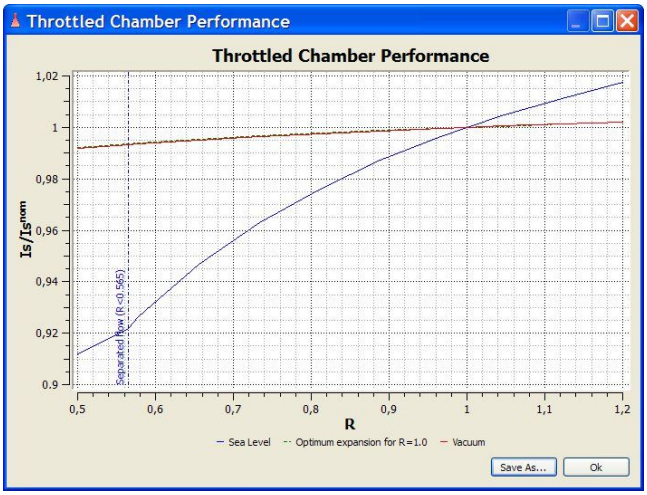


Throttled performance

By default, the program calculates ideal theoretical performance. If you want to calculate estimated delivered performance, switch on the flag “Calculate estimated delivered performance” on screen *Throttle Settings* .

If the problem is configured to solve the combustion parameters at *Injector* station (see chapter Nozzle Flow Model), the performance parameters are not available, otherwise the tab displays calculated performance of the rocket engine at the sea level conditions (`pa=1 atm`, or `14.7 psi`), optimum nozzle expansion (`pe=pa`), and vacuum conditions (`pa=0`) at the defined throttle values.

You can plot the diagrams *"specific impulse vs. throttle value"* or *"thrust vs. throttle value"* clicking the button **Plot** at the bottom-right corner of the tab. The throttle value where flow separation occurs is shown on the diagram by corresponded vertical marker line.



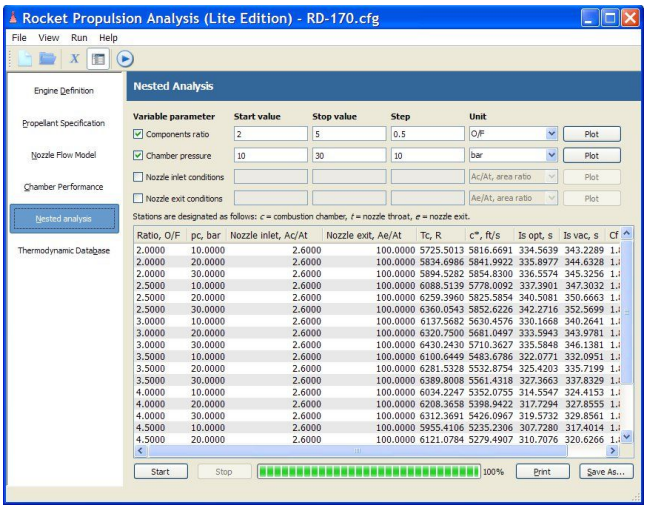
Plot - throttled performance

The performance parameters are displayed in the current units, defined in the Preferences

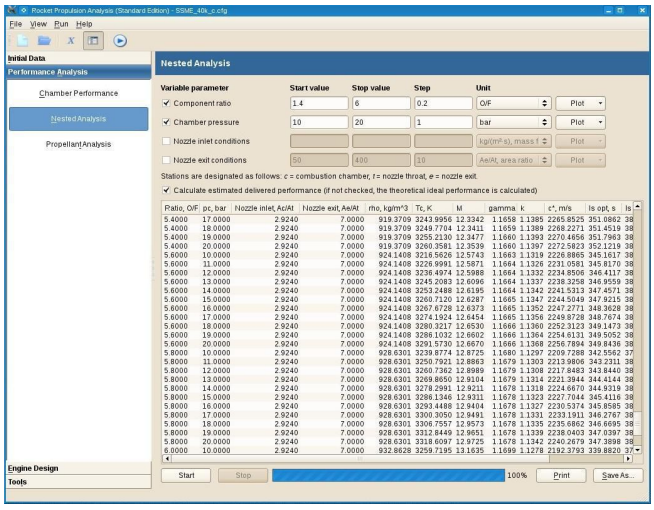
Dialog.

Nested Analysis

Using nested analysis, you can evaluate the performance of the rocket chamber for the range of parameter/s, stepping of up to four independent variables (component ratio, chamber pressure, nozzle inlet conditions, nozzle exit conditions).



Nested analysis (Lite Edition)



Nested analysis (Standard Edition)

Nested analysis inherits the initial configuration parameters, defined on the screens Engine Definition, Propellant Specification, and Nozzle Flow Model. You can define up to four variables parameters, which replace corresponding parameter/s of initial configuration, and start the nested analysis, clicking the button **Start** at the left-bottom corner of the tab.

Note: since the program performs the thermodynamic analysis for each unique combination of variable parameters, definition of parameters with a small step can lead to very long calculation time.

By default, the program calculates ideal theoretical performance. If you want to calculate estimated delivered performance, switch on the corresponding flag:

The program performs the thermodynamic analysis for each unique combination of variable parameters, stepping it by defined value, and displaying the results in the table at the bottom of the tab. The current status of nested analysis is shown by progress bar at the bottom of the tab. The running analysis can be stopped by clicking the button **Stop**.

After finishing the nested analysis, you can plot the diagrams *"specific impulse vs. variable parameter"*, *"chamber temperature vs. variable parameter"*, *"characteristic velocity vs. variable parameter"* or *"thrust coefficient vs. variable parameter"*, clicking the button **Plot** at

the right side of the corresponding parameter configuration.

Nested analysis plot

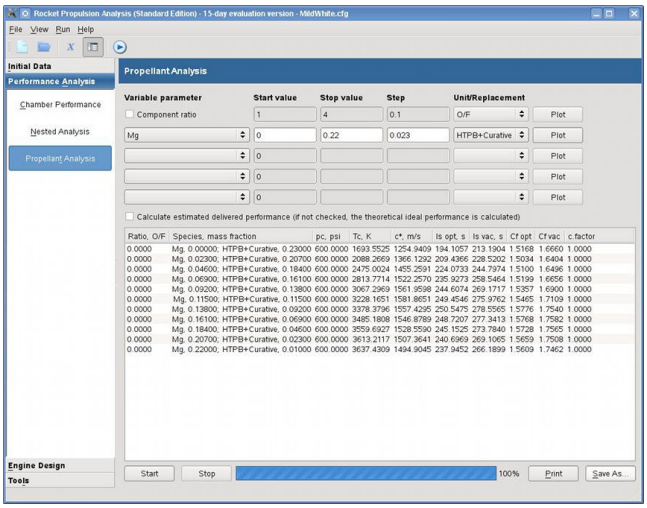
You can print out the results of nested analysis, clicking the button **Print**, or save the results as ASCII or HTML file clicking the button **Save As...** at the right-bottom corner of the screen.

In RPA Standard Edition, you can also save the results in PDF or ODF formats.

The defined parameters of the nested analysis are stored in the global INI-file and shared between different configurations.

Propellant Analysis

Using propellant analysis tool, you can evaluate different propellant compositions, with stepwise replacement of one component with another one.

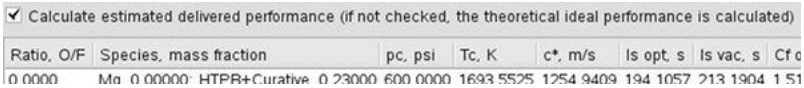


Propellant analysis

Propellant analysis inherits the initial configuration parameters, defined on the screens Engine Definition, Propellant Specification, and Nozzle Flow Model. You can define up to four pairs of component, which replace corresponding components of initial propellant composition, and start the propellant analysis, clicking the button **Start** at the left-bottom corner of the tab.

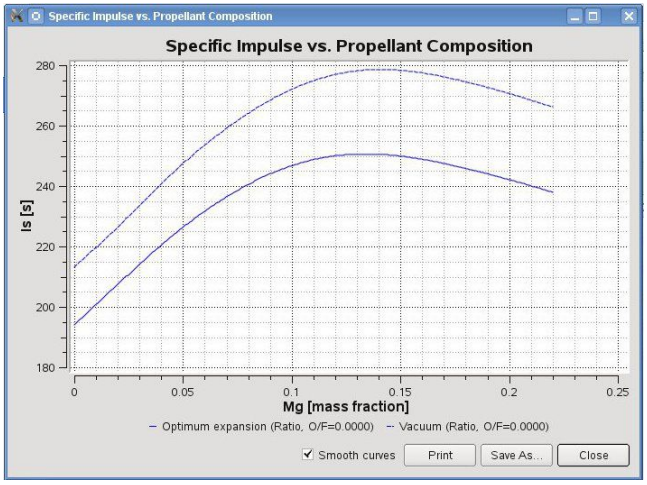
Note: since the program performs the thermodynamic analysis for each unique combination of variable parameters, definition of parameters with a small step can lead to very long calculation time.

By default, the program calculates ideal theoretical performance. If you want to calculate estimated delivered performance, switch on the corresponding flag:



The program performs the thermodynamic analysis for each unique combination of variable parameters, stepping it by defined value, and displaying the results in the table at the bottom of the tab. The current status of propellant analysis is shown by progress bar at the bottom of the tab. The running analysis can be stopped by clicking the button **Stop**.

After finishing the propellant analysis, you can plot the diagrams *"specific impulse vs. variable parameter", "chamber temperature vs. variable parameter", "characteristic velocity vs. variable parameter" or "thrust coefficient vs. variable parameter"*, clicking the button **Plot** at the right side of the corresponding parameter configuration.



Propellant analysis plot

You can print out the results of propellant analysis, clicking the button **Print**, or save the results as ASCII or HTML file clicking the button **Save As...** at the right-bottom corner of the screen.

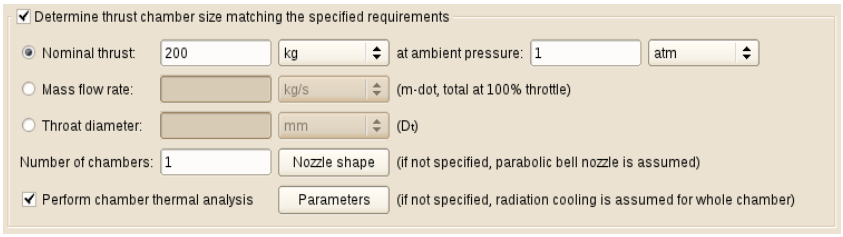
In RPA Standard Edition, you can also save the results in PDF or ODF formats.

The defined parameters of the propellant analysis are stored in the global INI-file and shared between different configurations.

Chamber Geometry

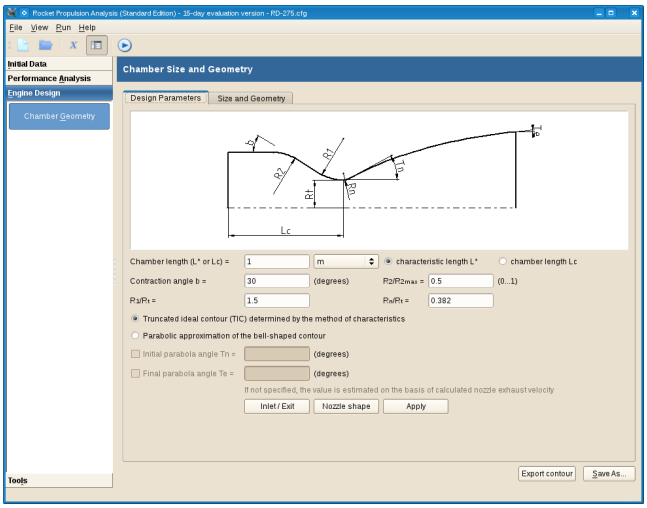
The screen Chamber Geometry consists of 2 tabs *Design Parameters* and *Size and Geometry*.

The input of parameters on the screen is only enabled if the flag "Determine thrust chamber size" on the screen Engine Definition is switched on, and at least one of sizing parameters is provided:



Design Parameters

On the tab *Design Parameters* you can define design parameters used to calculate the size of the combustion chamber and the shaped of the nozzle.

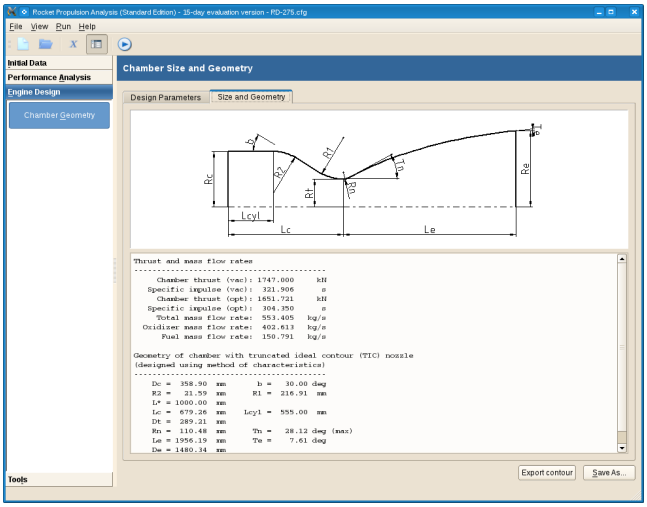


Design parameters

Note: such design parameters as nozzle inlet and exit conditions, as well as a nozzle shape (bell or conical) can be only changed on screen Nozzle Flow Model. Use provided buttons to jump direct to that screen and back.

Size and Geometry

The tab *Size and Geometry* displays the calculated size of the chamber and nozzle.



Size and geometry

If the thrust chamber sizing parameters have been configured (see screen Engine Definition), the program calculates the divergence thrust loss and displays it on the screen Thrust Chamber Size and Geometry:

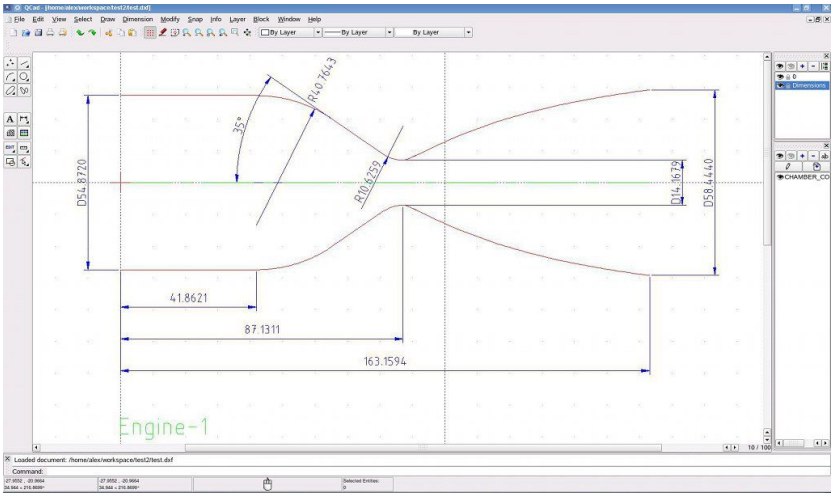
```
    De =   58.44   mm
e_div = 0.00478   (divergence loss)
```

After the first program run, you can use the calculated value of divergence thrust loss as well the calculated value of friction thrust loss (see Thermal Analysis) to adjust the nozzle efficiency on the screen Nozzle Shape and Efficiencies.

You can save the results as an ASCII file, clicking the button **Save As...**, or export the contour to the DXF file clicking the button **Export to DXF** at the right-bottom corner of the screen.

If exporting to DXF file, you provide additional export parameters:

The resulting DXF file can be opened in any CAD program.



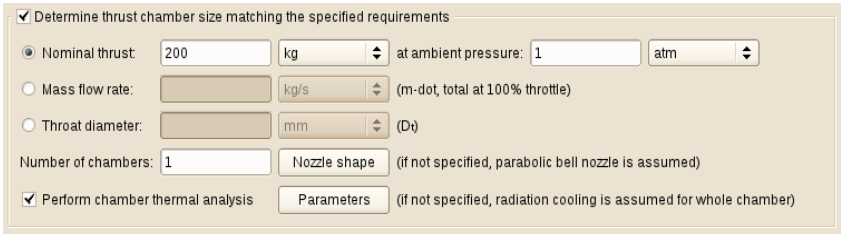
DXF file opened in QCad

Thermal Analysis

The program is capable of performing the steady-state thermal analysis.

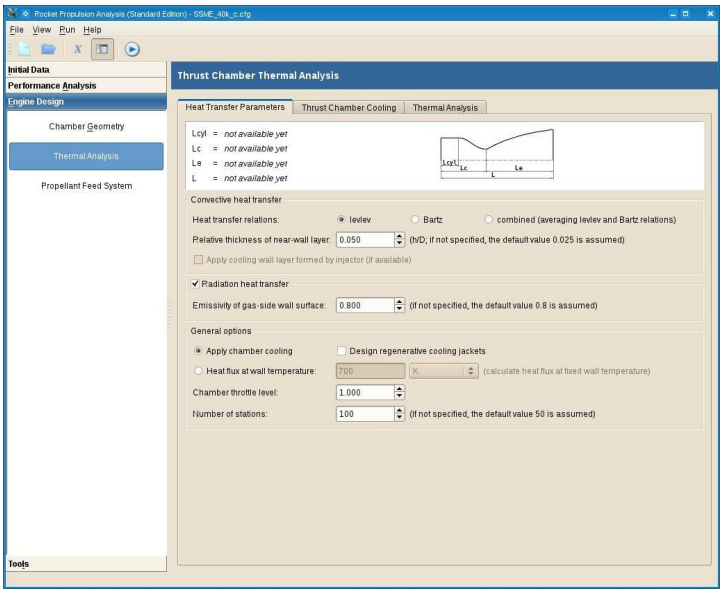
The screen Thermal Analysis consists of 3 tabs *Heat Transfer Parameters*, *Thrust Chamber Cooling* and *Thermal Analysis*.

The input of parameters on the screens are only enabled if both flags "Determine thrust chamber size" and "Perform chamber thermal analysis" on the screen Engine Definition are switched on, and at least one of sizing parameters is provided:



Heat Transfer Parameters

On the tab *Heat Transfer Parameters* you can define heat transfer parameters used to calculate the heat transfer rate distribution in the thrust chamber.



Heat transfer parameters

The program implements two different methods of calculating the gas-side heat transfer rates: levlev approach for calculation of convective heat transfer in the nozzle and Bartz semi-empirical correlation for gas-side heat transfer coefficient.

The user has the possibility to choose either one of two mentioned methods or combined approach, which calculates the resulting heat transfer rate as an average value of that calculated by levlev and Bartz methods.

The program can perform the thermal analysis, calculating the heat flux either as a convective heat transfer only or summing the contributions of convective and radiation heat transfers. In the last case, the user shall specify a radiation emissivity coefficient of the wall inner surface.

Option *Heat flux at wall temperature* allows to estimate the heat flux distribution at specified constant temperature of the gas-side wall surface.

Option *Apply chamber cooling* allows to estimate the temperature and heat flux distribution, applying the specified cooling (see chapter *Thrust Chamber Cooling*). When option *Design regenerative cooling jackets* is activated, the parameters of cooling jackets (such as diameter and number of tubes, or size and number of channels) will be automatically adjusted. Note

that you still need to define the location, type and connection of cooling jackets, thermal conductivity of the wall, and thickness of inner wall (or tube wall).

After running the thermal analysis, the obtained parameters of cooling jackets are available for editing under the tab *Thrust Chamber Cooling*.

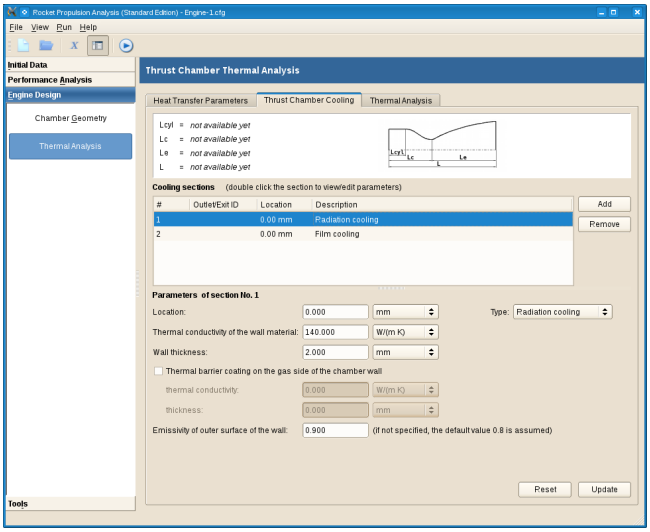
Option *Design regenerative cooling jackets* is not stored in configuration file and will be automatically deactivated after finishing the thermal analysis.

The parameter *Chamber throttle level* defines the throttle level of the engine. This parameter affects the conditions in combustion chamber and mass flow rate through the cooling jackets.

The parameter *Number of stations* defines the number of stations evenly distributed along the thrust chamber. In addition, the program automatically inserts stations at nozzle inlet and nozzle throat, as well as at location of film cooling slots and cooling sections (see *Thrust Chamber Cooling*).

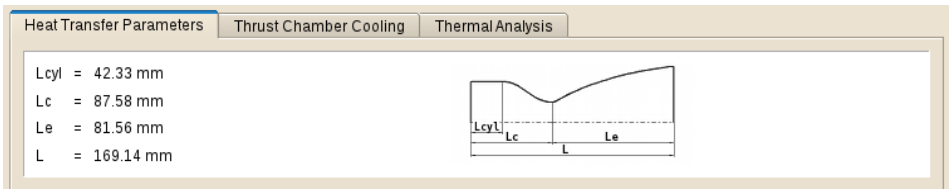
Thrust Chamber Cooling

On the tab *Thrust Chamber Cooling* you can define design parameters of thrust chamber cooling.



Thrust chamber cooling

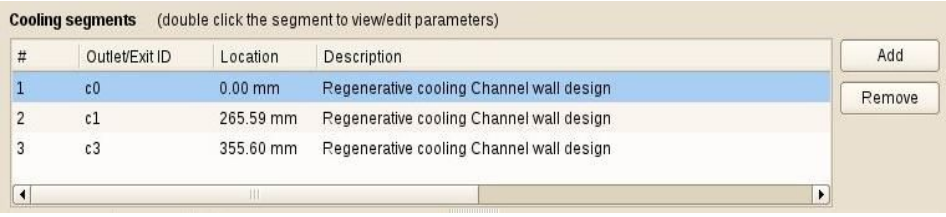
In order to make easier the definition of location of cooling sections or film slots, the program displays the dimensions of thrust chamber:



Note that the dimensions are only available after the first run of the program.

The table *Cooling segments* displays all cooling segments configured for specific thrust

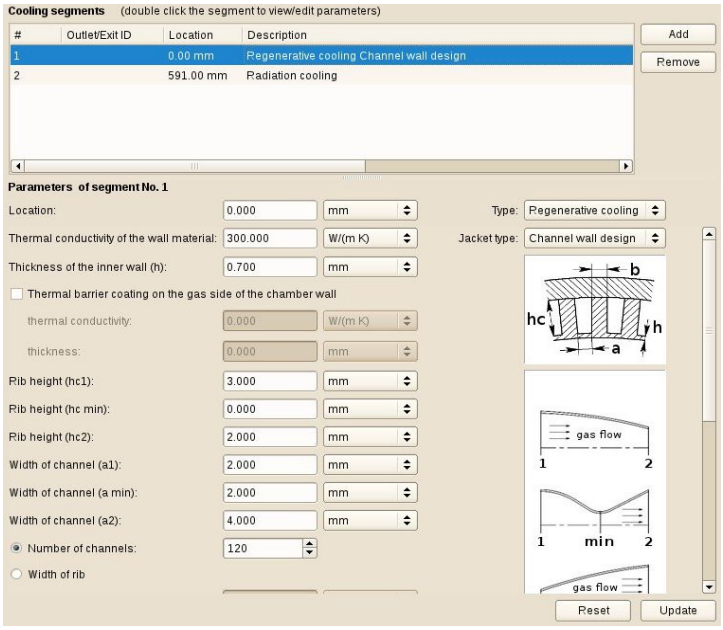
chamber, as well as its ID (if available), location and short description:



The user can add new segment pressing the button **Add** or delete existing segment pressing the button **Remove**.

Note: all new cooling segments are created as “Radiation cooling”. If you need another type of cooling, double click the created segment and change the type in the parameters area.

To view all parameters assigned to of the existing cooling segment, or to change the type of the segment or modify its parameters, double-click the segment on the list. All parameters of the segment will be displayed in the parameters area just below the table:



The number of currently edited segment is displayed at the bottom of the parameter area.

Use the button **Update** in order to re-assign the modified parameters to edited segment.

Use the button **Reset** in order to reload the parameters from the segment. Note that you will loss all changes in the parameters area.

The user has to define the location of the each cooling segment or film cooling, specifying the distance from the injector plate. The length of the segment is defined implicitly by the location of the next cooling segment or nozzle exit (if no further segment is available).

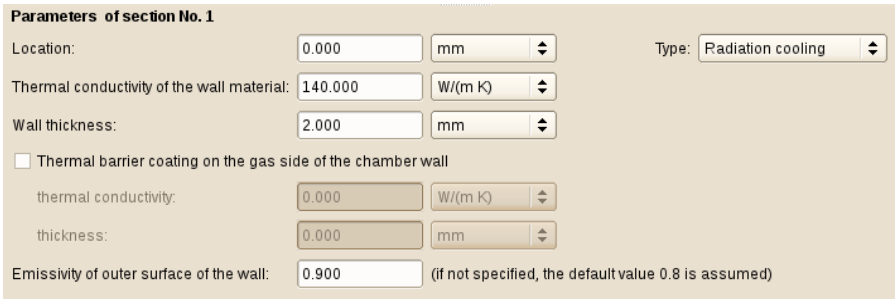
The program supports 3 types of chamber cooling: radiation cooling, regenerative cooling and film cooling.

Radiation Cooling

A radiation cooled chamber transmits the heat from its outer surface, chamber or nozzle extension. Radiation cooling is typically used for small thrust chambers with a high-temperature wall material and/or in low-heat flux regions, such as a nozzle extension.

Configuration of the radiation cooling section includes the following parameters:

- location of the section — required
- wall thickness and thermal conductivity — required
- thermal barrier coating layer thickness and thermal conductivity (see also Thermal Barrier Coating Layer) — optional
- radiation emissivity coefficient of the wall outer surface — required



Radiation cooling parameters

Radiation cooling segment can be combined with the film cooling.

Note: the current version of the program may fail to calculate the thermal state of radiation cooling segment with additional film cooling, if calculation of the radiation heat transfer is disabled (see screen "*Heat Transfer Parameters*").

Regenerative Cooling

Regenerative cooling is the most widely used method of cooling a thrust chamber and is accomplished by flowing high-velocity coolant over the back side of the chamber hot gas wall to convectively cool the hot liner. The coolant with the heat input from cooling the liner is then usually discharged into the injector and utilized as a propellant.

Configuration of the regenerative cooling segment includes the following parameters:

- location of the section — required
- type of design — required
- section ID — required if the exit from the segment connected with input of another regenerative cooling

	segment
• wall thickness and thermal conductivity	required
• thermal barrier coating layer thickness and thermal conductivity (see also *Thermal Barrier Coating Layer*)	optional
• coolant definition: component/s, initial temperature and pressure, relative mass flow rate *or* connection with exit of another regenerative cooling segment, specified by its ID	required
• flag of opposite flow direction	optional

Regenerative cooling segment can be combined with the film cooling.

The program supports 3 types of regenerative cooling design: coaxial shell jacket design, tubular wall jacket design and channel wall jacket design.

Coolant definition

The list of coolant components contains one or more species, displayed on the single row. To add species to the list, click the button **Add** at the bottom of the corresponding list. To remove the selected species from the list, click the button **Remove**.

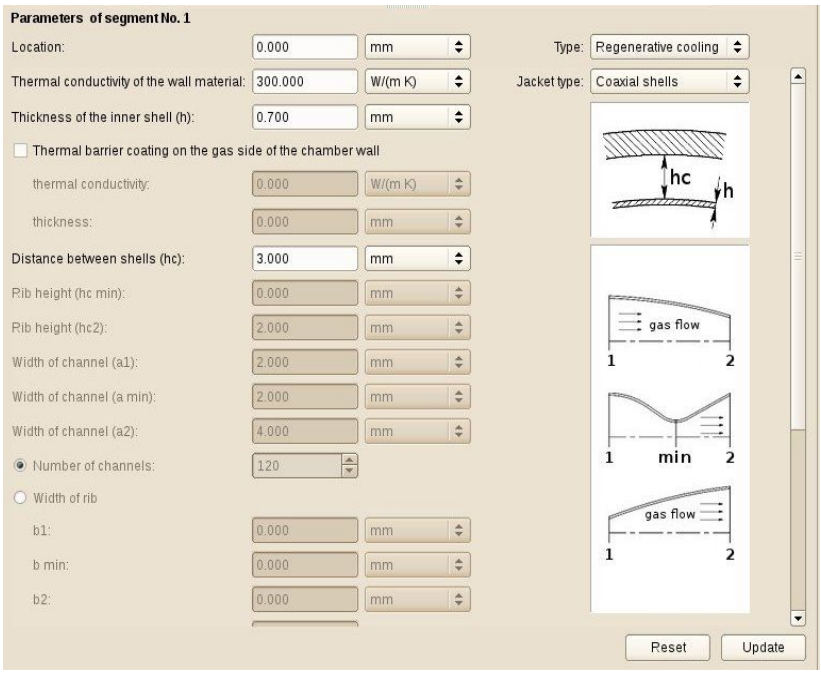
Note: only use the components available in the properties, because the extended NASA database does not include all required properties such as viscosity, thermal conductivity, density, temperature of vaporization/decomposition and heat of vaporization of liquid components.

Alternatively, you can configure the connection between the inlet of the current segment with the exit of another segment, specifying its ID. In this case the coolant component/s and the mass flow rate are exactly the same as defined for the first segment, whereas the initial temperature and pressure are equal to temperature and pressure at exit of the first segment, calculated during the thermal analysis.

Coaxial Shell Jacket Design

Design-specific configuration of the segment includes the following parameters:

• distance between inner and outer walls	required



Coaxial shell jacket parameters

The program supports the segments with constant distance between the walls. If you need to define the chamber with variable inter-wall distance, you have to define several segments each with different distance.

Typical mistakes during the analysis run:

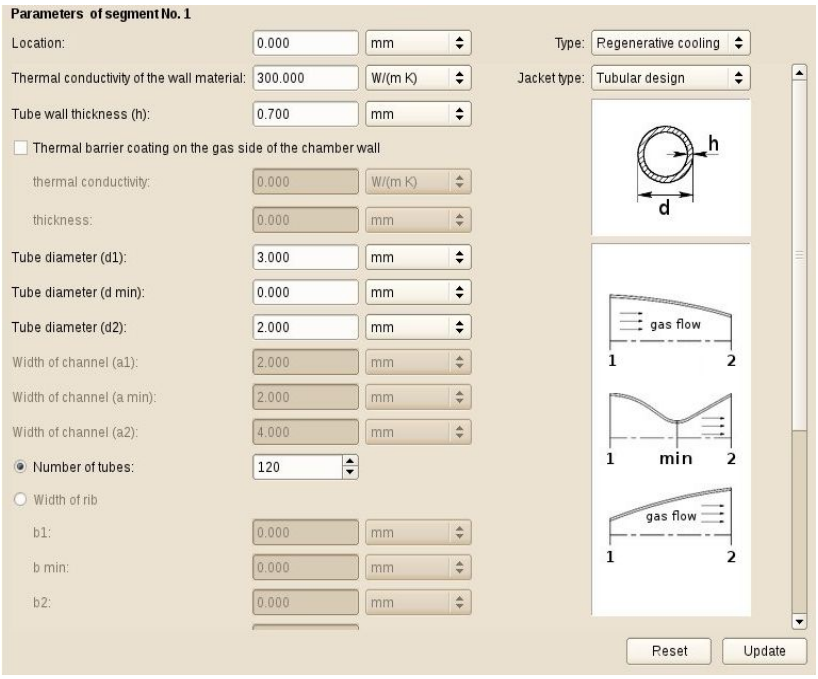
- too small inter-wall distance or too large coolant mass flow rate lead to high pressure drop; the message "Negative pressure drop" appears;
- due to too large inter-wall distance or too small mass flow rate the program cannot converge the solution.

Possible solution: change one or several relevant parameters.

Tubular Wall Jacket Design

Design-specific configuration of the segment includes the following parameters:

• diameter of tube “*d1”*, “*d2”* at both sides of the segment	required
• diameter of tube “*d min”* at minimal diameter of the segment	optional; can be defined if nozzle throat is located within current segment
• tube wall thickness “*h”* and thermal conductivity	required
• number of tubes	required
• flag of helical tube wall jacket design	required



Tube wall jacket parameters

The program supports the segments with constant number and variable diameter of the tubes. If you need to define the chamber with variable number of tubes, you have to define several segments each with different corresponding parameters.

Typical mistakes during the analysis run:

- inappropriate number or diameter of tubes, or too high coolant mass flow rate lead to high pressure drop; the message "Negative pressure drop" appears;
- due to inappropriate number or diameter of tubes, or too small mass flow rate the program cannot converge the solution.

Possible solution: change one or several relevant parameters or replace the single cooling segment with two or more segments with optimum parameters.

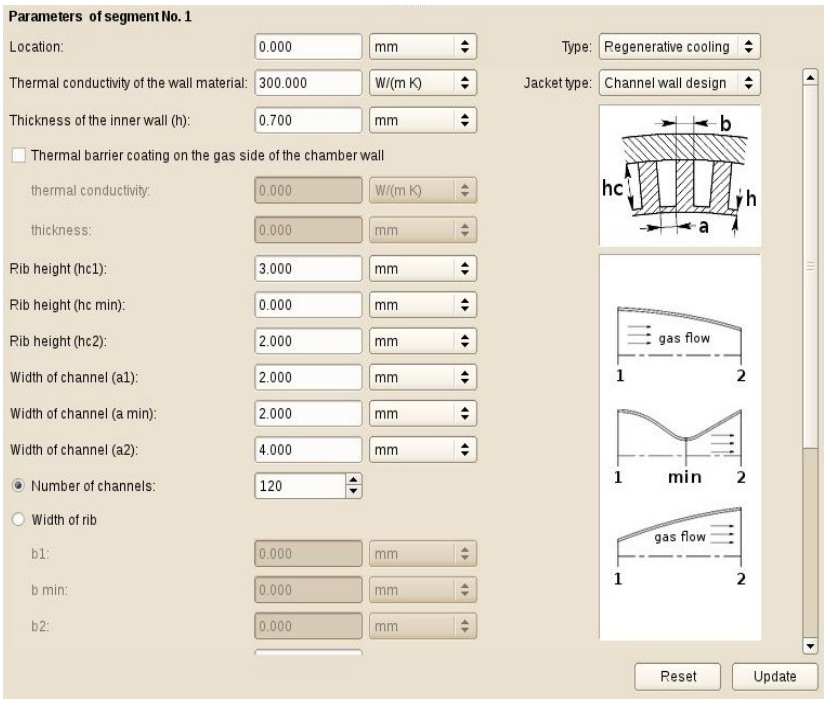
Channel Wall Jacket Design

Design-specific configuration of the segment includes the following parameters:

- height of ribs “*hc1”*, “*hc2”* at both sides of the segment — required
- height of ribs “*hc min”* at minimal diameter of the segment — optional; can be defined if nozzle throat is located within current segment
- width of channels “*a1”*, “*a2”* at both sides of the segment — required
- width of channels “*a min”* at minimal diameter of the segment — optional; can be defined

	if nozzle throat is located within current segment
• number of channels “*n”* *or* width of ribs “*b1”*, “*b2”* at both sides of the segment	required
• width of ribs “*b min”* at minimal diameter of the segment	optional; can be defined if nozzle throat is located within current segment
• angle between channels and generatix	optional

Note: index "1" corresponds to the parameters at the section end which is closer to injector plate; index "2" corresponds to the parameters at the section end which is closer to nozzle exit.



Channel wall jacket parameters

The program supports the segments with constant number of channels and angle between channels and generatix. If you need to define the chamber with variable number of channels and/or angle, you have to define several segments each with different corresponding parameters.

Typical mistakes during analysis run:

- inappropriate geometry and/or number of channels, or too high coolant mass flow rate lead to high pressure drop; the message "Negative pressure drop" appears;

- due to inappropriate geometry and/or number of channels, or too small mass flow rate the program cannot converge the solution.

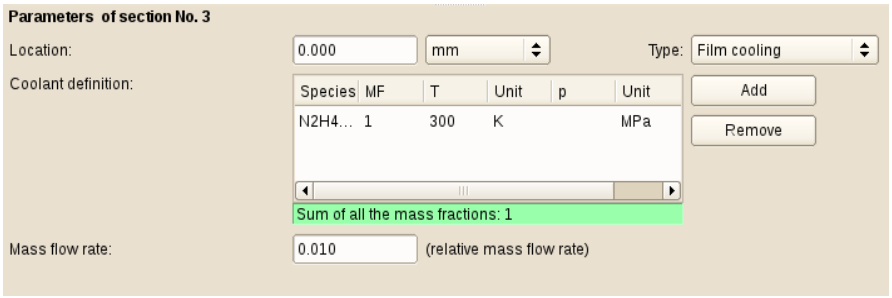
Possible solution: change one or several relevant parameters or replace the single cooling segment with two or more segments with optimum parameters.

Film Cooling

Film cooling provides protection from excessive heat by introducing a thin film of coolant through orifices around the injector periphery or in the chamber wall. This method is typically used in high heat flux regions in combination with regenerative or radiation cooling.

Configuration of the film cooling includes the following parameters:

- location (the distance from the injector plate) required
- coolant definition: required
  component/s,
  initial temperature,
  relative mass flow rate



Film cooling parameters

You can specify either liquid or gaseous species to be used as a film coolant.

Thrust chamber with several cooling segments

You can define more than one cooling segments

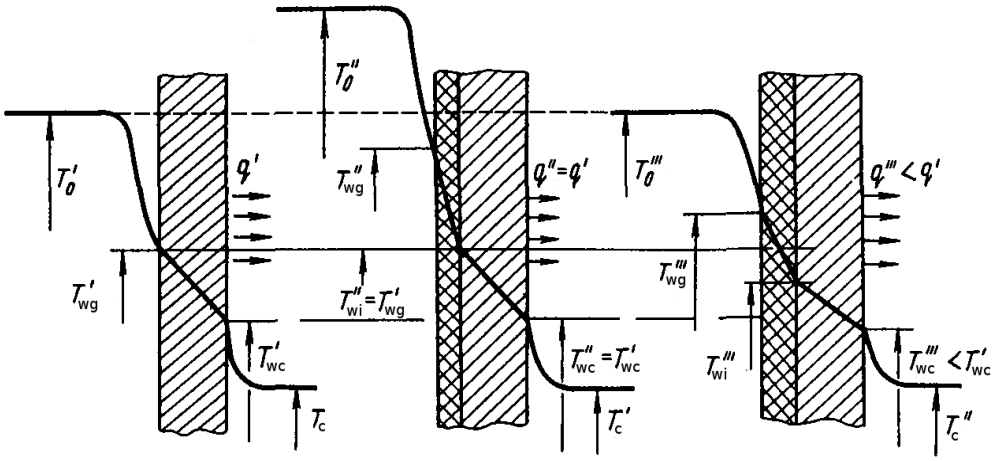
- to combine different types of cooling (e.g. regeneratively cooled combustion chamber + radiatively cooled nozzle extension + one or more film cooling);
- to optimize parameters along the chamber (e.g. number of tubes/channels, tube/channel geometry etc).

Thermal Barrier Coating Layer

The thermal barrier coating (TBC) is used to reduce the temperature gradient across the chamber wall.

The functional principle of a TBC, illustrated in figure below, is to increase the hot gas side wall temperature T_{wg}. This is achieved by applying a ceramic top layer (e.g. Y2O3-stabilized

ZrO2, PYSZ) onto the wall base material. The PYSZ ceramic offers a thermal conductivity λ of about 1.5 W/(m K). In order to prevent a coating overheat (>1500 K) coating thicknesses of less than 0.1 mm are required. These high hot gas side wall temperatures compared to the maximum temperatures of the uncoated wall lead to the remarkable reduction in heat flux.



Functional principle of a TBC

With such a protective layer, the application range of existing regenerative or radiation cooling thrust chambers can be enlarged towards higher combustion chamber pressures or an increased thrust chamber life.

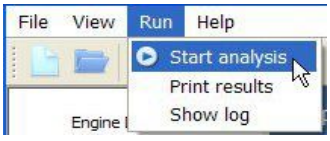
In general a TBC acts as an insulator to the inner liner, reducing the wall heat flux into the wall base material.

If you need to analyze the thermal state of the thrust chambers with TBC consisting of several layers with known thicknesses $\delta_{1,}\delta_{2,}\ldots,\delta_N$ and thermal conductivities $\lambda_{1,}\lambda_{2,}\ldots,\lambda_N$, you may calculate the effective thermal conductivity λ of the TBC with the thickness $\delta_1+\delta_2+\ldots+\delta_N$ from the following relation:

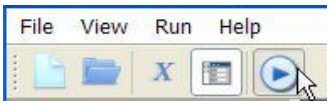
$$\lambda = \frac{\delta_1+\delta_2+\ldots\delta_N}{\frac{\delta_1}{\lambda_1}+\frac{\delta_2}{\lambda_2}+\ldots+\frac{\delta_N}{\lambda_N}}$$

Thermal Analysis

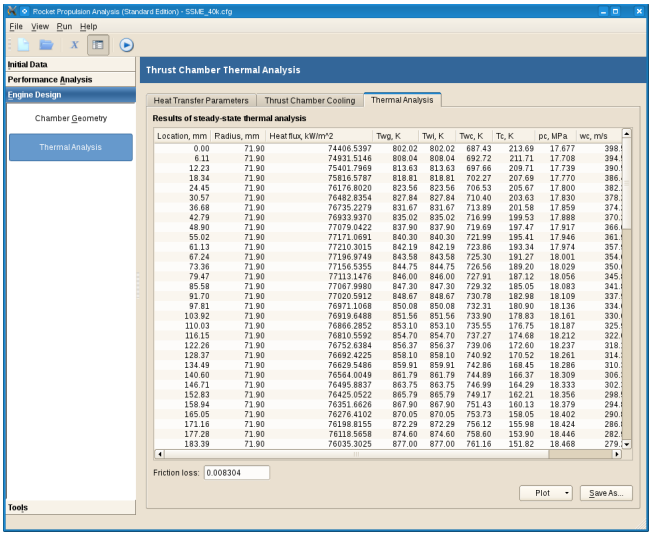
After specifying all required parameters, you can start the analysis by clicking menu **Run**, and then **Start analysis** in menu bar:



or clicking icon **Start** on the main toolbar:



After successful finishing the analysis the program automatically switches to the screen Thermal Analysis, that displays the calculated results.



Thermal analysis results

The thrust chamber geometry is printed out in columns "Location" and "Radius".

The column "Heat flux" displays the calculated heat flux (depending on heat transfer parameters, either the sum of convection heat flux and radiation heat flux or convection heat flux only) at the corresponding location.

The column "T_{wg}" displays the temperature of chamber wall on its hot gas side.

The column "T_{wi}" displays the temperature between the thermal barrier coating layer and chamber wall (if coating is available) or the temperature of chamber wall on its hot gas side (the same as T_{wg}).

The column "T_{wc}" displays the temperature of chamber wall on its cooler side.

The columns "p_c", "T_c", "wc", and "rhoc" display the pressure, temperature, velocity, and density of the coolant correspondingly (if applicable).

Using context menu, you can copy the content of the complete table or single selected row into the clipboard.

Click the button "Save As..." at the right-bottom corner of the screen in order to save the results as ASCII or HTML file.

You can plot the diagrams "Heat Flux vs. Location" or "Temperatures vs. Location", clicking the button Plot at the bottom-right corner of the tab. The location of nozzle throat is shown on the diagram by corresponded vertical marker line.

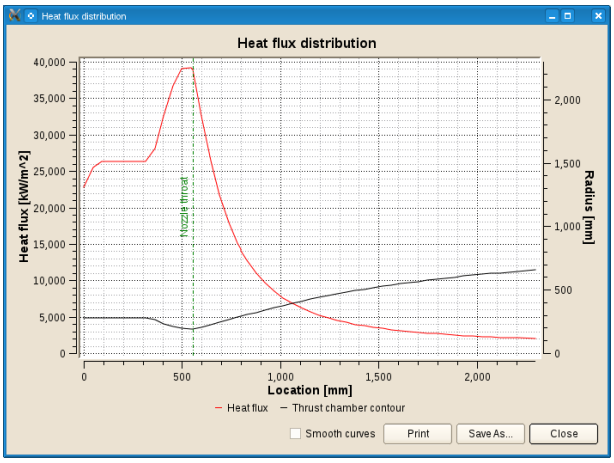


Diagram „Heat flux vs. Location“

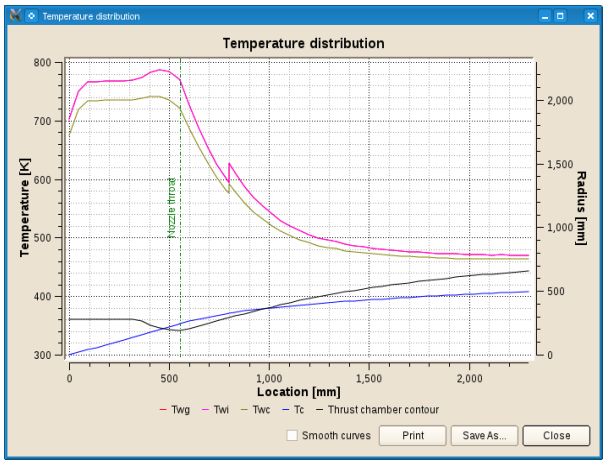


Diagram „Temperatures vs. Location“

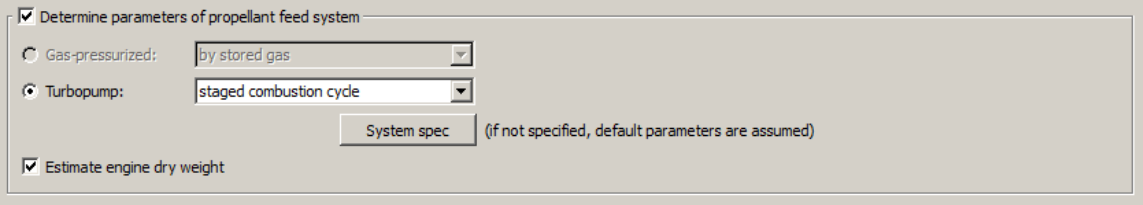
All results are displayed in the current units, defined in the Preferences Dialog.

Propellant Feed System (Cycle Analysis)

The program is capable of performing the liquid-propellant engine cycle analysis.

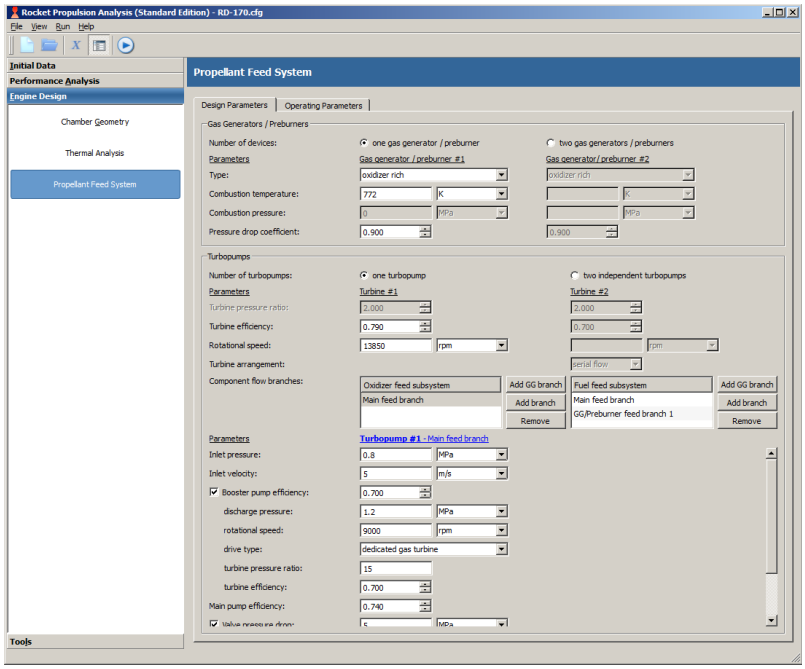
The screen Propellant Feed System consists of 2 tabs *Design Parameters* and *Operating Parameters*.

The input of parameters on the screen *Design Parameters* are only enabled if both flags "Determine thrust chamber size" and "Determine parameters of propellant feed system" on the screen Engine Definition are switched on, and the type of engine cycle is provided:



Design Parameters

On the tab *Design Parameters* user can define design parameters of the propellant feed system.



Propellant feed system design parameters

The program supports analysis of following engine cycles:

- Gas generator cycle
- Staged combustion cycle
- Full flow staged combustion cycle

For selected cycle, the following design parameters can be defined:

• Number of gas generators/preburners: one combustion device two combustion devices	required
• Type of gas generators/preburners: oxidizer-rich fuel-rich In staged combination cycle with two preburners, the type of both combustion devices has to be identical. In full flow staged combination cycle, the type of combustion devices has to be different.	required
• Combustion temperature The value is limited by heat resistance and high-temperature strength of used construction materials. Typical values for oxidizer-rich devices are 650-850 K Typical values for fuel-rich devices are 800-1100 K	required
• Combustion pressure Typical values for gas generator cycle: $p = (0.8...0.85)p_c$	required for gas generator cycle

For staged combustion cycles the parameter is disabled.	
• Pressure drop coefficient The value defines the pressure drop between injector of the combustion device and turbine inlet. Typical values are 0.9-0.95	required
• Number of independent turbopumps: one turbopump two independent turbopumps If two combustion devices are configured, the only selection possibly is two independent turbopumps	required

For each independent turbopump, the following design parameters can be defined:

• Turbine pressure ratio	required for gas generator cycle
• Turbine efficiency Typical values are 0.5-0.8	required
• Turbine rotational speed Used for estimation of engine dry weight.	required if flag *Estimate engine dry weight* is switched on
• Turbine arrangement: serial flow parallel flow	required for second turbine in configurations with one combustion device
• Main feed branch Defines parameters of component feed branch between engine inlet and • thrust chamber in gas generator cycle • preburner in staged combustion cycles	required
• GG/preburner feed branch Defines parameters of component feed branch between main pump outlet and combustion device.	0 to 2 branches depending on cycle diagram
• Additional branches Defines parameters of additional branch.	optional

For each main feed branch, the following design parameters can be defined:

• Inlet component pressure Typical values are 0.2-0.8 MPa and depend on tank conditions	required
• Inlet component velocity	required

Typical values are 4-10 m/s, for liquid hydrogen 10-20 m/s	
• Main pump efficiency Typical values are 0.6-0.8 For single-shaft turbopumps typical values are: 0.7-0.8 for oxidizer pump 0.55-0.7 for fuel pump	required
• Booster pump efficiency Typical values are 0.5-0.7	optional
• Booster pump discharge pressure	required if booster pump is configured
• Booster pump rotational speed Used for estimation of engine dry weight.	required if flag *Estimate engine dry weight* is switched on and booster pump is configured
• Booster pump drive type: dedicated hydraulic turbine dedicated gas turbine main gas turbine	required if booster pump is configured
• Booster turbine pressure ratio Typical values are 1.1-1.2 for hydraulic turbine, and 5-15 for gas turbine	required if booster pump with dedicated turbine is configured
• Booster turbine efficiency Typical values are 0.5-0.7	required if booster pump with dedicated turbine is configured
• Valve pressure drop Typical values: $\Delta p = (0.1...0.2) p_c$	optional
• Cooling jacket pressure drop Typical values: $\Delta p = (0.25...0.35) p_c$, where lower coefficient is selected for $p_c < 6\,MPa$ and upper coefficient is selected for $p_c > 8\,MPa$	optional
• Cooling jacket temperature raise	optional (not used in current version)
• Injector pressure drop Typical values: $\Delta p = (0.4...0.8)\sqrt{p_c}$ or $\Delta p = 0.3...1.5\,MPa$	optional

For each gas generator/preburner feed branch, the following design parameters can be defined:

• Assigned relative mass flow rate	Optional (not used in current version)
• Inlet relative cross-section area. Defines the inlet cross-section area of the branch relative to the inlet cross-section area the main branch. Typical values are 0.1-0.3	required
• Kick pump efficiency Typical values are 0.3-0.5 If no kick pump is defined, the pump of the main branch has to produces such a discharge pressure that it fits to inlet of gas generator/preburner. Usually no kick pump is required for gas generator cycle, whereas an absence of kick pump in staged combustion cycle leads to significant pressure raise.	optional
• Valve pressure drop Typical values: $\Delta p = (0.1...0.15) p_c$	optional
• Cooling jacket pressure drop Defines pressure drop in cooling jackets of additional devices (e.g. gas ducts, preburners, etc.) except cooling jackets of the thrust chambers.	optional
• Fixed pressure drop Defines additional pressure drops between main pump outlet and combustion device except pressure drops in valve, cooling jacket and injector.	optional
• Injector pressure drop Typical values: $\Delta p = (0.4...0.8) \sqrt{p_c}$	optional

For each optional feed branch, the following design parameters can be defined:

• Name	required
• Relative mass flow rate Defines the mass flow rate through the branch relative to the total mass flow rate at the inlet of the main branch.	required
• Inlet relative cross-section area Defines the inlet cross-section area of the branch relative to the inlet cross-section area the main branch. Typical values are 0.1-0.3	required
• Kick pump efficiency Typical values are 0.3-0.5 If no kick pump is defined, the pump of the main branch has to produces such a discharge pressure that it fits to inlet of gas generator/preburner. Usually no kick pump is required for	optional

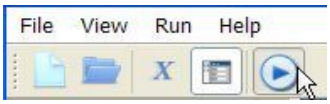
gas generator cycle, whereas an absence of kick pump in staged combustion cycle leads to significant pressure raise.	
• Valve pressure drop Typical values: $\Delta p = (0.1...0.15) p_c$	optional
• Cooling jacket pressure drop. Defines pressure drop in cooling jackets. If the branch is used to feed the coolant for the thrust chambers, the typical values: $\Delta p = (0.25...0.35) p_c$, where lower coefficient is selected for $p_c < 6\ MPa$ and upper coefficient is selected for $p_c > 8\ MPa$	optional
• Fixed pressure drop Defines additional pressure drops between main pump outlet and combustion device except pressure drops in valve and cooling jacket.	optional
• Discharge pressure Defines discharge pressure from the branch.	required

Subsystems of Propellant Feed System

Running Cycle Analysis

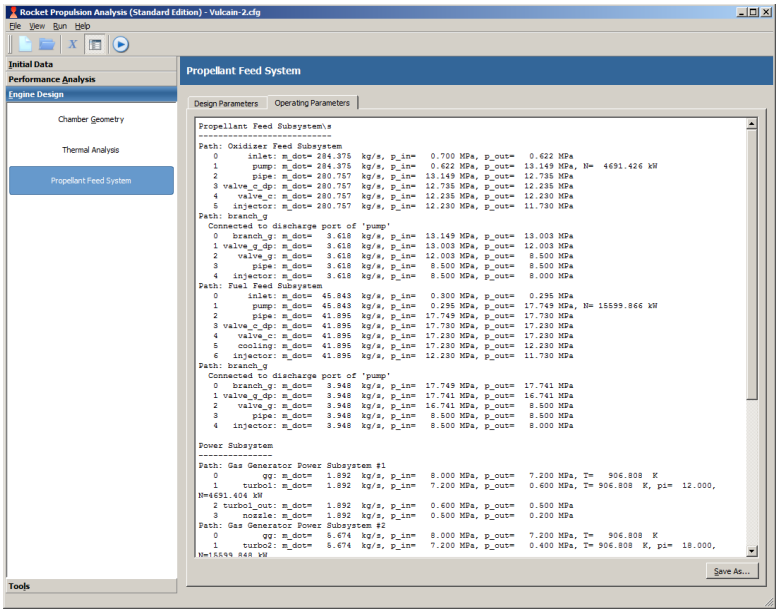
After specifying all required parameters, you can start the analysis by clicking menu **Run**, and then **Start analysis** in menu bar:

or clicking icon **Start** on the main toolbar:



Operating Parameters

After successful finishing the analysis the program automatically switches to the screen Operating Parameters, that displays the calculated results.



Cycle analysis results

Cycle Performance

Estimated engine performance is printed out below the operating parameters of the feed system:

```
Engine performance
------------------
Chamber performance
   specific impulse (vac):    429.51 s
   specific impulse (opt):    412.88 s
Engine performance
   specific impulse (vac):    419.66 s
   specific impulse (opt):    403.03 s
        correction factor:      0.98
```

For staged combustion cycle, the engine performance is identical to chamber performance.

Engine Dry Weight

If the flag *Estimate engine dry* weight was switched on, the estimated engine weight is printed out at the bottom of the screen *Operating Parameters*:

```
Engine dry weight estimation
----------------------------
                 chamber/s:    736.90 kg
              turbopump/s:    457.90 kg
      booster turbopump/s:      0.00 kg
         other components:    625.29 kg
                    total:   1820.08 kg
```

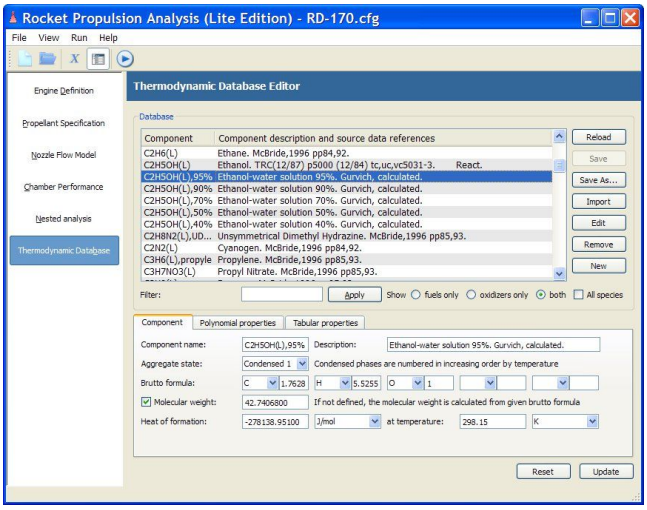
Thermodynamic Database Editor

Thermodynamic Database Editor is an embedded species viewer/editor. Using the tool, you can easily define new propellant components, or import components from *PROPEP* or *CEA2* species databases.

RPA distribution packages contain two database files `resources/thermo.inp` and `resources/properties.inp`. The file `resources/thermo.inp` contains the thermodynamic properties in format described in reference http://www.grc.nasa.gov/WWW/CEAWeb/def_formats.htm.

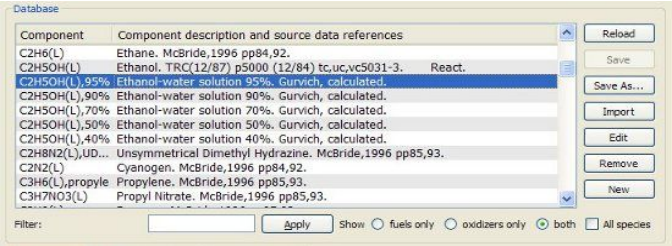
You can define your own species and save it into two additional database files `resources/usr_thermo.inp` and `resources/usr_properties.inp`. These files are not shipped within standard RPA distribution packages, and will not be rewritten after program update.

The tool consists of database viewer at the top of the screen, and species editor at the bottom. You can shrink or heighten the species viewer while the species editor will be heightened or shrunk, dragging the horizontal bar between the viewer and the editor down or up.



Thermodynamic Database Editor

The viewer features the species table, filter and the control buttons. Species table displays currently available species with respect to the filter settings:



You can force the viewer to display the fuels only, or oxidizers only, or both fuels and oxidizers, marking corresponding radio buttons. Mark the check box *"All species"* if you want to see all species, including atomized and/or ionized products of reaction, or keep it unmarked if you want to see only possible propellant components.

The filter pattern is applied to both columns of the table.

The control buttons can be used as follows:

- Click the button **Reload** to reload the default database. Any changes made since the database was last saved will be lost.
- Click the button **Save** to save the changes made since the database was last saved. The program creates a backup-copy of previous version of the database files, adding the character "~" to the name of the file.
- Click the button **Save As...** to save the current database in specified location.
- Click the button **Import** to import the species from external database file in *CEA2* or *PROPEP* formats. After successful loading the external database, the program displays the list of available species is the dialog window. Select one or more species that you want to import and click the button **OK**. All imported species are immediately available for thermodynamic analysis.
- Click the button **Edit** to load the data of the selected species into the species editor. You can also double click the species on the list to load it into the editor.
- Click the button **Remove** to remove the species from the current database. Note that removed species are immediately unavailable for thermodynamic analysis.
- Click the button **New** to reset the editor for creating a new species.

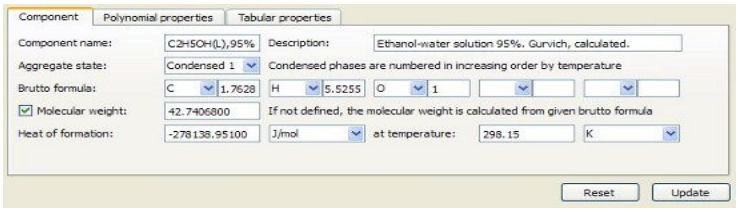
Note: all new species are saved into the user-defined database files `resources/usr_thermo.inp` and `resources/usr_properties.inp`.

Note: although you can import any component from *PROPEP* specues database, do not replace all components already available in *CEA2/RPA* database: the sources of the data in *CEA2* file are NASA Glen thermodynamic database and Gurvich thermodynamic database, both known for their high accuracy.

Note: since *PROPEP* library does not contain the component's temperature, always check standard temperature and tabular data for imported components.

Note: always check the log (click item **Run** in main menu, and then **Show log**; check the tabs *"Warnings"* and/or *"Errors"*) just after the import from *PROPEP* library.

The editor consists of three tabs *Component*, *Polynomial properties*, and *Tabular properties*, and the control buttons at the bottom of the editor:



To save the changed in existing species or save new species, click the button **Update**. To reset the species data, click the button **Reset**.

The tab *Component* displays the information about component, its aggregate state, chemical formula, molecular weight, heat of formation, and the temperature the heat of formation is defined for.

The component name is also an identifier of the species and must be unique within the database. The suffix (L) can be added to the end of the name for the liquid components.

The description usually contains common name of the species, as well as the reference information.

The chemical formula is given as a molecular formula (if applicable), or an exploded formula, followed by its molecular weight. The heat of formation (enthalpy) can be given in one one the units: `J/mol`, `cal/mol`, `kJ/kg`, `kcal/kg`, `Btu/lbm`, `kcal/lbm`. The heat of formation is followed by the temperature (given in one of the units: `K`, `R`, `C`, `F`), for which it has been defined. Note that if polynomial properties are available, the temperature is always `298.15 K` and cannot be changed.

Polynomial properties for the one or more temperature interval are given by 9 coefficients as described in reference http://www.grc.nasa.gov/WWW/CEAWeb/def_formats.htm. Click the button **Add** to add new temperature interval; click the button **Remove** to delete selected temperature interval.

Tabular properties for the one or more pressure and temperature intervals are given by values of specific heat `Cp` (`kJ/mol-K`), density `rho` (`kg/m`^{3}) and dynamic viscosity `mu` (`muPa-s`) for each unique combination of pressure and temperature. Click the button **Add p** to add new pressure interval; click the button **Remove p** to delete selected pressure interval. Click the button **Add T** to add new temperature interval; click the button **Remove T** to delete selected temperature interval.

Note: For the components which are supplied together with thermodynamic properties in the polynomial form, you do not need to define the specific heat (define "0" instead).

Note: For the gaseous components you do not need to define the density.

In the database file, the tabular data are formatted as follows:

```
!p, MPa      T, K        Cp, kJ/mol-K  rho, kg/m3    mu, muPa-s
Comp_name       2,3
p1          T1          Cp11        rho11       mu11
p1          T2          Cp12        rho12       mu12
```

```
p1        T3        Cp13      rho13     mu13
p2        T1        Cp21      rho21     mu21
p2        T2        Cp22      rho22     mu22
p2        T3        Cp23      rho23     mu23
```

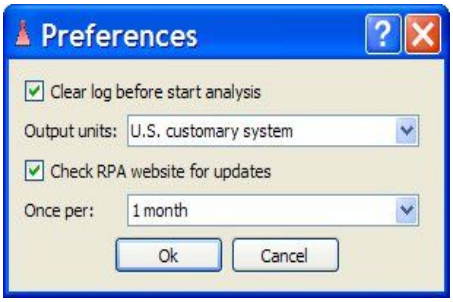
The minimalist data for the component consists of at least two rows:

```
!p, MPa      T, K        Cp, kJ/mol-K  rho, kg/m3    mu, muPa-s
Comp_name       1,2
 0.101325    273.15       0.078       823.0
 0.101325    373.15       0.078       823.0
```

This data defines the constant specific heat `Cp` and constant density rho, and allows to specify the initial temperature in the range `273.15-373.15 K` as well as the initial pressure in the range 0...(the-max-pressure-you-need). Viscosity is not defined (as it will only be required by RPA Standard Edition) and assumed to be equal 0.

Preferences

Dialog window Preferences can be used to set up global configuration parameters. Click the item *Help*, and then *Preferences* in main bar to open the window:



Dialog window "Preferences"

Mark the check box *"Clear log before start analysis"* to force the cleaning the analysis log before each run, or leave it unmarked to let the log accumulate the messages from all runs.

You can define the default output units that will be used by the program to display the results of analysis, selecting either Metric system (SI) or U.S. Customary system on the list Output units.

If selected Metric system (SI), the following units will be used:

Parameter	**Unit**
Temperature	`K (of reaction products)`
Temperature	`K (of propellant components)`
Pressure	`MPa`

Parameter	Unit
Specific impulse	m/s
Velocity	m/s
Mass flow rate	kg/s
Mass flux	kg/m^2-s
Thrust	N, kN
Density	kg/m^3
Enthalpy	kJ/kg, J/mol
Entropy, Specific heat, Gas constant	kJ/kg-K, J/mol-K

If selected U.S. Customary system, the following units will be used:

Parameter	Unit
Temperature	F (of reaction products)
Temperature	R (of propellant components)
Pressure	psi
Specific impulse	ft/s
Velocity	ft/s
Mass flow rate	lbm/s
Mass flux	lbm/ft^2-s
Thrust	lbf
Density	lbm/ft^3
Enthalpy	Btu/lbm, Btu/lb-mol
Entropy, Specific heat, Gas constant	Btu/lbm-R, Btu/lb-mol-R

Mark the check box *"Check RPA website for updates"* to force the program to check for available updates, or leave it unmarked to disable this function. You can also define how often should the program perform the check, selecting one of the following items on the list *"Once per"*:

- RPA start-up
- 1 day, 1 week
- 1 month

In order to force the program to check for available updates, click the item **Help**, and then **Check for Updates** in main bar.

Input and Output Units

You can enter the values of input parameters in any available units, freely mixing Metric (SI) and U.S. Customary systems. The program automatically converts all entered values to the Metric system, which is standard internal representation of both input parameters and results of calculation.

The following conversion factors are used to convert values in non-SI units to values in SI units:

Name	Value	Description
CONST_ATM	101325.0	Conversion factor from atm to Pa
CONST_AT	98066.5	Conversion factor from at (technical atmosphere) to Pa
CONST_BAR	100000.0	Conversion factor from bar to Pa
CONST_PSI	6894.75729316836	Conversion factor from psi (pound-force per square inch) to Pa
CONST_T0	298.15 K	Temperature 25 C
CONST_R0	8.314472 J/(mol·K)	Universal Gas Constant
CONST_G	9.80665 m/s^2	
CONST_POUND	0.45359237	Conversion factor from lbm (pound mass) to kg
CONST_POUND_FORCE	(CONST_POUND*CONST_G)	Conversion factor from lbf (pound-force) to N (newton)
CONST_FOOT	0.3048	Conversion factor from international foot to m
CONST_INCH	0.0254	Conversion factor from inch to m
CONST_MILE	(5280.0*CONST_FOOT)	Conversion factor from international mile to m
CONST_LBM_FOOT3	(CONST_POUND/$CONST\_FOOT^3$)	Conversion factor from "lbm/ft^3" to "kg/m^3
CONST_LBM_INCH3	(CONST_POUND/$CONST\_INCH^3$)	Conversion factor from "$lbm/inch^3$" to "kg/m^3"
CONST_MASS_FLUX	(CONST_POUND/$CONST\_FOOT^2$)	Conversion factor from "$lbm/(ft^2 \cdot s)$" to "$kg/(m^2 \cdot s)$"
CONST_LBM_MOLE	(1000.*CONST_LBM)	Conversion factor from lb-mole to mole
CONST_BTU	1055.05585262	Conversion factor from Btu to J
CONST_CAL	4.1868	Conversion factor from calorie to J
CONST_BTU_LBM	(CONST_BTU/CONST_LBM)	Conversion factor from Btu/lbm to J/kg
CONST_BTU_LBM_MOLE	(CONST_BTU/CONST_LBM_MOLE)	Conversion factor from Btu/lb-mol to J/mol

Name	Value	Description
CONST_BTU_LBM_R	(1000.*CONST_CAL)	Conversion factor from Btu/(lbm·R) to J/(kg·K)
CONST_BTU_LBM_F	CONST_BTU_LBM_R	Conversion factor from Btu/(lbm·F) to J/(kg·K)
CONST_BTU_LBM_MOLE_R	CONST_BTU_LBM_R	Conversion factor from Btu/(lb-mol·R) to J/(mol·K)

The Metric system is also used by default to display the results of calculation. You can change it to U.S. Customary system, using dialog window Preferences.

References:

- SP-811. NIST Guide for the Use of the International System of Units (SI). B.8 Factors for Units Listed Alphabetically
- Glossary of terms and table of conversion factors used in design of chemical propulsion systems. NASA SP-8126. 1979.
- George P. Sutton, Oscar Biblarz. Rocket Propulsion Elements, 7th Edition (pp.727-729).
- NASA-STD-3000. Man-Systems Integration Standards. Volume II. Appendix E - Units of Measure and Conversion Factors

Scripting Utility

The scripting language provided is based on the ECMAScript scripting language, as defined in standard ECMA-262.

Scripting utility implements binding to many internal functions of RPA, so that the user can program and run the following tasks:

- load, manipulate and write configuration files
- search the thermodynamic database by species name
- get thermodynamic properties of the species
- prepare mono- bi- and multi-propellant compositions
- run typical combustion problems (p,H)=const, (p,S)=const, (p,T)=const
- run typical rocket propulsion problems
- use in a custom JavaScript program all features listed above

Scripting utility can be started in either an interactive mode or a batch mode.

API Reference

Besides standard ECMA-262 API, RPA Scripting Utility provides the following APIs:

API	Description
Generic functions	Generic built-in functions and objects
Configuration API	API for loading, manipulation and writing configuration files.
Thermo API	API for searching the thermodinamic database, obtaining species properties, preparing mono- bi- and multipropellant compositions.
Reaction API	API for running typical combustion problems (p,H)=const, (p,S)=const, (p,T)=const, obtaining thermodinamic properties of the reaction products.
Performance API	API for running typical rocket propulsion problems and obtaining performance parameters.

Generic built-In Functions

Function	Description
`load(`*`path_to_script`*`)`	Load external script defined by it's path. Example: load("resources/scripts/config.js");

`print(`*`parameter`*`)`	Print out the parameter in console and log file.
	
	Examples:
	
	print("Starting analysis...");
	print("Is_v=" + pr.getNozzleExitSection().getIs_v().toPrecision(7));

`sprintf(format,` *`parameters`*`)`	The function is a limited Javascript implementation of sprintf, originated from the C programming language. Function returns the string with passed arguments formatted by the usual sprintf conventions. Possible format values: %% – returns a percent sign %b – binary number %c – the character according to the ASCII value %d – signed decimal number %f – floating-point number %o – octal number %s – string %x – hexadecimal number (lowercase letters) %X – hexadecimal number (uppercase letters) Additional format values, placed between the % and the letter (example %.2f): + – forces both + and - in front of numbers. By default, only negative numbers are marked - – left-justifies the variable value 0 – zero will be used for padding the results to the right string size [0-9] – specifies the minimum width held of to the variable value .[0-9] – specifies the number of decimal digits or maximum string length

Function	Description
	Example: var s = sprintf("Isp (vac)=%8.4f Isp (SL)=%8.4f s\n", Is_v, Is_SL);
`printf(format, parameters)`	The function is a limited Javascript implementation of printf, originated from the C programming language. Function prints out in console and log file the passed arguments formatted by the usual printf conventions. Possible format values: see description of function sprintf. Example: printf("Isp (vac)=%8.4f Isp (SL)=%8.4f s\n", Is_v, Is_SL);

`sscanf(format, parameters)`	The function is a limited Javascript implementation of sscanf, originated from the C programming language.
	
	Reads data from "str" and stores them according to parameter "format":
	a) into array which is returned as a function result (if no additional arguments specified),
	b) into the locations given by the additional arguments (and returns number of scanned parameters).
	
	Since JavaScript does not support scalar reference variables, any additional arguments to the function will only be allowable here as strings referring to a global variable (which will then be set to the value
	found in 'str' corresponding to the appropriate conversion specification in 'format'.
	
	Possible format values:
	%% – A % followed by another % matches a single %.
	%i – integer
	%c – the character according to the ASCII value
	%d, %D – signed decimal number (integer)
	%u – unsigned decimal number (integer)
	%f, %e – floating-point number
	%o – octal number (integer)
	%s – string
	%x, %X – hexadecimal number
	
	Examples:
	
	sscanf('Is_v=350.0 s', 'Is_v=%f s');
	// returns an array with one element: [350.0]
	
	var Is_v;
	sscanf('Is_v=350.0 s', 'Is_v=%f s', 'Is_v');
	// returns number of scanned values: 1
	// assigns the scanned value 350.0 to variable Is_v
	
	sscanf("Is_v=350.0 s Is_SL=290.0 s", "Is_v=%f s Is_SL=%f s");
	// returns an array with two elements: [350.0, 290.0]

Function	Description
`exit() or quit()`	Exit from the interactive interpreter.

Object File

Function	Description
`File(`*`path_to_file, mode`*`)`	Open the file specified by name in given mode. Supported modes: "w" – write file, truncate the content if file exists "wa" – write file, append to existing content if file exists "r" – read file Example: var f = File("C:\tmp\results.txt", "w");

`print(`*`parameter`*`)`	Print out the parameter into the file.
	
	Examples:
	
	var f = File("C:\tmp\results.txt", "w");
	f.print("Starting analysis...");
	f.print("Is_v=" + pr.getNozzleExitSection().getIs_v().toPrecision(7));
	f.close();

`printf(format, `*`parameters`*`)`	The function is a limited Javascript implementation of printf, originated from the C programming language.
	
	Function prints out into the file the passed arguments formatted by the usual printf conventions.
	
	Possible format values: see description of function sprintf.
	
	Example:
	
	var f = File("C:\tmp\results.txt", "w");
	f.printf("Isp (vac)=%8.4f Isp (SL)=%8.4f s\n", Is_v, Is_SL);
	f.close();

`readLine()`	The function reads the next available line from the file and returns it as a string. The function returns null if no string is available.
	
	Example:
	
	var f = File("C:\tmp\results.txt", "r");
	var s = f.readLine();
	f.close();

`close()`	The function closes the file. Example: var f = File("C:\tmp\results.txt", "w");

Function	Description
	f.printf("Isp (vac)=%8.4f Isp (SL)=%8.4f s\n", Is_v, Is_SL); f.close();

Configuration API

Configuration API is indented for searching the thermodynamic database, obtaining species properties, preparing mono- bi- and multi-propellant compositions.

See also JavaScript class `Config`.

Object ConfigFile

Function	Parameters	Description
`ConfigFile()`		Default constructor. Creates new empty configuration object. Example: c = ConfigFile();
`ConfigFile(String path)`	`File path`	Constructor. Creates new empty configuration object and assign the specified file. The assigned file can be loaded with function read(). Example: c = ConfigFile("examples/RD-275.cfg");

`read()`		Reads the assigned file.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();

`read(String path)`	`File path`	Reads the specified file. Example: c = ConfigFile();

Function	Parameters	Description
		c.read("examples/RD-275.cfg");

`write()`		Writes the configuration into assigned file.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.setName("RD-275");
		c.write();

`write(String path)`	`File path`	Writes the configuration into specified file.
		
		Example:
		
		c = ConfigFile();
		c.setName("RD-275");
		c.write("examples/RD-275.cfg");

`validate()`		Validates the correctness of the configuration.
		
		Example:
		
		c = ConfigFile();
		c.setName("RD-275");
		c.validate();

`Number getVersion()`		Returns the version of configuration file.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		print("Version: "+c.getVersion());

`setName(String name)`	`Engine name`	Assigns the engine name.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.setName("RD-275");

Function	Parameters	Description
		c.write();

`String getName()`		Returns the engine name.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		print("Engine name: "+c.getName());

`setInfo(String info)`	`Description`	Assigns the description.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.setInfo("Test case for RD-275");
		c.write();

`String getInfo()`		Returns the assigned description.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		print("Case description: "+c.getInfo());

`Object getGeneralOptions()`		Returns the associated GeneralOptions object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();

`Object getNozzleFlowOptions()`		Returns the associated NozzleFlowOptions object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();

Function	Parameters	Description
		Object n = c.getNozzleFlowOptions();

`Object getCombustionChamberConditions()`		Returns the associated CombustionChamberConditions object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object cc = c.getCombustionChamberConditions();

`Object getPropellant()`		Returns the associated Propellant object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object p = c.getPropellant();

`Object getEngineSize()`		Returns the associated EngineSize object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object s = c.getEngineSize();

`Object getChamberGeometry()`		Returns the associated ChamberGeometry object.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object s = c.getChamberGeometry();

Object GeneralOptions

Function	Parameters	Description

`Boolean isMultiphaseFlow()`		Returns true, if multiphase flow and phase transitions should be considered.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		if (g.isMultiphaseFlow()) {
		print("Multiphase flow flag is on");
		}

`setMultiphaseFlow(Boolean flag)`	`true or false`
	`Default valie is true.`
		Assigns multiphase flow flag. Note that for the most of solid propellant problems this flag should be switched on.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		g.setMultiphaseFlow(true);

`Boolean isIons()`		Returns true, if species ionization effects should be considered.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		if (g.isIons()) {
		print("Ionization effects flag is on");
		}

`setIons(Boolean flag)`	`true or false`
	`Default valie is true.`
		Assigns ionization effects flag.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		g.setIons(false);

Function	Parameters	Description

`Boolean isFlowSeparation()`		Returns true, if flow separation effects should be considered.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		if (g.isFlowSeparation()) {
		print("Flow separation flag is on");
		}

`setFlowSeparation(Boolean flag)`	`true or false`
	`Default valie is true.`
		Assigns flow separation effects flag.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		g.setFlowSeparation(false);

Object NozzleFlowOptions

See also chapter Nozzle Flow Model.

Function	Parameters	Description
`Boolean isCalculateNozzleFlow()`		Returns true, if complete engine performance analysis should be performed.
`setCalculateNozzleFlow(Boolean flag)`	`true or false` `Default valie is true.`	Set flag CalculateNozzleFlow.
`Boolean isFreezingConditions()`		Returns true if freezing conditions defined.
`Object getFreezingConditions()`		Returns FreezingConditions object, or null if not defined.
`Object setFreezingConditions()`		Returns FreezingConditions object. Create new object and assign to configuration, if not defined before.
`deleteFreezingConditions()`		Removes assigned FreezingConditions object.
`Boolean isNozzleInletConditions()`		Returns true if nozzle inlet conditions defined.
`Object`		Returns NozzleInletConditions object,

Function	Parameters	Description
`getNozzleInletConditions()`		or null if not defined.
`Object setNozzleInletConditions()`		Returns NozzleInletConditions object. Create new object and assign to configuration, if not defined before.
`deleteNozzleInletConditions()`		Removes assigned NozzleInletConditions object.
`Boolean isNozzleExitConditions()`		Returns true if nozzle exit conditions defined.
`Object getNozzleExitConditions()`		Returns NozzleExitConditions object, or null if not defined.
`Object setNozzleExitConditions()`		Returns NozzleExitConditions object. Create new object and assign to configuration, if not defined before.
`deleteNozzleExitConditions()`		Removes assigned NozzleExitConditions object.
`Boolean isEfficiencyFactors()`		Returns true if efficiency factors defined.
`Object getEfficiencyFactors()`		Returns EfficiencyFactors object, or null if not defined.
`Object setEfficiencyFactors()`		Returns EfficiencyFactors object. Create new object and assign to configuration, if not defined before.
`deleteEfficiencyFactors()`		Removes assigned EfficiencyFactors object.
`Boolean isAmbientConditions()`		Returns true if ambient conditions defined.
`Object getAmbientConditions()`		Returns AmbientConditions object, or null if not defined.
`Object setAmbientConditions()`		Returns AmbientConditions object. Create new object and assign to configuration, if not defined before.
`deleteAmbientConditions()`		Removes assigned AmbientConditions object.
`Boolean isThrottlingConditions()`		Returns true if throttling seetings defined.
`Object getThrottlingConditions()`		Returns ThrottlingConditions object, or null if not defined.
`Object setThrottlingConditions()`		Returns ThrottlingConditions object. Create new object and assign to configuration, if not defined before.
`deleteThrottlingConditions()`		Removes assigned ThrottlingConditions object.

Object CombustionChamberConditions

Function	Parameters	Description
`Number getPressure(String unit)`	`Pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Returns chamber pressure in specified unit, or in "Pa" if unit not specified.
`setPressure(Number p, String unit)`	`Pressure value and unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Assigns chamber pressure in specified unit, or in "Pa" if unit not specified.

Object FreezingConditions

Function	Parameters	Description
`Boolean isCalculate()`		Returns true if frozen equilibrium should be calculated.
`ssCalculate(Boolean flag)`		Assigns frozen equilibrium flow flag.
`Boolean isPressure()`		Returns true if condition defined by presure.
`Number getPressure(String unit)`	`Pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Returns pressure in specified unit, or in "Pa" if unit not specified.
`setPressure(Number p, String unit)`	`Pressure value and unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Assigns pressure in specified unit, or in "Pa" if unit not specified.
`deletePressure()`		Remove condition defined by presure.
`Boolean isExpansionRatio()`		Returns true if condition defined by expansion area ratio.
`Number getExpansionRatio()`		Returns expansion area ratio.
`setExpansionRatio(Number a)`		Assigns expansion area ratio.
`deleteExpansionRatio()`		Remove condition defined by expansion area ratio.
`Boolean isPressureRatio()`		Returns true if condition defined by pressure ratio.
`Number getPressureRatio()`		Returns pressure ratio.
`setPressureRatio(Number a)`		Assigns pressure ratio.
`deletePressureRatio()`		Remove condition defined by pressure ratio.

Object NozzleInletConditions

Function	Parameters	Description
`Boolean isContractionAreaRatio()`		Returns true if nozzle inlet contraction area ratio defined.
`Number getContractionAreaRatio()`		Returns nozzle inlet contraction area ratio.
`setContractionAreaRatio(Number r)`	`nozzle inlet contraction area ratio (Ac/At)`	Assigns nozzle inlet contraction area ratio.
`deleteContractionAreaRatio()`		Removes nozzle inlet contraction area ratio.
`Boolean isMassFlux()`		Returns true if combustion chamber mass flux defined.
`Number getMassFlux(String unit)`	`Desired mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")`	Returns combustion chamber mass flux in specified unit .
`setMassFlux(Number f, String unit)`	`f - mass flux unit - mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")`	Assigns nozzle inlet mass flux.
`deleteMassFlux()`		Removes nozzle inlet mass flux.

Object NozzleSectionConditions

Function	Parameters	Description
`Boolean isAreaRatio()`		Returns true if nozzle section defined by area ratio (A/At).
`Number getAreaRatio()`		Returns assigned area ratio (A/At).
`setAreaRatio(Number r)`	`Area ratio (A/At)`	Assigns area ratio.
`deleteAreaRatio()`		Removes area ratio definition.
`Boolean isPressureRatio()`		Returns true if nozzle section defined by pressure ratio (pt/p).
`Number getPressureRatio()`		Returns assigned pressure ratio (pt/p).
`setPressureRatio(Number r)`	`Pressure ratio (pt/pt)`	Assigns pressure ratio.

Function	Parameters	Description
deletePressureRatio()		Removes pressure ratio definition.
Boolean isPressure()		Returns true if nozzle section defined by pressure.
Number getPressure(String unit)	Pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")	Returns assigned pressure in desired unit ro in "Pa", if unit not specified.
setPressure(Number p, String unit)	p - pressure unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")	Assigns pressure.
deletePressure()		Removes pressure definition.

Object EfficiencyFactors

Function	Parameters	Description
Boolean isApplyEfficiencyFactors()		Returns true if defined efficiency correction factors should be used for calculation of delivered performance.
setApplyEfficiencyFactors(Boolean flag)	ApplyEfficiencyFactors flag	Assigns ApplyEfficiencyFactors flag.
Boolean isReactionEfficiency()		Returns true if reaction efficiency assigned.
setReactionEfficiency(Number r)	Reaction efficiency	Assigns reaction efficiency.
Number getReactionEfficiency()		Returns assigned reaction efficiency.
deleteReactionEfficiency()		Removes assigned reaction efficiency.
Boolean isNozzleEfficiency()		Returns true if nozzle efficiency assigned.
setNozzleEfficiency(Number r)	Nozzle efficiency	Assigns nozzle efficiency.
Number getNozzleEfficiency()		Returns assigned nozzle efficiency.
deleteNozzleEfficiency()		Removes assigned nozzle efficiency.
Boolean isNozzleLength()		Returns true if nozzle length assigned.
setNozzleLength(Number l)	Nozzle length (%)	Assigns nozzle length.
Number getNozzleLength()		Returns assigned nozzle length (%).
deleteNozzleLength()		Removes assigned nozzle length.
Boolean isConeHalfAngle()		Returns true if conical nozzle half angle assigned.
setConeHalfAngle(Number a)	Half angle	Assigns conical nozzle half angle assigned.
Number getConeHalfAngle()		Returns assigned conical nozzle half

Function	Parameters	Description
		angle assigned.
`deleteConeHalfAngle()`		Removes assigned conical nozzle half angle assigned.

Object AmbientConditions

Function	Parameters	Description
`Boolean isFixedPressure()`		Returns true if fixed ambient pressure defined.
`Number getFixedPressure(String unit)`	`unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Returns fixed ambient pressure in desired unit.
`setFixedPressure(Number p, String unit)`	`p - pressure unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Assigns fixed ambient pressure in specified unit.
`deleteFixedPressure()`		Removes fixed ambient pressure.
`Boolean isRangePressure()`		Returns true if ambient condition defined as a pressure range.
`Number getRangePressureMin(String unit)`	`unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Returns low value of pressure range in desired unit.
`Number setRangePressureMin(Number p, String unit)`	`p - pressure unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Assigns low value of pressure range.
`Number getRangePressureMaxn(String unit)`	`unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Returns high value of pressure range in desired unit.
`setRangePressureMax(Number p, String unit)`	`p - pressure unit - pressure unit (one of "Pa" (default), "psi", "atm", "bar", "at", "MPa")`	Assigns high value of pressure range.
`deleteRangePressure()`		Removes pressure range.

Object ThrottlingConditions

Function	Parameters	Description
`Boolean isFixedFlowrate()`		Returns true if fixed mass flow rate defined.
`Number getFixedFlowrate()`		Returns defined fixed mass flow rate (decimal fraction, r/nominal).
`setFixedFlowrate(Number r)`	`mass flow rate, decimal fraction r/nominal`	Assign fixed mass flow rate.
`Boolean isRangeFlowrate()`		Returns true if flow rate condition defined as a flow rate range.
`Number getRangeFlowrateMin()`		Returns low value of flowe rate range (decimal fraction, r/nominal).
`setRangeFlowrateMin(Number r)`	`flow rate, decimal fraction r/nominal`	Assigns low value of pressure range.
`Number getRangeFlowrateMax()`		Returns high value of flowe rate range (decimal fraction, r/nominal).
`setRangeFlowrateMax(Number r)`	`flow rate, decimal fraction r/nominal`	Assigns high value of pressure range.
`deleteRangeFlowrate()`		Removes flow rate range.

Object Propellant

Function	Parameters	Description
`Number getRatio()`		Returns assigned propellant component ratio.
`String getRatioType()`		Returns type of assigned propellant component ratio ("O/F", "alpha", or "optimal").
`setRatio(Number r, String unit)`	`r - ratio` `unit - type of ratio (one of "O/F", "alpha", or "optimal")`	Assigns propellant component ratio.
`Number getOxidizerListSize()`		Returns number of species in oxidizer (not applicable for monopropellant mixture composition).
`Object getOxidizer(Number i)`	`index`	Returns Component object.
`addOxidizer(Object c)`	`Component object`	Adds specified component to the oxidizer.
`Number getFuelListSize()`		Returns number of species in fuel (not applicable for monopropellant mixture composition).
`Object getFuel(Number i)`	`index`	Returns Component object.

Function	Parameters	Description
addFuel(Object c)	Component object	Adds specified component to the fuel.
Number getSpeciesListSize()		Returns number of species in propellant mixture (not applicable for bipropellant composition).
Object getSpecies(Number i)	index	Returns Component object.
addSpecies(Object c)	Component object	Adds specified component to the mixture.
Object getPropellant()		Returns either Propellant or Mixture object ready for passing as parameter to constructors Reaction and Chamber. .

Object Component

Function	Parameters	Description
String getName()		Returns species name.
setMassFraction(Number f)	Mass fraction	Assigns mass fraction of the species in the component (for bipropellant systems) or propellant (for monopropellant systems).
Number getMassFraction()		Returns mass fraction.
setT(Number t, String unit)	t - temperature unit - temperature unit (one of "K", "F", "C")	Assigns initial temperature of the species.
Number getT(String unit)	temperature unit (one of "K", "F", "C")	Returns initial temperature of the species in desired unit.
setP(Number p, String unit)	p - temperature unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Assigns initial temperature of the species.
Number getP(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns initial pressure of the species in desired unit.

Object EngineSize

See also chapter Engine Definition.

Function	Parameters	Description
Boolean isThrust()		Returns true if engine thrust defined.
Number getThrust(String unit)	thrust unit (one of "kN", "kg", "lbf",	Returns engine thrust in desired unit.

Function	Parameters	Description
	"N")	
setThrust(Number t, String unit)	t - thrust unit - thrust unit (one of "kN", "kg", "lbf", "N")	Assigns engine thrust.
deleteThrust()		Removes thrust definition.
Boolean isAmbientPressure()		Returns true if ambient pressure defined.
Number getAmbientPressure(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns ambient pressure in desired unit.
setAmbientPressure(Number p, String unit)	p - pressure unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Assigns ambient pressure.
deleteAmbientPressure()		Removes ambient pressure definition.
Boolean isMdot()		Returns true if mass flow rate defined.
Number getMdot(String unit)	mass flow rate unit (one of "kg/s", "lbm/s")	Returns mass flow rate in desired unit.
setMdot(Number m, String unit)	m - mass flow rate unit - mass flow rate unit (one of "kg/s", "lbm/s")	Assigns mass flow rate.
deleteMdot()		Removes mass flow rate definition.
Boolean isThroatD()		Returns true if throat diameter defined.
Number getThroatD(String unit)	diameter unit (one of "mm", "in", "m", "ft")	Returns throat diameter.
setThroatD(Number d, String unit)	d - diameter unit - diameter unit (one of "mm", "in", "m", "ft")	Assigns throat diameter.
deleteThroatD()		Removes throat diameter definition.
Number getChambersNo()		Returns number of chambers.
setChambersNo(Number)	number of chambers	Assigns number of chambers.
Object getChamberGeometry()		Returns ChamberGeometry object.

Object ChamberGeometry

See also chapter Chamber Geometry.

Function	Parameters	Description
`Number getChamberLength(String unit)`	`length unit (one of "mm", "in", "m", "ft")`	Returns chamber charactersictic length (L*) in dsired unit.
`setChamberLength(Number L, String unit)`	`L - charactersictic length` `unit - length unit (one of "mm", "in", "m", "ft")`	Assigns chamber charactersictic length (L*).
`Number getContractionAngle()`		Returns nozzle inlet contraction angle (deg).
`setContractionAngle(Number a)`	`angle (deg)`	Assigns nozzle inlet contraction angle.
`Number getR1ToRtRatio()`		Returns ratio R1/Rt.
`setR1ToRtRatio(Number r)`	`ratio`	Assigns ratio R1/Rt.
`Number getRnToRtRatio()`		Returns ratio Rn/Rt.
`setRnToRtRatio(Number r)`	`ratio`	Assigns ratio Rn/Rt.
`Number getR2ToR2maxRatio()`		Returns ratio R2/R2max.
`setR2ToR2maxRatio(Number r)`	`ratio`	Assigns ratio R2/R2max.
`Boolean isTIC()`		Returns true if nozzle is shaped as truncated ideal contour.
`setTIC(Boolean s)`	`flag`	Assigns TIC flag.
`deleteTOC()`		Removes TIC flag.
`Boolean isParabolicExitAngle()`		Returns true if nozzle exit half angle defined.
`Number getParabolicExitAngle()`		Returns nozzle exit half angle (deg).
`setParabolicExitAngle(Number a)`	`angle (deg)`	Assigns nozzle exit half angle (deg).
`deleteParabolicExitAngle()`		Removes nozzle exit half angle definition.

Thermo API

Thermo API is indented for loading, manipulation and writing configuration files.

Object database

Function	Parameters	Description
`species()`		Static function. Returns Species object. Example: print(database.species("H2O2"));

Object Species

Function	Parameters	Description
`String getName()`		Returns species name.
`String getDescr()`		Returns species description.
`Boolean isReactantOnly()`		Returns true if species could not be usually used as an propellant component.
`Boolean isIon()`		Returns true if species is ionized.
`Number getCharge()`		Returns the charge of ionized species.
`Number getValence()`		Returns the valency of species.
`Boolean isCondensed()`		Returns true if species is condensed.
`Number getCondensed()`		Returns the number of condensed phase in increased order by temperature.
`Number getDHf298_15(String unit)`	`heat of formation unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm" )`	Returns $H^0(298.15)$ - heat of formation at the temperature 298.15 K and pressure 1 bar in desired unit.
`Number getDH298_15_0(String unit)`	`heat of formation unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm" )`	Returns $H^0(298.15) – H^0(0)$, if available.
`Number getT0(String unit)`	`temperature unit (one of "K", "F", "C")`	Returns standard temperature of the species in desired unit.
`Number getP0(String unit)`	`pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`	Returns standard pressure of the species in desired unit.
`Number getMinimumT(String unit)`	`temperature unit (one of "K", "F", "C")`	Returns minimum temperature of the species in desired unit.
`Number getMaximumT(String unit)`	`temperature unit (one of "K", "F", "C")`	Returns maximum temperature of the species in desired unit.
`Number checkT(double t, String unit)`	`t - temperature unit - temperature unit (one of "K", "F", "C")`	Checks whether the specified temperature is valid. If valid, returns specified temperature. Otherwise throws an exception.
`Number getM()`		Returns molecular weight of species.
`Number getR(String unit)`	`gas constant unit`	Returns gas constant in desired unit

Function	Parameters	Description
	(one of "J/(mol K)", "J/(kg K)", "kJ/(kg K)", "Btu/(lb-mol R)", "Btu/(lbm R)")	(applicable for gaseous species).
Number getCp(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns specific heat or molar heat capasity at specified temperature and constant pressure in desired unit.
Number getH(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy at specified temperature in desired unit.
Number getS(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - entropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")	Returns specific or molar entropy at specified temperature in desired unit.
Number getG(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - Gibbs energy unit (one of "J/mol", "Btu/lb-mol", "J/kg",	Returns Gibbs energy at specified temperature in desired unit.

Function	Parameters	Description
	"kJ/kg", "Btu/lbm")	

Object Propellant

Function	Parameters	Description
setRatio(Number r, String type)	r - ratio type - ratio type (one of "O/F", "alpha")	Assigns components ratio.
Number getRatio(String type)	ratio type (one of "O/F", "alpha")	Returns assigned components ratio.
Number getRatio0(String type)	ratio type (one of "O/F", "alpha")	Returns stoichiometric components ratio.
Object add(String type, String name, Number r)	type - component type ("o" or "f") name - species name r - mass fraction of the species in component	Adds species into component, assigns specified mass fraction, and returns Species object. The components designated as follows: "o" - oxidizer, "f" - fuel.
Object add(String type, String name, Number t, String tunit, Number p, String punit, Number r)	type - component type ("o" or "f") name - species name t - initial temperature tunit - temperature unit (one of "K", "F", "C") p - initial pressure punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") r - mass fraction of the species in component	Adds species with initial temperature and pressure into component, assigns specified mass fraction, and returns Species objects. The components designated as follows: "o" - oxidizer, "f" - fuel.
Object addOxidizer(String name, Number r)	name - species name r - mass fraction of the species in component	Adds species into oxidizer and assigns specified mass fraction.
Object addOxidizer(String type,	name - species name	Adds species with initial temperature

Function	Parameters	Description

`String name, Number t, String tunit, Number p, String punit, Number r)`	`t - initial temperature`
	
	`tunit - temperature unit (one of "K", "F", "C")`
	
	`p - initial pressure`
	
	`punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`
	
	`r - mass fraction of the species in component`
		and pressure into oxidizer, assigns specified mass fraction, and returns Species object.

`Object addFuel(String name, Number r)`	`name - species name`
	
	`r - mass fraction of the species in component`
		Adds species into fuel, assigns specified mass fraction, and returns Species object.

`Object addFuel(String type, String name, Number t, String tunit, Number p, String punit, Number r)`	`name - species name`
	
	`t - initial temperature`
	
	`tunit - temperature unit (one of "K", "F", "C")`
	
	`p - initial pressure`
	
	`punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`
	
	`r - mass fraction of the species in component`
		Adds species with initial temperature and pressure into fuel, assigns specified mass fraction, and returns Species object.

`add(String type, Object mixture, Number r)`	`type - component type ("o" or "f")`
	
	`mixture - Mixture object`
	
	`r - mass fraction of the mixture in`
		Adds mixture into component and assigns specified mass fraction. The components designated as follows: "o" - oxidizer, "f" - fuel.

Function	Parameters	Description
	component	
addOxidizer(Object mixture, Number r)	mixture - Mixture object r - mass fraction of the mixture in component	Adds mixture into oxidizer and assigns specified mass fraction.
addFuel(Object mixture, Number r)	mixture - Mixture object r - mass fraction of the mixture in component	Adds mixture into fuel and assigns specified mass fraction.
Number getM()		Returns molecular weight of propellant.
Number getH(String unit)	enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy of propellant in desired unit.
Number size(String type)	type - component type ("o" or "f")	Returns number of species included into component.
Object getSpecies(String type, Number i)	type - component type ("o" or "f") i - index	Returns Species object.
Boolean checkFractions()		Returns true if mass fractions assigned correctly (i.e. the sum of all mass fractions within each component is equals to 1.0).
Boolean checkFractions(String type)	type - component type ("o" or "f")	Returns true if mass fractions assigned correctly (i.e. the sum of all mass fractions within specified component is equals to 1.0).
Number getFraction(String type, Number i)	type - component type ("o" or "f") i - index	Returns assigned mass fraction.
setFraction(String type, Number i, Number f)	type - component type ("o" or "f") i - index f - mass fraction	Assigns mass fraction.
print(String units)	desired units (one of "SI" or "US")	Print out the information about propellant in desired units.

Object Mixture

Function	Parameters	Description

`Object addSpecies(String name, Number r)`	`name - species name`
	
	`r - mass fraction of the species in component`
		Adds species into mixture, assigns specified mass fraction, and returns Species object.

`Object addSpecies(String name, Number t, String tunit, Number p, String punit, Number r)`	`name - species name`
	
	`t - initial temperature`
	
	`tunit - temperature unit (one of "K", "F", "C")`
	
	`p - initial pressure`
	
	`punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`
	
	`r - mass fraction of the species in component`
		Adds species with initial temperature and pressure into mixture, assigns specified mass fraction, and returns Species objects.

`addMixture(Object mixture, Number r)`	`mixture - Mixture object`
	
	`r - mass fraction of the mixture in component`
		Adds mixture into this mixture object and assigns specified mass fraction.

`Number getValence()`		Returns the resulting valency of mixture.
`Number getM()`		Returns molecular weight of mixture.
`Number getH(String unit)`	`enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")`	Returns specific or molar enthalpy of mixture in desired unit.

`setT(Number t, String unit)`	`t - temperature`
	
	`unit - temperature unit (one of "K", "F", "C")`
		Assigns temperature to all species of the mixture.

`setP(Number p, String unit)`	`t - temperature`
	
	`punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`
		Assigns temperature to all species of the mixture.

Function	Parameters	Description
`Number size()`		Returns number of species included into mixture.
`Object getSpecies(Number i)`	`index`	Returns Species object.
`Boolean checkFractions()`		Returns true if mass fractions assigned correctly (i.e. the sum of all mass fraction within each component is equals to 1.0).
`Boolean checkFractions()`		Returns true if mass fractions assigned correctly (i.e. the sum of all mass fractions is equals to 1.0).
`Number getFraction(Number i)`	`index`	Returns assigned mass fraction.

`setFraction(Number i, Number f)`	`i - index`
	
	`f - mass fraction`
		Assigns mass fraction.

`print(String units)`	`desired units (one of "SI" or "US")`	Print out the information about mixture in desired units.

Reaction API

Reaction API is indented for running typical combustion problems `(p,H)=const`, `(p,S)=const`, `(p,T)=const`, `(v,U)=const`, `(v,S)=const`, `(v,T)=const` obtaining thermodynamic properties of the reaction products.

Object Product

Object Product represents an individual product of reaction.

Function	Parameters	Description
`String getName()`		Returns product's name.
`String getDescr()`		Returns product's description.
`Number getN()`		Returns number of moles of product.
`Number getT(String unit)`	`temperature unit (one of "K", "F", "C")`	Returns assigned temperature of product.
`Boolean isIon()`		Returns true if product is ionized.
`Number getCharge()`		Returns the charge of ionized product.
`Number getValence()`		Returns the valency of product.
`Boolean isCondensed()`		Returns true if product is condensed.
`Number getCondensed()`		Returns the number of condensed phase in increased order by temperature.
`Number getDHf298_15(String`	`heat of formation`	Returns $H^0(298.15)$ - heat of formation

Function	Parameters	Description
unit)	unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	at the temperature 298.15 K and pressure 1 bar in desired unit.
Number getDH298_15_0(String unit)	heat of formation unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns $H^0(298.15) - H^0(0)$, if available.
Number getT0(String unit)	temperature unit (one of "K", "F", "C")	Returns standard temperature of the product in desired unit.
Number getP0(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns standard pressure of the product in desired unit.
Number getMinimumT(String unit)	temperature unit (one of "K", "F", "C")	Returns minimum temperature of the product in desired unit.
Number getMaximumT(String unit)	temperature unit (one of "K", "F", "C")	Returns maximum temperature of the product in desired unit.
Number checkT(double t, String unit)	t - temperature unit - temperature unit (one of "K", "F", "C")	Checks whether the specified temperature is valid. If valid, returns specified temperature. Otherwise throws an exception.
Number getM()		Returns molecular weight of product.
Number getR(String unit)	gas constant unit (one of "J/(mol K)", "J/(kg K)", "kJ/(kg K)", "Btu/(lb-mol R)", "Btu/(lbm R)")	Returns gas constant in desired unit (applicable for gaseous species).
Number getCp(String unit)	unit - specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns specific heat or molar heat capasity at assigned temperature and constant pressure in desired unit.
Number getCp(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C")	Returns specific heat or molar heat capasity at specified temperature and constant pressure in desired unit.

Function	Parameters	Description
	unit - specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	
Number getH(String unit)	unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy at assigned temperature in desired unit.
Number getH(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy at specified temperature in desired unit.
Number getS(String unit)	unit - entropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")	Returns specific or molar entropy at assigned temperature in desired unit.
Number getS(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K", "F", "C") unit - entropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")	Returns specific or molar entropy at specified temperature in desired unit.
Number getG(String unit)	unit - Gibbs energy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns Gibbs energy at assigned temperature in desired unit.
Number getG(Number t, String tunit, String unit)	t - temperatur tunit - temperature unit (one of "K",	Returns Gibbs energy at specified temperature in desired unit.

Function	Parameters	Description
	"F", "C") unit - Gibbs energy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	

Object Reaction

Function	Parameters	Description
Reaction(Object p, Boolean multiphase, Boolean ionization)	p - Propellant or Mixture object multiphase - multiphase flag ionization - ionization flag	Constructor. Creates Reaction object, using specified Propellant or Mixture object. If multiphase flag is true, phase transitions effects are considered. If ionization flag is true, ionization effects are considered.
setPT(Number p, String p_unit, Number T, String T_unit)	p - pressure p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") T - temperature T_unit - temperature unit (one of "K", "F", "C")	Assigns pressure and temperature of combustion and switches the solving reaction problem to type (p,T)=consr.
setPH(Number p, String p_unit, Number H, String H_unit)	p - pressure p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") H - enthalpy H_unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "kJ/mol")	Assigns combustion pressure and enthalpy of propellant and switches the solving reaction problem to type (p,H)=const.
setPS(Number p, String p_unit, Number S, String S_unit)	p - pressure p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Assigns combustion pressure and enropy of propellant and switches the solving reaction problem to type (p,S)=const.

Function	Parameters	Description
	S - enropy S_unit - enropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "kJ/(mol K)")	
setVT(Number v, String v_unit, Number T, String T_unit)	v - specific volume v_unit - specific volume unit (one of "m^3/kg", "ft^3/lbm") T - temperature T_unit - temperature unit (one of "K", "F", "C")	Assigns specific volume (1/rho) and temperature of combustion and switches the solving reaction problem to type (v,T)=const.
setVH(Number v, String v_unit, Number U, String U_unit)	v - specific volume v_unit - specific volume unit (one of "m^3/kg", "ft^3/lbm") U – internal energy U_unit - internal energy unit (one of "J/mol", "Btu/lb-mol", "kJ/mol")	Assigns combustion specific volume (1/rho) and internal energy of propellant and switches the solving reaction problem to type (v,U)=const.
setVS(Number v, String v_unit, Number S, String S_unit)	v - specific volume v_unit - specific volume unit (one of "m^3/kg", "ft^3/lbm") S - enropy S_unit - enropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "kJ/(mol K)")	Assigns combustion specific volume (1/rho) and enropy of propellant and switches the solving reaction problem to type (v,S)=const.
solve(Boolean startWithCondensed)	startWithCondensed flag; if not specified default	Solves the prepared problem. In some cases the problem does not converge because condensed species

Function	Parameters	Description
	value is false.	not included before first iteration. To solve such a problems, set startWithCondensed to true.
reset(Boolean startWithCondensed)	startWithCondensed flag; if not specified default value is false.	Reset the problem before repeating the solving. In some cases the problem does not converge because condensed species not included before first iteration. To solve such a problems, set startWithCondensed to true.
getP(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns assigned pressure.
getV(String unit)	specific volume unit (one of "m^3/kg", "ft^3/lbm")	Returns assigned specific volume.
getT(String unit)	temperature unit (one of "K", "F", "C")	Returns the temperature of reaction.
getH(String unit)	enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy of reaction products.
getU(String unit)	internal energy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar internal energy of reaction products.
getS(String unit)	entropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")	Returns specific or molar entropy of reaction products.
Boolean hasCondensedPhase()		Returns true if reaction products contains condensed species.
Object getResultingMixture()		Returns Mixture object, containing all products of reaction. Note that function Mixture.getSpecies() actually returns Product object.
print(String units)	desired units (one of "SI" or "US")	Print out the information about problem results.

Object Derivatives

Function	Parameters	Description
Derivatives(Object r)	Reaction object	Constructor. Creates new object Derivatives for given Reaction object.
Derivatives(Object r)	Reaction object	Constructor. Creates new object Derivatives for given Reaction object.
Number getCp(String unit)	specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns specific heat or molar heat capasity of reaction products at constant pressure in desired unit.
Number getCv(String unit)	specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns specific heat or molar heat capasity of reaction products at constant volume in desired unit.
Number getR(String unit)	gas constant unit (one of "J/(mol K)", "J/(kg K)", "kJ/(kg K)", "Btu/(lb-mol R)", "Btu/(lbm R)")	Returns gas constant of reaction products in desired unit.
Number getK()		Returns isentropic exponent of reaction products.
Number getGamma()		Returns specific heat ratio of reaction products.
Number getA(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns velocity of sound in desired unit.
Number getRho(String unit)	density unit (one of "kg/m^3", "g/m^3" or "lbm/ft^3")	Returns density of reaction products in desired unit.
Number getRhoGas(String unit)	density unit (one of "kg/m^3", "g/m^3" or "lbm/ft^3")	Returns density of gaseous reaction products in desired unit.
Number getZ()		Returns mass fraction of condensed reaction products.
Number getM()		Returns molecular weight of reaction products.
Number getV(String unit)	unit (one of "x10^-4 kg/(m s)",	Returns viscosity of reacting mixture.

Function	Parameters	Description
	"mpoise" or "x10^-4 lb/(ft s)")	
Number getCr(String unit)	unit (one of "W/(m K)", "mW/(cm K)" or "Btu/(hr ft R)")	Returns equilibrium conductivity of reacting mixture.
Number getCfr(String unit)	unit (one of "x10^-4 kg/(m s)", "mpoise" or "x10^-4 lb/(ft s)")	Returns frozen conductivity of reacting mixture.
Number getCeql(String unit)	unit (one of "x10^-4 kg/(m s)", "mpoise" or "x10^-4 lb/(ft s)")	Returns effective conductivity of reacting mixture.
Number getCpr(String unit)	unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns equilibrium specific heat of reacting mixture at constant pressure
Number getCpfr(String unit)	unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns frozen specific heat of reacting mixture at constant pressure
Number getCpeql(String unit)	unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	Returns effective specific heat of reacting mixture at constant pressure
Number getPrr()		Returns equilibrium Prandtl number (Pr) of reacting mixture.
Number getPrfr()		Returns frozen Prandtl number (Pr) of reacting mixture.
Number getPreql()		Returns effective Prandtl number (Pr) of reacting mixture.
print(String units)	desired units (one of "SI" or "US")	Prints out the information derivative properties.

Performance API

Performance API is indented for running typical rocket propulsion problems and obtaining performance parameters.

Object Chamber

Function	Parameters	Description
Chamber(Object p, Boolean multiphase, Boolean ionization)	p - Propellant or Mixture object multiphase - multiphase flag ionization - ionization flag	Constructor. Creates Chamber object, using specified Propellant or Mixture object. If multiphase flag is true, phase transitions effects are considered. If ionization flag is true, ionization effects are considered.
setP(Number p, String unit)	p - pressure pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Assigns combustion chamber pressure. Enthalpy of propellant is automatically calculated from assigned propellant or mixture.
setPH(Number p, String p_unit, Number H, String H_unit)	p - pressure p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") H - enthalpy H_unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "kJ/mol")	Assigns combustion chamber pressure and enthalpy of propellant or mixture.
setFcr(Number f)	area ratio	Assigns nozzle inlet contraction area ratio (A/At).
setMr(Number m, String unit)	m - mass flux unit - mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")	Assigns combustion chamber mass flux.
solve(Boolean finiteChamberSection, Boolean startWithCondensed)		Solve the problem.
Number getFcr()		Returns nozzle inlet contraction area ratio (A/At).
Number getMr(String unit)	mass flux unit (one	Returns combustion chamber mass

Function	Parameters	Description
	of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")	flux in desired unit.
Number getPc_0(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns stagnation pressure at nozzle inlet in desired unit.
Number getSigmaC()		Returns stagnation pressure drop coefficient.
Number getWc(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns velocity at nozzle inlet (combustion chamber end) in desired unit.
Number getCstar(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns chamber characteristic velocity in desired unit.
Object getReaction(Number station) Object getReaction(String station)	chamber station: 0 or "inj" - injector 1 or "inl" - nozzle inlet (combustion chamber end) 2 or "thr" - nozzle throat	Returns Reaction object for specified station.
Object getDerivatives(Number station) Object getDerivatives(String station)	chamber station: 0 or "inj" - injector 1 or "inl" - nozzle inlet (combustion chamber end) 2 or "thr" - nozzle throat	Returns Derivatives object for specified station.

Object NozzleSectionConditions

Function	Parameters	Description
NozzleSectionConditions(Object c, Number p, String ptype) NozzleSectionConditions(Object c, Number p, Number ptype)	c - Chamber p - parameter ptype - type of parameter: 0, "A/At" or "A/A*" - parameter is area ratio 1 or "p" -	Constructor. Creates NozzleSectionConditions object, using specified Chamber object. Parameter defines the nozzle station.

Function	Parameters	Description
	parameter is pressure (Pa) 2, "pc/p" or "pi" - parameter is pressure ratio.	
Number getFr()		Returns expansion area ratio (A/At) at nozzle section.
Number getPi()		Returns pressure ratio (p/pc) at nozzle section.
Number getP(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns pressure at nozzle section in desired unit.
Number getW(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns velocity at nozzle section in desired unit.
Number getMach()		Returns Mach number at nozzle section.
Number getMr(String unit)	mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")	Returns mass flux at nozzle section in desired unit.
Number getIs_v(String unit)	specific impulse unit (one of "s", "m/s" or "ft/s")	Returns vacuum specific impulse in desired unit.
Number getIs(String unit)	specific impulse unit (one of "s", "m/s" or "ft/s")	Returns optimum expansion specific impulse in desired unit.
Number getIs_H(Number p, String punit, String unit)	p - ambient pressure punit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") unit - specific impulse unit (one of "s", "m/s" or "ft/s")	Returns expansion specific impulse at specified ambient pressure in desired unit.
Number getCf_v()		Returns vacuum thrust coefficient.
Number getCf()		Returns optimum expansion thrust coefficient.
Number getCf_H(Number p, String punit)	p - ambient pressure punit - pressure	Returns thrust coefficient at specified ambient pressure .

Function	Parameters	Description
	unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	
Object getReaction()		Returns Reaction object for this nozzle station.
Object getDerivatives()		Returns Derivatives object for this nozzle station.

Object ChamberFr

Function	Parameters	Description
ChamberFr(Object c, Number p, String ptype) ChamberFr(Object c, Number p, Number ptype)	c - Propellant or Mixture object p - parameter ptype - type of parameter: 0, "A/At" or "A/A*" - parameter is area ratio 1 or "p" - parameter is pressure (Pa) 2, "pc/p" or "pi" - parameter is pressure ratio.	Constructor. Creates Chamber object, using specified Propellant or Mixture object. Parameter defines the nozzle station where frozen equilibrium model is applied.
setP(Number p, String unit)	p - pressure pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Assigns combustion chamber pressure. Enthalpy of propellant is automatically calculated from assigned propellant or mixture.
setPH(Number p, String p_unit, Number H, String H_unit)	p - pressure p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") H - enthalpy H_unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "kJ/mol")	Assigns combustion chamber pressure and enthalpy of propellant or mixture.
setFcr(Number f)	area ratio	Assigns nozzle inlet contraction area ratio (A/At).
setMr(Number m, String unit)	m - mass flux	Assigns combustion chamber mass

Function	Parameters	Description
	unit - mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")	flux.
solve(Boolean finiteChamberSection, Boolean startWithCondensed)		Solve the problem.
Object getEquilibriumSection()		Returns NozzleSectionConditions object for the nozzle station where shifting equilibrium model is switched to frozen model.
Number getFcr()		Returns nozzle inlet contraction area ratio (A/At).
Number getMr(String unit)	mass flux unit (one of "kg/(m2·s)", "kg/(m2-s)", "kg/(s·m2)", "kg/(s-m2)", "lbm/(ft2·s)", "lbm/(ft2-s)", "lbm/(s·ft2)", "lbm/(s-ft2)")	Returns combustion chamber mass flux in desired unit.
Number getPc_0(String unit)	pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")	Returns stagnation pressure at nozzle inlet in desired unit.
Number getSigmaC()		Returns stagnation pressure drop coefficient.
Number getWc(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns velocity at nozzle inlet (combustion chamber end) in desired unit.
Number getCstar(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns chamber characteristic velocity in desired unit.
Object getReaction(Number station) Object getReaction(String station)	chamber station: 0 or "inj" - injector 1 or "inl" - nozzle inlet (combustion chamber end) 2 or "thr" - nozzle throat	Returns Reaction object for specified station.
Object getDerivatives(Number station)	chamber station: 0 or "inj" -	Returns Derivatives object for specified station.

Function	Parameters	Description

`Object getDerivatives(String station)`	`injector`
	`1 or "inl" - nozzle inlet (combustion chamber end)`
	`2 or "thr" - nozzle throat`	

Object NozzleSectionConditionsFr

Function	Parameters	Description

`NozzleSectionConditionsFr(Object c, Number p, String ptype)`
 	
`NozzleSectionConditionsFr(Object c, Number p, Number ptype)`
	`c - ChamberFr object`
	
	`p - parameter`
	
	`ptype - type of parameter:`
	`0, "A/At" or "A/A*" - parameter is area ratio`
	`1 or "p" - parameter is pressure (Pa)`
	`2, "pc/p" or "pi" - parameter is pressure ratio.`
		Constructor.
		
		Creates NozzleSectionConditionsFr object, using specified ChamberFr object.
		
		Parameter defines the nozzle station.

`Number getT(String unit)`	`temperature unit (one of "K", "F", "C")`	Returns temperature of reaction products in desired unit.
`Number getH(String unit)`	`unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")`	Returns specific or molar enthalpy of reaction products in desired unit.
`Number getS(String unit)`	`unit - entropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")`	Returns specific or molar entropy of reaction products in desired unit.
`Number getCp(String unit)`	`specific heat unit (one of "J/(kg K)", "kJ/(kg K)", "Btu/(lbm R)")` `or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")`	Returns specific heat or molar heat capasity of reaction products at constant pressure in desired unit.
`Number getCv(String unit)`	`specific heat unit (one of "J/(kg K)",`	Returns specific heat or molar heat capasity of reaction products at

Function	Parameters	Description
	"kJ/(kg K)", "Btu/(lbm R)") or molar heat capasity unit (one of "J/(mol K)", "Btu/(lb-mol R)")	constant volume in desired unit.
Number getR(String unit)	gas constant unit (one of "J/(mol K)", "J/(kg K)", "kJ/(kg K)", "Btu/(lb-mol R)", "Btu/(lbm R)")	Returns gas constant of reaction products in desired unit.
Number getK()		Returns isentropic exponent of reaction products.
Number getA(String unit)	velocity unit (one of "m/s" or "ft/s")	Returns velocity of sound in desired unit.
Number getRho(String unit)	density unit (one of "kg/m^3", "g/m^3" or "lbm/ft^3")	Returns density of reaction products in desired unit.
Number getM()		Returns molecular weight of reaction products.

Object Performance

Function	Parameters	Description
Performance(Object c)	ConfigFile object	Constructor. Creates Performance object, using specified ConfigFile object.
clearForRestart()		Prepares the object for restart.
solve()		Solves the configured problem.
Number optimizeForSpecificImpulse()		Calculates the optimum component ratio and solves the problem using found ratio, returning it.
Number optimizeForSpecificImpulse(Number left, Number right)	left - low value of oxidizer excess coefficient right - high value of oxidizer excess coefficient	Calculates the optimum component ratio and solves the problem using found ratio, returning it. Parameters define the range of component ratio.
Boolean isOptimized()		Returns true if solution is found as result of optimization.
Object getPropellant()		Returns Propellant object.
Object getMixture()		Returns Mixture object.
Object getData()		Returns assigned ConfigFile object.

Function	Parameters	Description
`Object getChamber()`		Returns Chamber object. Only applicable for problems, where nozzle flow is solved.
`Object getExitSection()`		Returns NozzleSectionConditions or NozzleSectionConditionsFr object. Only applicable for problems, where nozzle flow is solved.
`Object getOverExpansionSection()`		Returns NozzleSectionConditions or NozzleSectionConditionsFr object, that represent the nozzle station where flow separation occurs. Only applicable for problems, where nozzle flow is solved.
`Number getPaCrit()`		Returns critical ambient pressure. Only applicable for problems, where nozzle flow is solved.
`Object solveNozzleSection(Number condition, String type, Boolean checkForFreezing, Boolean checkForOverexpansion, Number pa, String paunit)`		Returns NozzleSectionConditions or NozzleSectionConditionsFr object for specified conditions. Only applicable for problems, where nozzle flow is solved.
`Object getReaction()`		Returns Reaction object. Only applicable for problems, where nozzle flow is not solved.

`Object getDerivatives()`	`chamber station:`
	`0 or "inj" - injector`
	`1 or "inl" - nozzle inlet (combustion chamber end)`
	`2 or "thr" - nozzle throat`
		Returns Derivatives object.
		Only applicable for problems, where nozzle flow is not solved.

`printHeader()`		Prints results header.
`printResults(String units)`	`desired units (one of "SI" or "US")`	Prints out results.

Scripting examples

Performance - Example 1

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.
```

```
 performance1.js

 This script loads existing configuration file,
 solves the performance problem and prints out the results.

*************************************************/

// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration.
p = Performance(c);

// Solve the problem
p.solve();

// Print out the results in SI units (default).
p.printResults();
```

Performance - Example 2

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 performance2.js

 This script loads existing configuration file,
 solves the performance problem and prints out the
 combustion parameters for each station "injector",
 "nozzle inlet", "nozzle throat" and "nozzle exit" separately.

 *************************************************/

// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration.
p = Performance(c);

// Solve the problem.
p.solve();

// Get the combustion chamber object.
chamber = p.getChamber();
```

```
// Get objects Reaction and Derivatives for injector station (0).
injector_r = chamber.getReaction(0);
injector_d = chamber.getDerivatives(0);

// Get objects Reaction and Derivatives for nozzle inlet station (1).
nozzleInlet_r = chamber.getReaction(1);
nozzleInlet_d = chamber.getDerivatives(1);

// Get objects Reaction and Derivatives for nozzle throat station (2).
throat_r = chamber.getReaction(2);
throat_d = chamber.getDerivatives(2);

// Get objects Reaction and Derivatives for nozzle exit station.
nozzleExit_r = p.getNozzleExitSection().getReaction();
nozzleExit_d = p.getNozzleExitSection().getDerivatives();

// Print out the results in US units.

print("*****************************************************");
print("Injector");
print("*****************************************************");
injector_r.print("US");
injector_d.print("US");

print("*****************************************************");
print("Nozzle Inlet");
print("*****************************************************");
nozzleInlet_r.print("US");
nozzleInlet_d.print("US");

print("*****************************************************");
print("Nozzle Throat");
print("*****************************************************");
throat_r.print("US");
throat_d.print("US");

print("*****************************************************");
print("Nozzle Exit");
print("*****************************************************");
nozzleExit_r.print("US");
nozzleExit_d.print("US");
```

Performance - Example 3

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
```

```
 additional information or have any questions.

 performance2.js

 This script loads existing configuration file, and runs a number
 of problems,  replacing the pre-configured O/F ratio with values
 from array.

**************************************************/

load("resources/scripts/printf.js");

// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration.
p = Performance(c);

// Array of O/F weight ratios
r = [2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1];

// Print out table header
printf("#%4s %8s %8s %8s", "r", "Is_v,s", "Is_opt,s", "Is_sl,s");

// Solve the performance problem for each ratio in the array.
for (i=0; i<r.length; ++i) {
    // Assign new O/F weight ratio, replacing pre-configured one.
    p.getPropellant().setRatio(r[i], "O/F");

    // Solve the problem.
    p.solve();

    // Print out current O/F weight ratio and calculated specific
    // impulse in vacuum, at optimum expansion, and at sea level.
    printf(" %4.2f %8.2f %8.2f %8.2f",
        p.getPropellant().getRatio("O/F"),
        p.getNozzleExitSection().getIs_v("s"),
        p.getNozzleExitSection().getIs("s"),
        p.getNozzleExitSection().getIs_H(1, "atm", "s")
    );

    // Prepare the solver for restart.
    p.clearForRestart();
}
```

Performance - Example 4

```
/**************************************************
 RPA - Tool for Rocket Propulsion Analysis
```

```
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 performance4.js

 This script loads existing configuration file, solves the main problem to
obtain chamber conditions,
 and then calculates the performance for nozzles with different expansion
area ratio.

**************************************************/

load("resources/scripts/printf.js");

// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration,
// and then solve the problem to get chamber/throat conditions.
p = Performance(c);
p.solve();

// Print out configured area ratio and corresponding vacuum
// specific impulse
print("#Configured A/At = "+p.getNozzleExitSection().getFr().toFixed(2)+"
 Is_v = "+p.getNozzleExitSection().getIs_v("m/s").toPrecision(7)+" m/s");

// Define an array with different expansion area ratios.
// Note that area ratio 26.2 is equal to the pre-configured one for RD-275.
r = [10, 20, 26.2, 40];

// Print out table header
printf("#%5s %8s %8s %8s", "A/At", "Is_v,s", "Is_v,m/s", "Is,ft/s");

// Calculate performance for each area ratio in the array.
for (i=0; i<r.length; ++i) {
     s = p.solveNozzleSection(r[i], "A/At");

    // Print out current area  ratio and calculated vacuum specific
    // impulse in s, m/s and ft/s.
    printf(" %5.2f %8.2f %8.2f %8.2f",
        r[i],
        s.getIs_v("s"),
        s.getIs_v("m/s"),
        s.getIs_v("ft/s")
    );

}
```

Performance - Example 5

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 performance5.js

*************************************************/

load("resources/scripts/printf.js");


// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration,
// and then solve the problem to get chamber/throat conditions.
p = Performance(c);
p.solve();

// Get the combustion chamber object.
chamber = p.getChamber();

// Nozzle area ratios
subsonic = [1.54, 1.35, 1.2, 1];
supersonic = [5, 10, 26.2, 50, 100];

printf("#%6s %5s %5s %8s", "A/At", "Mach", "p,MPa", "Is_v,s");

// For each subsonic nozzle section, print out A/At, Mach number,
// and pressure
for (i=0; i<subsonic.length; ++i) {
    s = NozzleSectionConditions(chamber, subsonic[i], "A/At", false);
    printf(" %6.2f %5.2f %5.2f", s.getFr(), s.getMach(), s.getP("MPa"));
}

// For each supersonic nozzle section, print out A/At, Mach number,
// pressure, and vacuum specific impulse
for (i=0; i<supersonic.length; ++i) {
    s = NozzleSectionConditions(chamber, supersonic[i], "A/At", true);
    printf(" %6.2f %5.2f %5.2f %8.2f", s.getFr(), s.getMach(),
s.getP("MPa"), s.getIs_v("s"));
}
```

Propellant

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 propellant.js

*************************************************/
load("resources/scripts/printf.js");

prop = Propellant();
prop.setRatio(6.0, "O/F");    // Set O/F weight ratio
prop.addOxidizer("O2(L)");    // Add oxidizer at it's normal temperature and
                              // atmospheric pressure
prop.addFuel("H2(L)", 0.8);   // Add 1st fuel component at it's normal
                              // temperature and atmospheric pressure
prop.addFuel("RP-1", 0, "K", 3, "atm", 0.2); // Add 2nd fuel component at
                                             // it's normal temperature and
                                             // pressure 3 atm
// The sum "H2(L) mass fraction" (0.8) + "RP-1 mass fraction" (0.2) must be
equal to 1.0

chamber = Chamber(prop);
chamber.setP(10, "MPa");      // Chamber pressure
chamber.setFcr(3);            // Nozzle inlet contraction area ratio
chamber.solve(true);          // finiteChamberSection=true

nozzleExit = NozzleSectionConditions(chamber, 40, "A/At", true);

// Get objects Reaction and Derivatives for injector station (0).
injector_r = chamber.getReaction(0);
injector_d = chamber.getDerivatives(0);

// Get objects Reaction and Derivatives for nozzle inlet station (1).
nozzleInlet_r = chamber.getReaction(1);
nozzleInlet_d = chamber.getDerivatives(1);

// Get objects Reaction and Derivatives for nozzle throat station (2).
throat_r = chamber.getReaction(2);
throat_d = chamber.getDerivatives(2);

// Get objects Reaction and Derivatives for nozzle exit.
nozzleExit_r = nozzleExit.getReaction();
nozzleExit_d = nozzleExit.getDerivatives();

// Print out propellant information
```

```
prop.print("US");

print("****************************************************");
print("Injector");
print("****************************************************");
injector_r.print("US");
injector_d.print("US");

print("****************************************************");
print("Nozzle Inlet");
print("****************************************************");
nozzleInlet_r.print("US");
nozzleInlet_d.print("US");

print("****************************************************");
print("Nozzle Throat");
print("****************************************************");
throat_r.print("US");
throat_d.print("US");

print("****************************************************");
print("Nozzle Exit");
print("****************************************************");
nozzleExit_r.print("US");
nozzleExit_d.print("US");

print("****************************************************");
print("Performance");
print("****************************************************");
printf(" Is_v=%8.2f s\n Is_opt=%8.2f s\n Is_sl=%8.2fs",
    nozzleExit.getIs_v("s"),
    nozzleExit.getIs("s"),
    nozzleExit.getIs_H(1, "atm", "s")
);
```

Mixture

```
/****************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 mixture.js

****************************************************/

load("resources/scripts/printf.js");
```

```
mix = Mixture();
// Add 1st component at it's normal temperature and atmospheric pressure
mix.addSpecies("O2(L)", 0.8);
// Add 2nd component at it's normal temperature and atmospheric pressure
mix.addSpecies("H2(L)", 0.02);
// Add 3rd component at it's normal temperature and pressure 3 atm
mix.addSpecies("RP-1", 0, "K", 3, "atm", 0.15);
// Add 4th component at it's normal temperature and atmospheric pressure
mix.addSpecies("AL(cr)", 0.03);
// The sum
//   "O2(L) mass fraction" (0.8) +
//   "H2(L) mass fraction" (0.02)  +
//   "RP-1 mass fraction" (0.15) +
//   "AL(cr) mass fraction" (0.03)
// must be equal to 1.0

chamber = Chamber(mix);
chamber.setP(10, "MPa"); // Chamber pressure
chamber.setFcr(3);       // Nozzle inlet contraction area ratio
chamber.solve(true);     // finiteChamberSection=true

nozzleExit = NozzleSectionConditions(chamber, 40, "A/At", true);

// Get objects Reaction and Derivatives for injector station (0).
injector_r = chamber.getReaction(0);
injector_d = chamber.getDerivatives(0);

// Get objects Reaction and Derivatives for nozzle inlet station (1).
nozzleInlet_r = chamber.getReaction(1);
nozzleInlet_d = chamber.getDerivatives(1);

// Get objects Reaction and Derivatives for nozzle throat station (2).
throat_r = chamber.getReaction(2);
throat_d = chamber.getDerivatives(2);

// Get objects Reaction and Derivatives for nozzle exit.
nozzleExit_r = nozzleExit.getReaction();
nozzleExit_d = nozzleExit.getDerivatives();

// Print out propellant information
mix.print("US");

print("*************************************************");
print("Injector");
print("*************************************************");
injector_r.print("US");
injector_d.print("US");

print("*************************************************");
print("Nozzle Inlet");
```

```
print("*****************************************************");
nozzleInlet_r.print("US");
nozzleInlet_d.print("US");

print("*****************************************************");
print("Nozzle Throat");
print("*****************************************************");
throat_r.print("US");
throat_d.print("US");

print("*****************************************************");
print("Nozzle Exit");
print("*****************************************************");
nozzleExit_r.print("US");
nozzleExit_d.print("US");

print("*****************************************************");
print("Performance");
print("*****************************************************");
printf(" Is_v=%8.2f s\n Is_opt=%8.2f s\n Is_sl=%8.2fs",
   nozzleExit.getIs_v("s"),
   nozzleExit.getIs("s"),
   nozzleExit.getIs_H(1, "atm", "s")
);
```

Reaction

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 reaction.js

*************************************************/

prop = Propellant();
prop.setRatio(6.0, "O/F");     // Set O/F weight ratio
prop.addOxidizer("O2(L)");     // Add oxidizer at it's normal temperature
                               // and atmospheric pressure
prop.addFuel("H2(L)", 0.8);    // Add 1st fuel component at it's normal
                               // temperature and atmospheric pressure
prop.addFuel("RP-1", 0, "K", 3, "atm", 0.2); // Add 2nd fuel component at
                                             // it's normal temperature and
                                             // pressure 3 atm
// The sum "H2(L) mass fraction" (0.8) + "RP-1 mass fraction" (0.2)
// must be equal to 1.0
```

```
// Print out propellant information
prop.print("US");

print("****************************************************");
print("Problem (p,H)=const");
print("****************************************************");

r1 = Reaction(prop);
r1.setPH(10, "MPa", prop.getH("Btu/lb-mol"), "Btu/lb-mol");
r1.solve();

d1 = Derivatives(r1);

// Print out reaction information
r1.print("US");
d1.print("US");

print("****************************************************");
print("Problem (p,T)=const");
print("****************************************************");

r2 = Reaction(prop);
r2.setPT(10, "MPa", 6062.38174, "F", true);
r2.solve();

d2 = Derivatives(r2);

// Print out reaction information
r2.print("US");
d2.print("US");

print("****************************************************");
print("Problem (p,S)=const");
print("****************************************************");

r3 = Reaction(prop);
r3.setPS(10, "MPa", 0.050, "Btu/(lb-mol R)");
r3.solve();

d3 = Derivatives(r3);

// Print out reaction information
r3.print("US");
d3.print("US");
```

Reaction Products

```
/****************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
```

```
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 reaction.js

*************************************************/

load("resources/scripts/printf.js");

prop = Propellant();
prop.setRatio(6.0, "O/F");     // Set O/F weight ratio
prop.addOxidizer("O2(L)");     // Add oxidizer at it's normal
                               // temperature and atmospheric pressure
prop.addFuel("H2(L)");         // Add fuel at it's normal temperature
                               // and atmospheric pressure

// Print out propellant information
prop.print("US");

r = Reaction(prop);
r.setPH(10, "MPa", prop.getH("J/mol"), "J/mol");
r.solve();

d = Derivatives(r);

products = r.getResultingMixture();

// Reaction products total number of moles
// The absolute number does not matter, and only used
// for calculation of mole fraction
totalMoles = 0;
for (i=0; i<products.size(); ++i) {
    totalMoles += products.getSpecies(i).getN();
}

// Reaction products total mass (kg)
// The absolute number does not matter, and only used
// for calculation of mass fraction
totalMass = totalMoles*d.getM()/1000;

printf("%15s %9s %9s %4s", "Name", "Mass Frac", "Mole Frac", "Cond");

sum1 = 0;
sum2 = 0;

for (i=0; i<products.size(); ++i) {
    // Reaction product
    s = products.getSpecies(i);

    // Mass of reaction product (kg)
```

```
    // The absolute number does not matter, and only used
    // for calculation of mass fraction
    mass = s.getN()*s.getM()/1000;

    massFraction = mass/totalMass;
    moleFraction = s.getN()/totalMoles;

    sum1 += massFraction;
    sum2 += moleFraction;

    // We are printing out mass fraction in  format "%9.7f",
    // so skip all products with massFraction<1e-7
    if (massFraction<1e-7) {
        continue;
    }

    printf(
        "%15s %9.7f %9.7f %4d",
        s.getName(),
        massFraction,
        moleFraction,
        s.getCondensed()
    );
}

printf(
    "%15s %9.7f %9.7f",
    "Summ:",
    sum1,
    sum2
);
```

Frozen Equilibrium

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 frozen.js

*************************************************/

load("resources/scripts/printf.js");

prop = Propellant();
prop.setRatio(6.0, "O/F");  // Set O/F weight ratio
prop.addOxidizer("O2(L)");  // Add oxidizer at it's normal temperature and
```

```
atmospheric pressure
prop.addFuel("H2(L)");       // Add fuel at it's normal temperature and
                             // atmospheric pressure

// Define chamber to calculate performance with frozen equilibrium flow,
// specifying  nozzle area ratio where shifting equilibrium model
// switched to frozen one
chamber = ChamberFr(prop, true, true, 10, "A/At");
chamber.setP(10, "MPa");  // Chamber pressure
chamber.setFcr(3);        // Nozzle inlet contraction area ratio
chamber.solve(true);      // finiteChamberSection=true

// Get nozzle area ratio where shifting equilibrium
// model switched to frozen one
frozenAt = chamber.getEquilibriumSection().getFr();

// Define an array with different expansion area ratios.
r = [2, 5, 10, 20, 26.2, 40];

// Print out table header
printf("#%5s %8s %8s %8s", "A/At", "Is_v,s", "Is_v,m/s", "Is,ft/s");

// Calculate performance for each area ratio in the array.
for (i=0; i<r.length; ++i) {
   s = r[i]>frozenAt ?
     NozzleSectionConditionsFr(chamber, r[i], "A/At", true) :
     NozzleSectionConditions(chamber, r[i], "A/At", true);

   // Print out current area  ratio and calculated vacuum
// specific impulse in s, m/s and ft/s.
   printf(" %5.2f %8.2f %8.2f %8.2f",
      r[i],
      s.getIs_v("s"),
      s.getIs_v("m/s"),
      s.getIs_v("ft/s")
   );
}
```

Nested Analysis

```
/*************************************************
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 nested_analysis1.js

*************************************************/
```

```
load("resources/scripts/printf.js");

// Load configuration file "examples/RD-275.cfg".
c = ConfigFile("examples/RD-275.cfg");
c.read();

// Create Performance object, initializing it with loaded configuration.
p = Performance(c);

// Array of O/F weight ratios
r = [2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1];

// Define an array with different expansion area ratios.
a = [10, 20, 30, 40];

// Print out table header
printf("#%4s %5s %8s %8s %8s", "r", "A/At", "Is_v,s", "Is_opt,s",
"Is_sl,s");

// Solve the performance problem for each ratio in the array.
for (i=0; i<r.length; ++i) {
    // Assign new O/F weight ratio, replacing pre-configured one.
    p.getPropellant().setRatio(r[i], "O/F");

    // Solve the combustion problem for given O/F ratio.
    p.solve();

    // Calculate performamce for each area ratio in the array.
    for (j=0; j<a.length; ++j) {
        s = p.solveNozzleSection(a[j], "A/At");

        // Print out current O/F weight ratio and calculated specific
        // impulse in vacuum, at optimum expansion, and at sea level.
        printf(" %4.2f %5.2f %8.2f %8.2f %8.2f",
            r[i], a[j],
            s.getIs_v("s"),
            s.getIs("s"),
            s.getIs_H(1, "atm", "s")
        );
    }

    // Prepare the solver for restart.
    p.clearForRestart();
}
```

Propellant Analysis

```
/*************************************************
```

```
 RPA - Tool for Rocket Propulsion Analysis
 Copyright 2009-2011 Alexander Ponomarenko
 Please contact author <contact@lpre.de> or visit
 http://www.propulsion-analysis.com if you need
 additional information or have any questions.

 propellant_analysis.js

*************************************************/

load("resources/scripts/printf.js");

mix = Mixture();
mix.addSpecies("O2(L)", 0.8);     // Add 1st component at it's normal
                                  // temperature and atmospheric pressure
mix.addSpecies("H2(L)", 0.15);    // Add 2nd component at it's normal
                                  // temperature and atmospheric pressure
mix.addSpecies("RP-1", 0, "K", 3, "atm", 0.03); // Add 3rd component at
                                                // it's normal temperature
                                                // and pressure 3 atm
mix.addSpecies("AL(cr)", 0.02);   // Add 4th component at it's
                                  // normal temperature and atmospheric
                                  // pressure

// Total mass fraction of components #2 (RP-1) and #3 (AL(cr))
sf = mix.getFraction(2) + mix.getFraction(3);

// Array with different values of AL(cr) mass fraction
m = Array();
for (i=0; i<=1.0; i+=0.2) {
    m[m.length] = sf*i;
}

// Print out table header
printf("#%6s %6s %8s %8s %8s", "RP-1", "AL(cr)", "Is_v,s", "Is_opt,s",
"Is_sl,s");

for (i=0; i<m.length; ++i) {
    // Change mas fractions of components #2 (RP-1) and #3 (AL(cr))
    mix.setFraction(2, sf - m[i]);
    mix.setFraction(3, m[i]);

    chamber = Chamber(mix);
    chamber.setP(10, "MPa");     // Chamber pressure
    chamber.setFcr(3);           // Nozzle inlet contraction area ratio
    chamber.solve(true);         // finiteChamberSection=true

    nozzleExit = NozzleSectionConditions(chamber, 40, "A/At", true);

    printf(" %6.3f %6.3f %8.2f %8.2f %8.2f",
        mix.getFraction(2),
```

```
            mix.getFraction(3),
            nozzleExit.getIs_v("s"),
            nozzleExit.getIs("s"),
            nozzleExit.getIs_H(1, "atm", "s")
        );
    }
```