RPA-C

Rocket Propulsion Software+Engineering

Version 2.0

User Manual

2021

Table of Contents

Introduction

RPA-C is an easy-to-use tool for the thermodynamic analysis. It features an intuitive graphical user interface with convenient grouping the input parameters and analysis results.

RPA-C utilizes an expandable chemical species library based on NASA Glenn thermodynamic database, that includes data for numerous ingredients and combustion products. With embedded species editor, the users may also easily define new species, or import species from *PROPEP* or *CEA2* databases.

The calculation method is based on robust, proven and industry-accepted Gibbs free energy minimization approach to obtain the combustion composition.

RPA-C is a special version of RPA – Tool for Rocket Propulsion Analysis (http://www.propulsion-analysis.com).

System requirements

The system requirements for running RPA-C are described below.

Operating systems (64-bit edition):

- Microsoft Windows 10 or later

Additional software:

- Microsoft VC++ 2019 run-time libraries for target operating system and architecture are installed
- Qt v.5.x run-time libraries for target operating system

Optional software:

- Scilab 6.x to use Scilab plugin

Hardware requirements:

- RPA-C runs on the minimum standard PC workstation or Laptop PC configurations specified by Microsoft to properly operate the target operating system

Installation

RPA-C is distributed in ZIP package for Windows x86-64 architectures. The package includes the binary files of RPA-C and Qt v.5.x as well as thermodynamic database and examples.

The program does not require special installation procedure: after obtaining the distribution ZIP package, extract all files from the package into selected directory. To uninstall the

program, delete the installation directory.

You may install several different versions of the program and run them in parallel.

RPA-C for Windows depends on MS VC++ 2019 run-time libraries. If your computer does not have it installed, please download and install all required components of this package.

Running RPA-C

Graphical User Interface

You start RPA-C either by double-clicking on an executable file RPA-C.exe in installation directory, or typing `RPA-C.exe` on a command line.

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

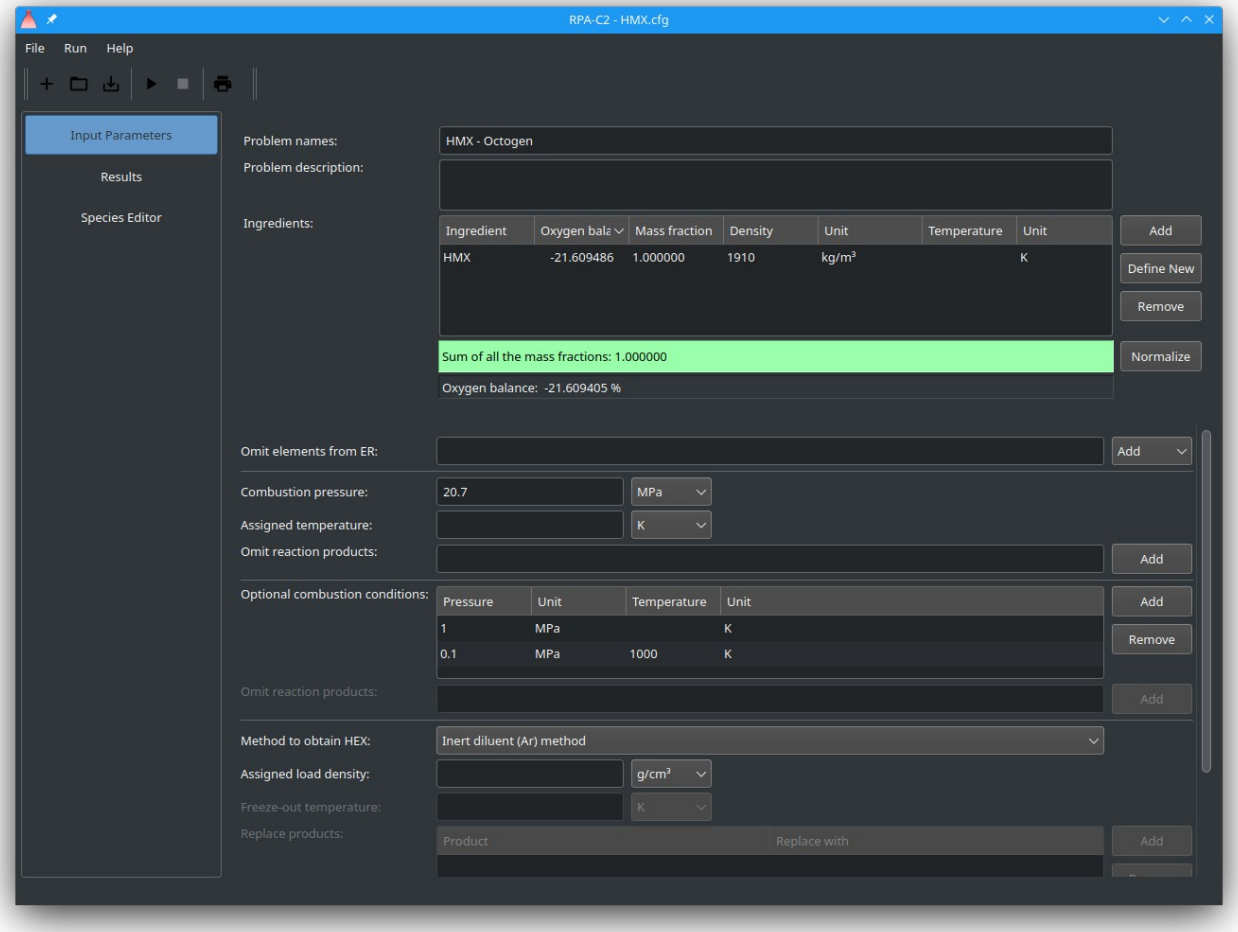
Option	Value	Description
`-t`	File path	To specify the path to the thermodynamic database. Default path is resources/thermo.inp
`-ut`	File path	To specify the path to the user-defined thermodynamic database. Default path is <user-data>/usr_thermo.inp
`-p`	File path	To specify the path to the properties database. Default path is resources/properties.inp
`-up`	File path	To specify the path to the user-defined properties database. Default path is <user-data>/usr_properties.inp
`-i`	File path	To specify the path to the problem configuration file. Default path is path to the last used configuration 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.

The working area consists of three screens: input parameters, results, and species editor. 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.



Main window of RPA-C

Scripting utility

Scripting utility is a tool that can be used to automate the execution of user's problems, or to generate the results in the text format with desired formatting.

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

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

The available command-line arguments are shown below:

Option	Value	Description
`-h` `--help`		Get help screen
`-v` `--version`		Get version of the RPA-C
`--std_thermo`	File path	To specify the path to the thermodynamic database. Optional. Default path is resources/thermo.inp
`-t` `--usr_thermo`	File path	To specify the path to the user-defined thermodynamic database. Optional. Default path is <user-data>/usr_thermo.inp
`--std_properties` `--usr_properties`	File path	To specify the path to the properties database. Optional. Default path is resources/properties.inp
`-p` `--usr_properties`	File path	To specify the path to the user-defined properties database. Optional. Default path is <user-data>/usr_properties.inp
`--input`	File path	To specify the path to the script file to start the utility in batch mode. If not specified, the utility is started in interactive mode.
`--stream`		Execute script from standard input stream.
`--server`		Start scripting engine as a server.
`--port`	Port No	Listen on server specified port. Optional. Default port is 8080

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

Type "exit" or “quit” 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-C-Script.exe --input some_script.js
```

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

Server mode

To start scripting utility in server mode, execute one the following commands:

```
RPA-C-Script.exe --server
```

to listen on default server port 8080, run

```
RPA-C-Script.exe --server --port 80
```

to listen on specified server port (80 in this example).

Now you may connect to the server using Internet browser or any other tool that supports HTTP protocol, for example, CURL (https://curl.se/windows/) or custom tools.

To get the help screen with provided HTTP API, execute the command /about using any internet browser:

```
http://localhost:8080/about
```

Response:

```
RPA-C-Script Server v.2.0.0
Web API

  /evaluate      POST
  /reset         POST
  /log/info      GET
  /log/warn      GET
  /log/error     GET
  /about         GET
```

Example of executing the example script “script-examples/mixture.js” file with CURL:

```
curl -i --request POST
      --header "Content-Type: text/plain"
      --header "Accept: text/plain"
      -F "file=@script-examples/mixture.js"
      http://localhost:8080/evaluate
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 967

Initial mixture (built-in printout)
*********************************

Initial mixture (script printout)
*********************************
-------------------------------------------------------------
 Propellant Mixture           Mass            Mole
                           fractions       fractions
-------------------------------------------------------------
        NH4CLO4(cr)      0.7000000       0.4452445
```

```
              AL(cr)      0.2000000       0.5539381
      HTPB+Curative       0.1000000       0.0008174
-------------------------------------------------------------
              Total:      1.0000000       1.0000000
-------------------------------------------------------------
Exploded chemical formula:
   based on 1 mole:     (N)0.449331 (H)2.580360 (CL)0.445245 (O)1.791604
(AL)0.553938 (C)0.536191
    based on 100 g:     (N)0.006013 (H)0.034529 (CL)0.005958 (O)0.023974
(AL)0.007413 (C)0.007175

Equivalence ratio: 0.630733 (omitted: CU)
Equivalence ratio: 0.630733 (omitted: )
```

In consecutive calls you may use any object (both standard Java and RPA-C specific), created in one of the previous calls.

For example:

```
curl -i --request POST
      --header "Content-Type: text/plain"
      --header "Accept: text/plain"
      -F "file=@script-examples/mixture.js"
      http://localhost:8080/evaluate


curl -i --request POST
      --header "Content-Type: text/plain"
      --header "Accept: text/plain"
      --data 'omit = ["CU"]'
      http://localhost:8080/evaluate


curl -i --request POST
      --header "Content-Type: text/plain"
      --header "Accept: text/plain"
      --data 'mix.getEquivalenceRatio(omit)'
      http://localhost:8080/evaluate
```

Response, returning the equivalence ratio “0.6307330246120119” for the mixture object created after the first request:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 18

0.6307330246120119
```

To reset the current context removing all created JavaScript objects, use the command “/reset”:

```
curl -i --request POST
      --header "Content-Type: text/plain"
      --header "Accept: text/plain"
      http://localhost:8080/reset
```

Using server mode with Octave scripts

GNU Octave (https://www.gnu.org) is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab.

Octave provides API to access web-based REST API, which can be used to create additional helper Octave functions:

```
rpa_url = "http://localhost:8080";
global evaluate_url = [rpa_url "/evaluate"];
global reset_url = [rpa_url "/reset"];

function result = rpa_execute(script)
    global evaluate_url;
    result = urlread(evaluate_url, "post", {"script", script});
endfunction

function rpa_reset()
    global reset_url;
    urlread(reset_url, "post", "");
endfunction

function script = rpa_load_script(fname)
    script = fileread(fname);
endfunction
```

Example of executing the RPA-C script from Octave script:

```
s = rpa_load_script("script-examples/mixture.js");
rpa_execute(s);

rpa_execute("omit = ['CU']");
er = rpa_execute("mix.getEquivalenceRatio(omit)");

rpa_reset();
```

In addition to Octave functions, the helper RPA-C JavaScript functions can be used to assist the preparation the problem for the analysis from Octave.

For example, here is the JavaScript file “helper.js” with helper functions:

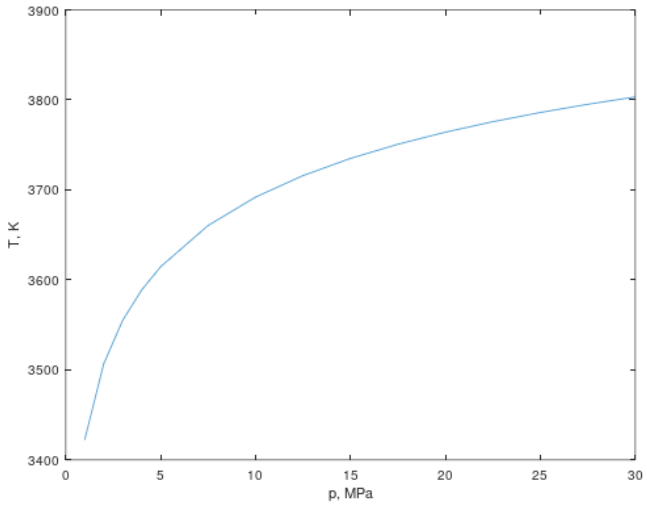
```
function prepareMixture(){
    var mix = new Mixture();
    for (var i=0; i<arguments.length; i++) {
        var arg = arguments[i];
```

```
            mix.addSpecies(arg[0], arg[1]);
        }
        return mix;
    }

    function solve(mix, p, p_units) {
        var c = new Combustor(mix, true, true);
        c.setP(p, p_units);
        c.solve(true, false);
        return {"e":c.getEquilibrium(), "d":c.getDerivatives()};
    }
```

the usage of these functions in Octave script (together with Octave helper functions):

```
s2=rpa_load_script("helper.js");
rpa_execute(s2);

# Prepare mixture ans store it in server context as variable "mix"
rpa_execute("mix = prepareMixture(['NH4CLO4(cr)', 0.7], ['AL(cr)', 0.2],
['HTPB+Curative', 0.1])");

p_array = [1, 2, 3, 4, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30];
T_array = []; # Vector to collect calculated temperatures

for p = p_array
    # Result "res" is stored in the server context
    # and can be reused to obtain calculated values
    rpa_execute(sprintf("res = solve(mix, %f, 'MPa')", p));
    # Obtain required value from stored result
    T_array(end+1) = str2num(rpa_execute("res.e.getT('K')"));
end

plot(p_array, T_array);
xlabel ("p, MPa");
ylabel ("T, K");

rpa_reset();
```

and the result:



Configuration Files

The 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.

The program provides the following user interface control elements to handle the configuration files:

- Item “File” in menu bar with following items:
  - “New” – to create new problem configuration
  - “Open” – to load the problem configuration from existing configuration file
  - “Save” – to save current problem configuration into associated configuration file (if exists), or into new or existing configuration file (if problem configuration is new)
  - “Save As...” – to save current problem configuration into new or existing configuration file
  - List of up to 10 last used configuration files for quick switching between the problems
- Button “New” on toolbar which is a short-cut for menu item “New”
- Button “Open” on toolbar which is a short-cut for menu item “Open”

Application supports the following keyboard short-cuts:

- “Ctrl+N” – a short-cut for menu item “New”
- “Ctrl+O” – a short-cut for menu item “Open”
- “Ctrl+S” – a short-cut for menu item “Save”

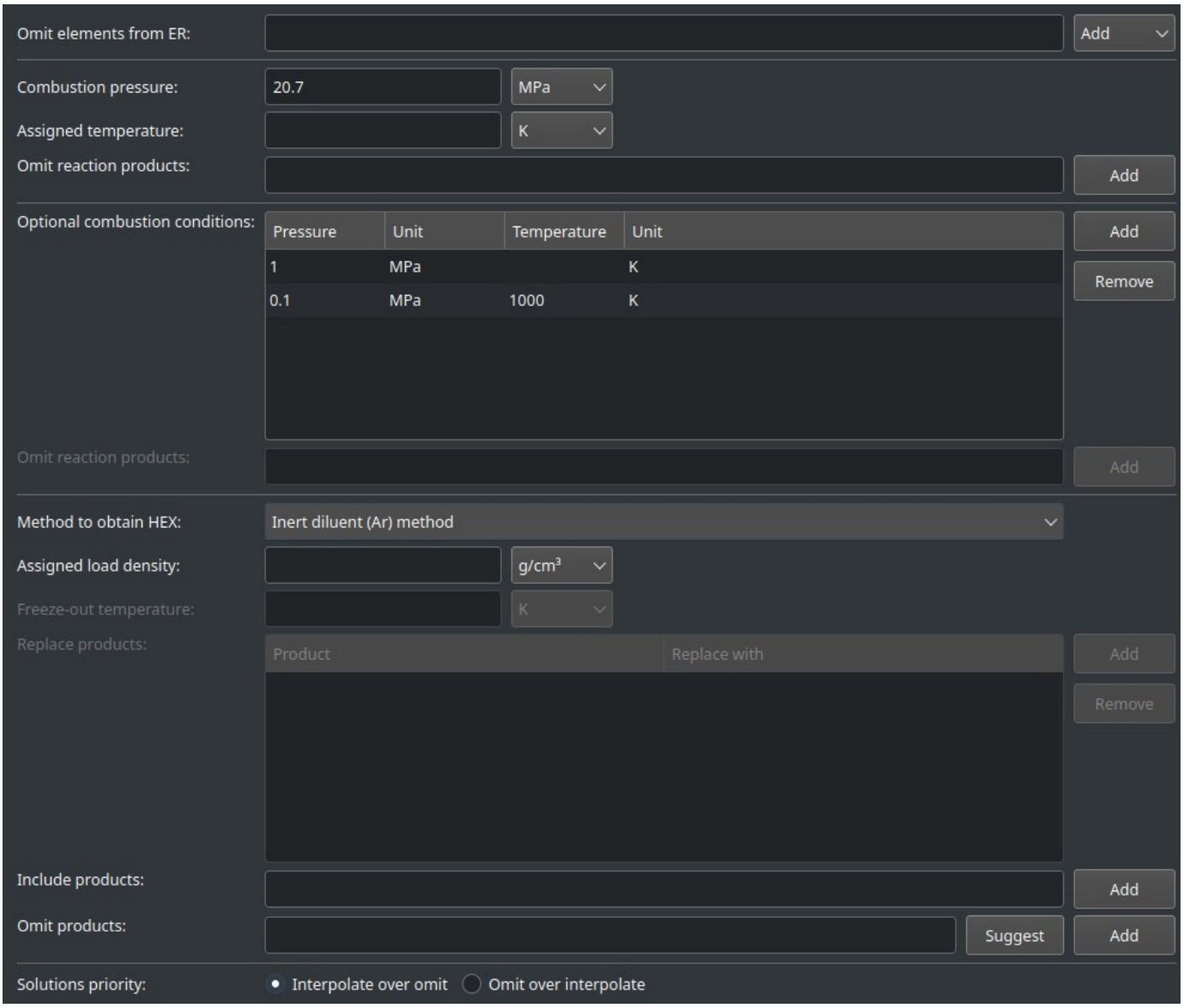
When you try to create new configuration or load existing configuration from the file, the program requests the confirmation to allow the re-loading the existing configuration.

The program displays the names of 10 last used configuration files in the menu “File”. The last used configuration file is loaded automatically when the program is started next time on the same workstation under the same user account.

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

Input Parameters

Screen “Input Parameters” is used to define the input parameters of the problem.



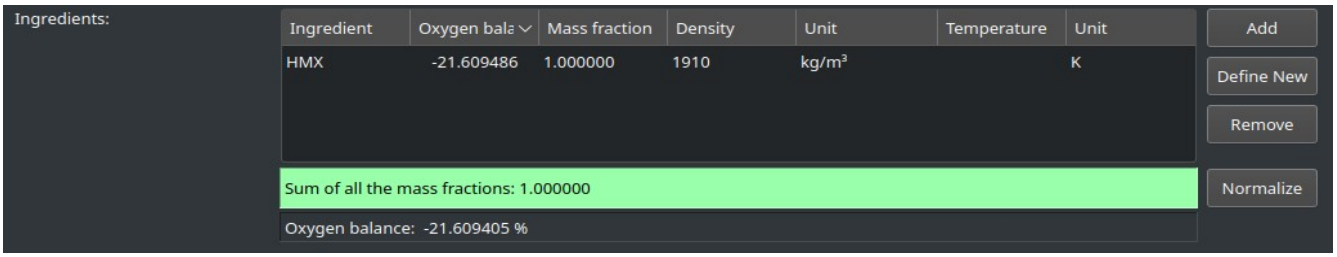
Input parameters

Problem Identification

Problem identification includes “Problem Name” and “Problem description”. Both parameters are optional and can be used to help the user to identify the configuration.

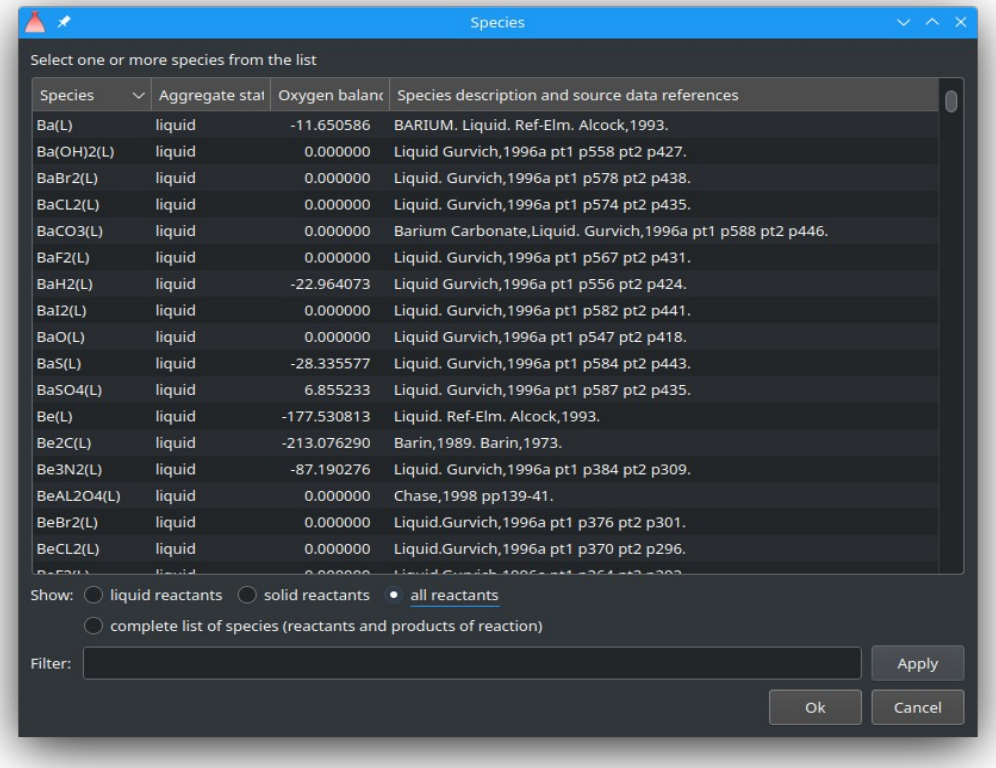
Ingredients Specification

List of ingredients contains one or more ingredients, displayed on the single row. To add ingredient to the list, click the button **Add** at the right side of the list. To remove the selected ingredient from the list, click the button **Remove**.



List of ingredients

After clicking on button **Add**, the dialog window *"Species"* appears:



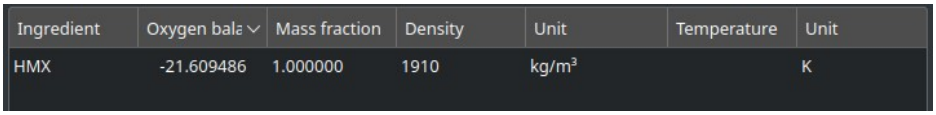
Species Database

You can filter the list in the dialog window, using a simple text match or an regular expressions. The filter pattern is applied to columns “Species“ and “Species description...“ of the table.

Mark the check box *”Show complete list of species (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 ingredients.

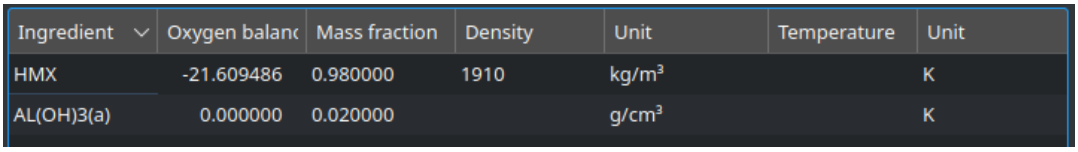
Select one or more species on the list and click the button **OK**. Click the button **Cancel** if you want to leave without adding any species.

The ingredient on the list features 4 parameters: ingredient name, mass fraction of the ingredient in the mixture, density of the ingredient, and temperature of the ingredient:

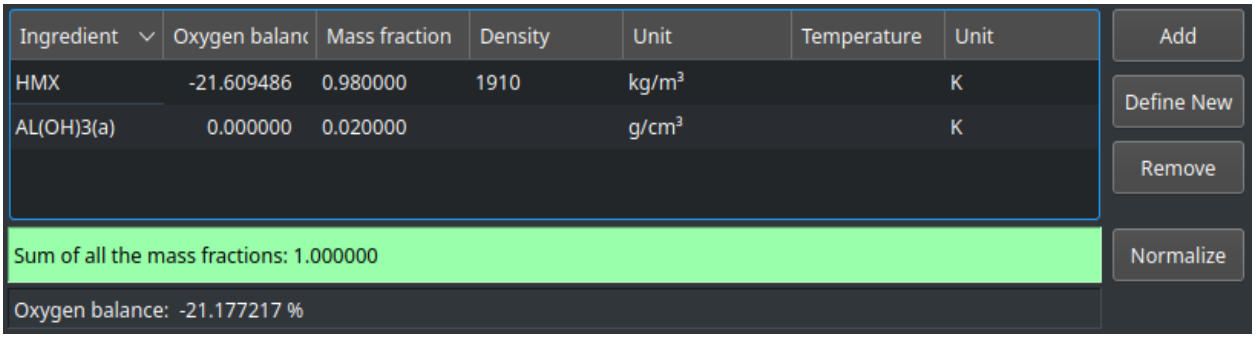


The last two parameters (density and temperature) are optional.

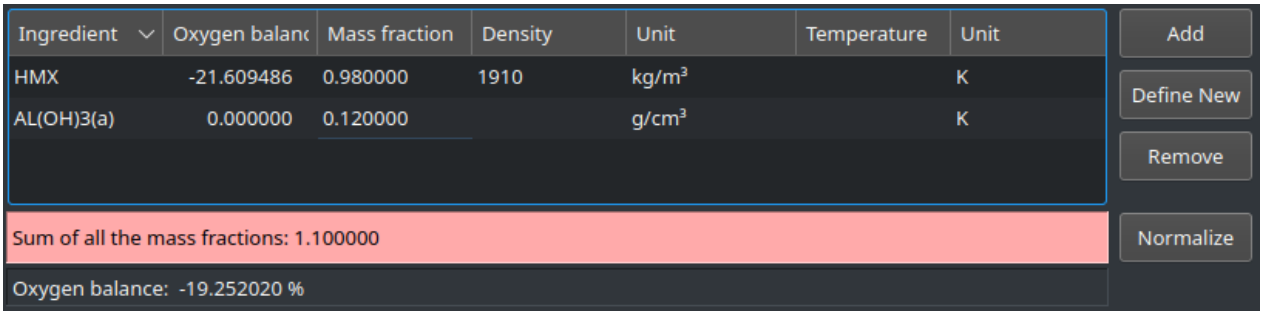
When composing the mixture from several ingredients, the sum of all the mass fractions of ingredients has to be equal to 1. To change the mass fraction for the species, double-click on the corresponding cell, enter the new value and press Enter button (or click away):



The 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:

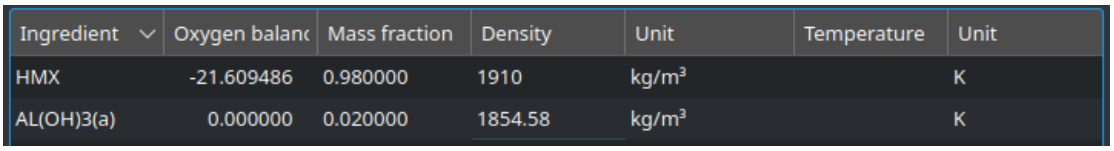


If the sum is incorrect, the background color is light-red:



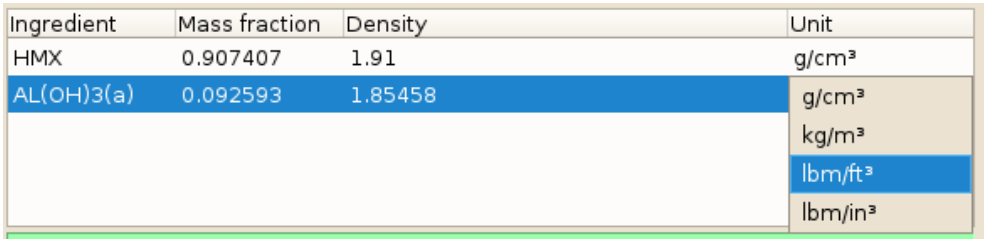
You can use the button “Normalize” to adjust the mass fractions to get the sum of all fractions 1.00.

The density is an optional parameter. If not specified, the density is obtained from species database. To specify the density of the ingredient, double-click on the corresponding cell, enter the new value and then press Enter button (or click away):



Ingredient	Oxygen balanc	Mass fraction	Density	Unit	Temperature	Unit
HMX	-21.609486	0.980000	1910	kg/m³		K
AL(OH)3(a)	0.000000	0.020000	1854.58	kg/m³		K

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



By default, all chemical elements available in the mixture will be used to calculate the equivalence ratio. If you want to omit some of elements from this calculation, you should provide the list of chemical elements to be omitted.

You may either add the elements choosing them from the drop-down list on the right side or type the comma- or semicolon-separated list directly in the input box:

Combustion Conditions

The program obtains the chemical equilibrium for one of more defined combustion conditions as described in paper "RPA: Design Tool for Liquid Rocket Engine Analysis" by Alexander Ponomarenko.

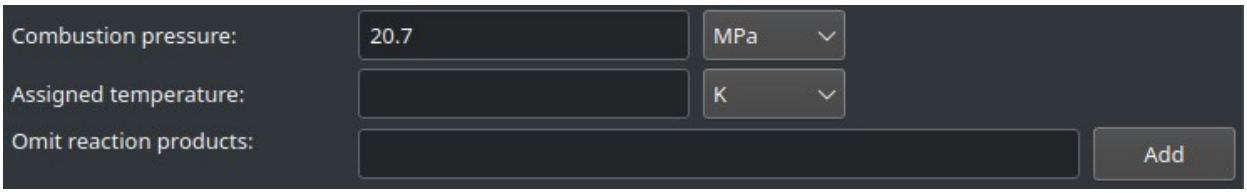
Main Conditions

Main conditions include the following parameters:

- Combustion pressure.
- Assigned temperature.
- List of reaction products to be omitted from the calculation of chemical equilibrium.

Combustion pressure is a mandatory parameter. The assigned temperature is an optional parameter: when specified, the combustion process proceeds at the constant pressure and temperature *(p,T)=const*. Otherwise, the process proceeds at the constant pressure and enhalpy *(p,H)=const*.

Use the list "Omit reaction products" to exclude possible species from the calculation of the chemical equilibrium.



Main conditions

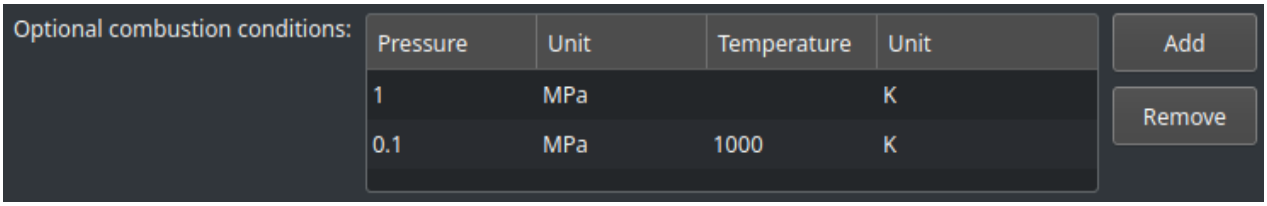
To add new product to the list, click the button **Add** at the right side of the list and select the product dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window). To remove the selected species from the list, click the button **Remove**. To replace the already added product double-click it and select other product in dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window).

Optional Conditions

Within the single run, the program may calculate chemical equilibrium for several different combustion conditions. These conditions are optional.

To add new conditions to the list, click the button **Add** at the right side of the list. To remove the selected conditions from the list, click the button **Remove**.

Each conditions configuration on the list features 2 parameters: combustion pressure and assigned temperature:



To change the pressure or temperature, double-click on the corresponding cell, enter the new value and press Enter button or click away. To change the pressure or temperature units, double-click on the corresponding cell, select the desired unit on the list, and then press Enter button or click away.

Combustion pressure is a mandatory parameter. The assigned temperature is an optional parameter: when specified, the combustion process proceeds at the constant pressure and temperature *(p,T)=const*. Otherwise, the process proceeds at the constant pressure and enhalpy *(p,H)=const*.

HEX Configuration

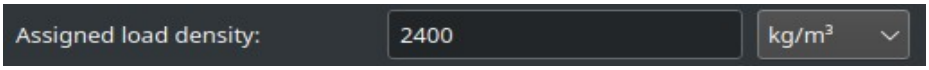
The program calculates the Heat of Explosion (HEX) as described in paper "Techniques for the Estimation of Heats of Explosion (HEX) Using Thermochemical Codes" by Robert A. Fifer and Jeffrey B. Morris using two of described methods:

- Exact method
- Inert diluent (Ar) method

Each of methods supports its own set of input parameters. All parameters which are not supported by particular method are disabled.

Both methods require the density of the mixture, which is obtained as follows:

- as explicitly assigned load density of the mixture:



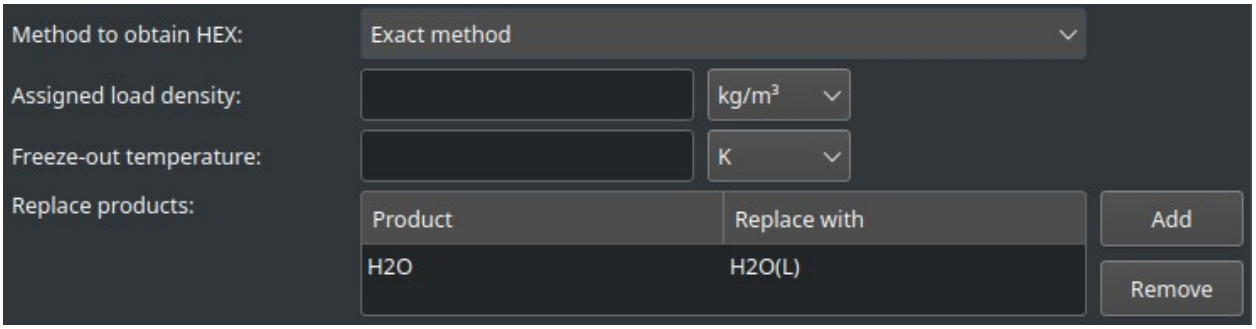
- if assigned load density is not specified, the density is calculated using mass fractions and density of ingredients available on the list of ingredients (see chapter *Ingredients Specification*)
- if ingredient density is not available on the list of ingredients, the programs obtains it from the database (see chapter *Species Editor*)

If program cannot obtain the density of the mixture using steps above in provided order, then the HEX calculation is not performed.

Exact Method

Exact method supports the following input parameters:

- Assigned load density of ingredients.
- Freeze-out temperature.
- List of reaction products to be replaced by other products before calculation of HEX.



Configuration of exact method

If freeze-out temperature is not defined, the program automatically assigns the temperature 900 K.

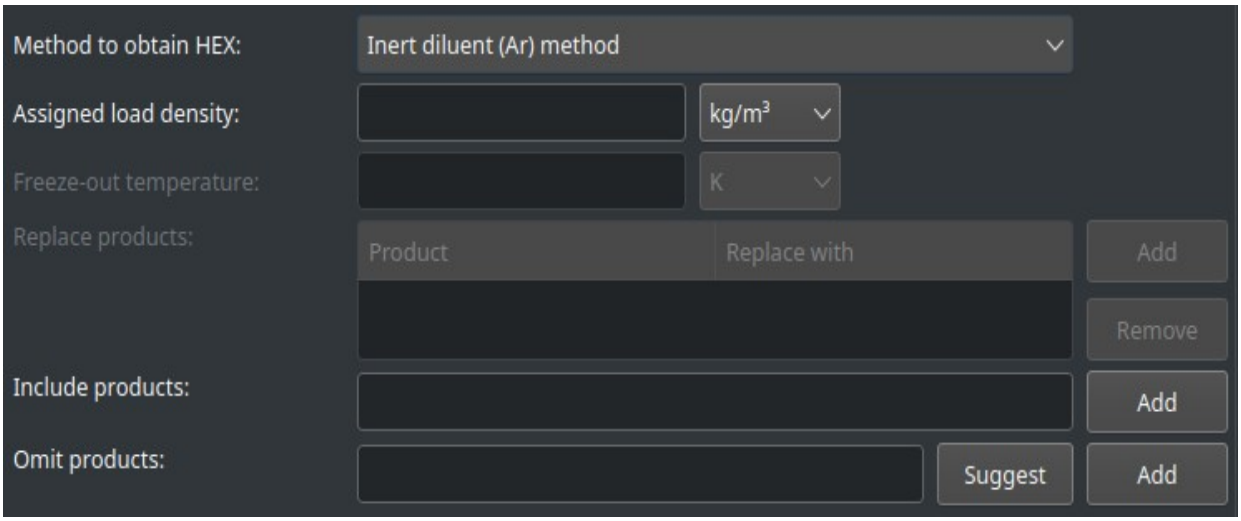
The list “Replace products” automatically includes the replacement “H2O->H2O(L)”.

To add new replacement to the list, click the button **Add** at the right side of the list and select the product dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window). After that provide the replacement by double-click the empty cell in the column “Replace with”. To remove the replacement from the list, click the button **Remove**. To modify the already added product or replacement, double-click the correspomding cell and select other product in dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window).

Inert Diluent Method

Inert diluent method supports the following input parameters:

- Assigned load density of ingredients.
- List of reaction products to be considered in the calculation. Note that all other species will be excluded from the calculation of HEX.
- List of reaction products to be omitted from the calculation of HEX.



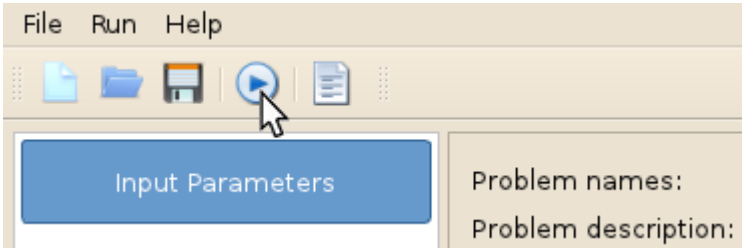
Configuration of inert diluent method

To add new product to any of the lists, click the button **Add** at the right side of the corresponding list and select the product dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window). To remove the selected species from the list, click the button **Remove**. To replace the already added product double-click it and select other product in dialog window Species Database (see chapter *Ingredients Specification* for more information about this dialog window).

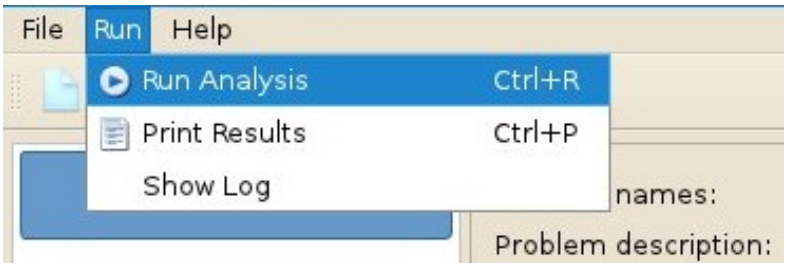
Running Analysis

To start the analysis,

- click the button “Run” on the toolbar:



- or choose the item “Start Analysis” in the menu bar “Run”:



After successful finishing the analysis, the program automatically switches to the screen “Results”.

If analysis could not be performed, the program displays a dialog window with relevant information (error message or message about incomplete input data).

The analysis log is accessible under the menu item “Show log” in menu “Run”. The dialog window displays the records for 3 different log levels in separate tabs:

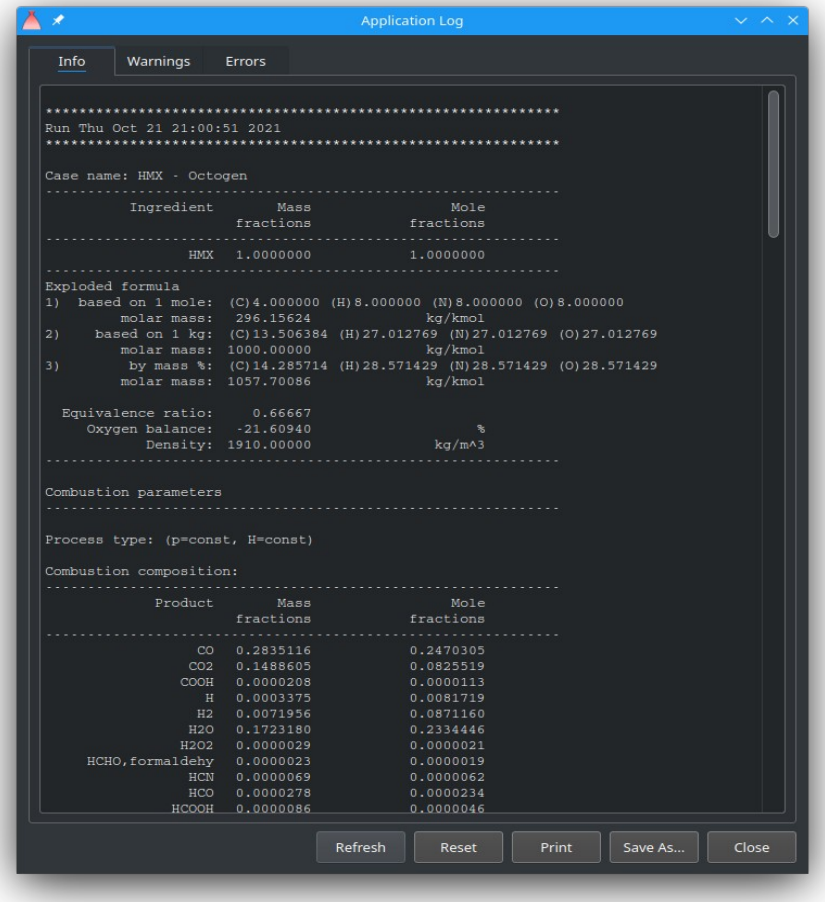
- Info – to display the log records with the level “Information”
- Warnings – to display the records with the level “Warnings”
- Errors – to display the records with the level “Error”

When thermodynamic analysis is performed without any error, the tab “Info” contains the following information:

- Date and time the analysis was performed
- List of ingredients with mass and mole fractions of each ingredient
- Exploded formula and molecular weight based on 1 mole of initial mixture
- Exploded formula and molecular weight based on 100 g of initial mixture
- Equivalence ratio
- Mixture density
- List of species in combustion composition
- Combustion parameters

The current contents of the log can be printed out or saved into the file in plain text format using the buttons **Print** and **Save As...** correspondingly.

By pressing the button **Reset** the user may delete the current contents of the log.




```
************************************************************
Run Thu Oct 21 21:00:51 2021
************************************************************

Case name: HMX - Octogen
------------------------------------------------------------
          Ingredient       Mass                Mole
                         fractions           fractions
------------------------------------------------------------
                 HMX   1.0000000           1.0000000
------------------------------------------------------------
Exploded formula
1)  based on 1 mole:  (C)4.000000 (H)8.000000 (N)8.000000 (O)8.000000
        molar mass:   296.15624              kg/kmol
2)    based on 1 kg:  (C)13.506384 (H)27.012769 (N)27.012769 (O)27.012769
        molar mass:  1000.00000              kg/kmol
3)        by mass %:  (C)14.285714 (H)28.571429 (N)28.571429 (O)28.571429
        molar mass:  1057.70086              kg/kmol

  Equivalence ratio:     0.66667
     Oxygen balance:   -21.60940                 %
            Density:  1910.00000              kg/m^3
------------------------------------------------------------

Combustion parameters
------------------------------------------------------------

Process type: (p=const, H=const)

Combustion composition:
------------------------------------------------------------
             Product       Mass                Mole
                         fractions           fractions
------------------------------------------------------------
                  CO   0.2835116           0.2470305
                 CO2   0.1488605           0.0825519
                COOH   0.0000208           0.0000113
                   H   0.0003375           0.0081719
                  H2   0.0071956           0.0871160
                 H2O   0.1723180           0.2334446
                H2O2   0.0000029           0.0000021
    HCHO,formaldehy   0.0000023           0.0000019
                 HCN   0.0000069           0.0000062
                 HCO   0.0000278           0.0000234
               HCOOH   0.0000086           0.0000046
```

Analysis log

Aspects of HEX calculation

Interpolation of results obtained using Exact Method

When using Exact Method, the HEX cannot be obtained from direct calculation at specific freeze-out temperature (either assigned by User or automatically assigned by application) due to issues with obtaining equilibrium at this temperature.

At the same time, HEX can still be obtained for lower or higher temperatures.

In such case, the program obtains HEX for several points below and above assigned freeze-out temperature, and calculates the HEX at assigned temperature as an interpolated value, using either linear interpolation (if only 2 points are available), or polynomial interpolation (if more than 2 points are available).

When HEX value is obtained from interpolation, the program adds the word “interpolated” into the row “Heat of explosion” on the screen Results.

Suggest products to be omitted using Inert diluent method

When using inert diluent method, the obtained HEX value can be unrealistic low. As suggested in work *“Techniques for the Estimation of Heats of Explosion (HEX) Using Thermochemical Codes”* (http://www.dtic.mil/docs/citations/ADA268584), in such cases it is necessary to exclude many species that would otherwise be predicted in unreasonable amounts, or to allow only significant products.

For that purpose, the program provides a button “Suggest” which automatically adds products onto the list “Omit products” using following approach:

- obtain equilibrium for the mixture which consists of 1 part of the original mixture and 999 parts of Ar (inert diluent);
- scan the list of products of reaction and put the products onto the list of omitted products they meet following criteria:
  - chemical formula contains more than one unique chemical elements
  - number of at least one atom in the chemical formula is greater than 2

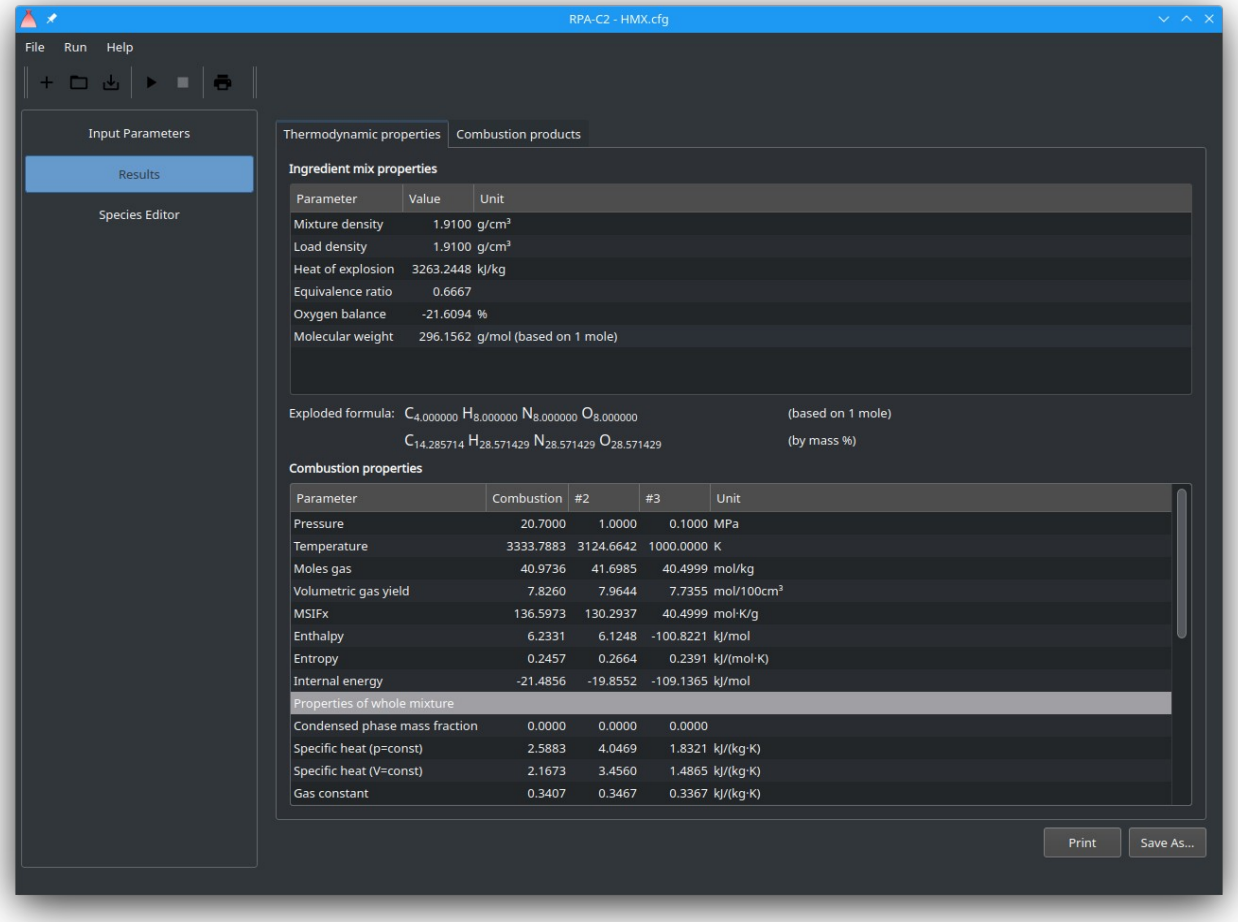
  For example, the following products will be added onto the list of omitted products: CH4, C2H6, C2H5OH, NH3; while the following products won't be added onto the list: CO, CO2, H2, H2O.
- For systems which contains chemical elements C, H and N, always add following products onto the list: C(gr), CH4, and NH3 (as suggested in work *“Techniques for the Estimation of Heats of Explosion (HEX) Using Thermochemical Codes”*).

As the automatic suggestion might be sub-optimal, the user may modify the suggested list before starting the solution.

Results of Analysis

The screen “Results” consists of two tabs:

- Thermodynamic properties
- Combustion products



Screen „Results“ with selected tab „Thermodynamic properties“

Tab “Thermodynamic properties” contains two tables: table “Propellant Properties” and table “Combustion properties”.

The table “Ingredient mix properties” displays the following parameters:

- Density of the mixture (either assigned load density or calculated density)
- Load density
- Calculated heat of explosion
- Calculated equivalence ratio of the mixture with the list of omitted chemical elements
- Calculated oxygen balance
- Molecular weight of the mixture
- Exploded formula

Ingredient mix properties

Parameter	Value	Unit
Mixture density	1.9100	g/cm³
Load density	1.9100	g/cm³
Heat of explosion	3263.2448	kJ/kg
Equivalence ratio	0.6667	
Oxygen balance	-21.6094	%
Molecular weight	296.1562	g/mol (based on 1 mole)

Exploded formula: $C_{4.000000}\ H_{8.000000}\ N_{8.000000}\ O_{8.000000}$ (based on 1 mole)

$C_{14.285714}\ H_{28.571429}\ N_{28.571429}\ O_{28.571429}$ (by mass %)

Table „Ingredient mix properties“

The table “Combustion properties” displays the calculated thermodynamic properties of combustion products.

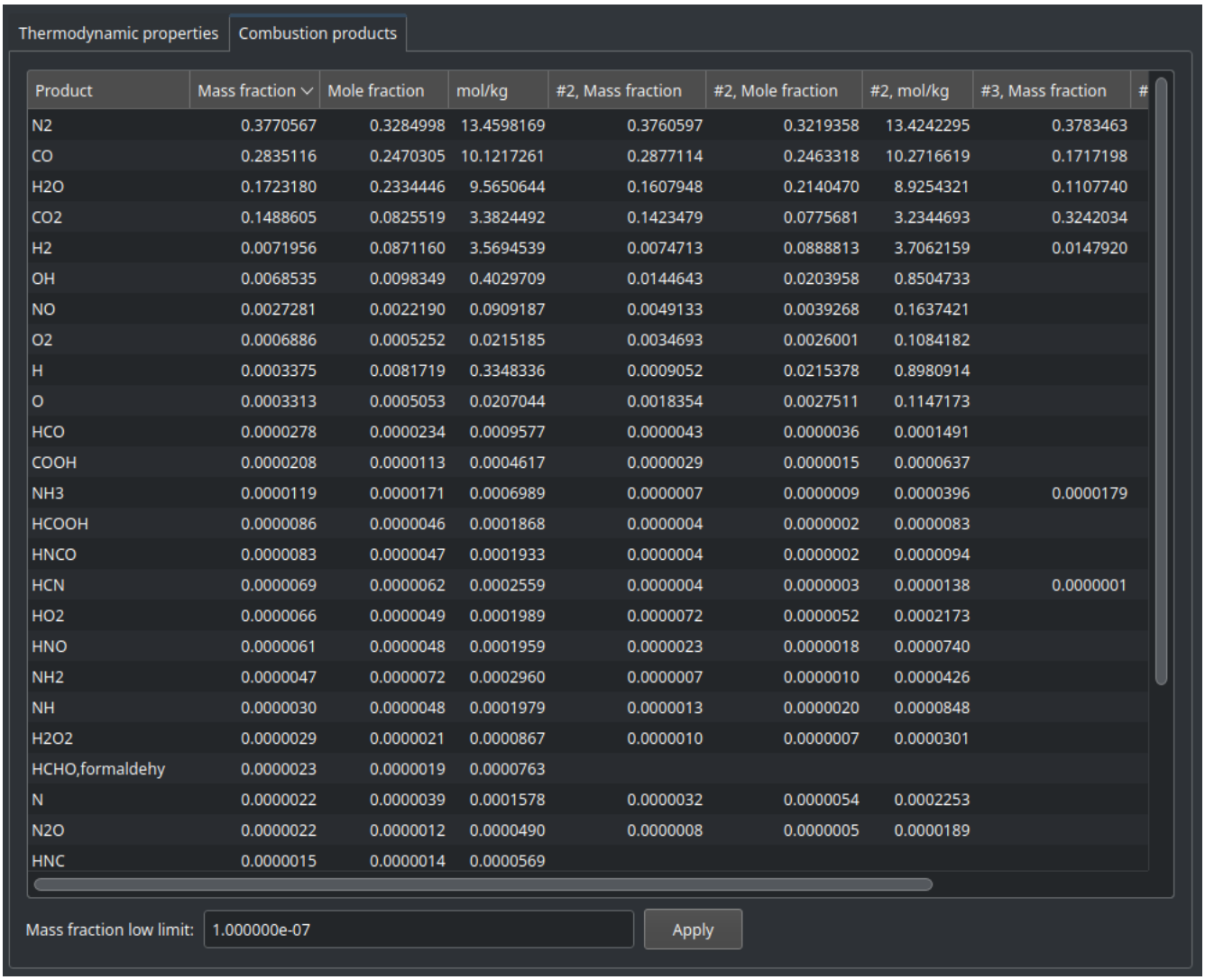
Combustion properties

Parameter	Combustion	#2	#3	Unit
Pressure	20.7000	1.0000	0.1000	MPa
Temperature	3333.7883	3124.6642	1000.0000	K
Moles gas	40.9736	41.6985	40.4999	mol/kg
Volumetric gas yield	7.8260	7.9644	7.7355	mol/100cm³
MSIFx	136.5973	130.2937	40.4999	mol·K/g
Enthalpy	6.2331	6.1248	-100.8221	kJ/mol
Entropy	0.2457	0.2664	0.2391	kJ/(mol·K)
Internal energy	-21.4856	-19.8552	-109.1365	kJ/mol
Properties of whole mixture				
Condensed phase mass fraction	0.0000	0.0000	0.0000	
Specific heat (p=const)	2.5883	4.0469	1.8321	kJ/(kg·K)
Specific heat (V=const)	2.1673	3.4560	1.4865	kJ/(kg·K)
Gas constant	0.3407	0.3467	0.3367	kJ/(kg·K)
Molecular weight	24.4060	23.9817	24.6914	
Gamma	1.1942	1.1710	1.2325	
Isentropic exponent	1.1871	1.1529	1.2314	
Density	0.0182	0.0009	0.0003	g/cm³
Sonic velocity	1161.1516	1117.5867	643.9262	m/s
Properties of gas mixture only				

Table „Combustion properties“

Tab “Combustion products” contains one table with all combustion products which mass fraction is above the defined low limit. The low limit is defined by user in separate input box below the table and activated by pressing the button **Apply**:

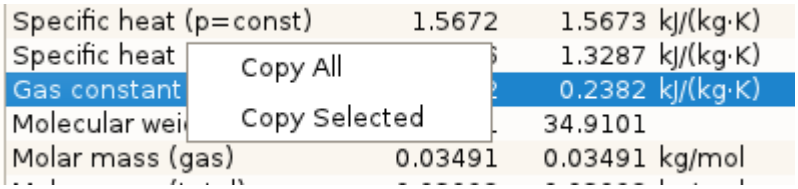
The table contains the list of reaction products in the first columns and 3 columns for each defined combustion conditions: “Mass fraction”, “Mole fraction”, and “Moles in 100g”.



Product	Mass fraction	Mole fraction	mol/kg	#2, Mass fraction	#2, Mole fraction	#2, mol/kg	#3, Mass fraction	#
N2	0.3770567	0.3284998	13.4598169	0.3760597	0.3219358	13.4242295	0.3783463	
CO	0.2835116	0.2470305	10.1217261	0.2877114	0.2463318	10.2716619	0.1717198	
H2O	0.1723180	0.2334446	9.5650644	0.1607948	0.2140470	8.9254321	0.1107740	
CO2	0.1488605	0.0825519	3.3824492	0.1423479	0.0775681	3.2344693	0.3242034	
H2	0.0071956	0.0871160	3.5694539	0.0074713	0.0888813	3.7062159	0.0147920	
OH	0.0068535	0.0098349	0.4029709	0.0144643	0.0203958	0.8504733		
NO	0.0027281	0.0022190	0.0909187	0.0049133	0.0039268	0.1637421		
O2	0.0006886	0.0005252	0.0215185	0.0034693	0.0026001	0.1084182		
H	0.0003375	0.0081719	0.3348336	0.0009052	0.0215378	0.8980914		
O	0.0003313	0.0005053	0.0207044	0.0018354	0.0027511	0.1147173		
HCO	0.0000278	0.0000234	0.0009577	0.0000043	0.0000036	0.0001491		
COOH	0.0000208	0.0000113	0.0004617	0.0000029	0.0000015	0.0000637		
NH3	0.0000119	0.0000171	0.0006989	0.0000007	0.0000009	0.0000396	0.0000179	
HCOOH	0.0000086	0.0000046	0.0001868	0.0000004	0.0000002	0.0000083		
HNCO	0.0000083	0.0000047	0.0001933	0.0000004	0.0000002	0.0000094		
HCN	0.0000069	0.0000062	0.0002559	0.0000004	0.0000003	0.0000138	0.0000001	
HO2	0.0000066	0.0000049	0.0001989	0.0000072	0.0000052	0.0002173		
HNO	0.0000061	0.0000048	0.0001959	0.0000023	0.0000018	0.0000740		
NH2	0.0000047	0.0000072	0.0002960	0.0000007	0.0000010	0.0000426		
NH	0.0000030	0.0000048	0.0001979	0.0000013	0.0000020	0.0000848		
H2O2	0.0000029	0.0000021	0.0000867	0.0000010	0.0000007	0.0000301		
HCHO,formaldehy	0.0000023	0.0000019	0.0000763					
N	0.0000022	0.0000039	0.0001578	0.0000032	0.0000054	0.0002253		
N2O	0.0000022	0.0000012	0.0000490	0.0000008	0.0000005	0.0000189		
HNC	0.0000015	0.0000014	0.0000569					

Table „Combustion products“

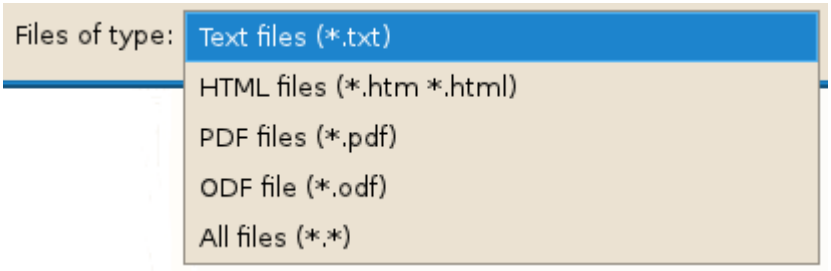
The contents of any of 3 tables can be copied into the operating system clipboard using context menu. The user may decide to copy the selected rows only, or all rows in the current table:



The contents of all of 3 tables can be printed out or saved into the file using buttons **Print** and

Save As... in the right-bottom corner of the window. The program may save the results in one of the following formats:

- plain text format
- HTML format
- PDF format
- ODF format (Open Document Format for office applications)



Supported file formats to store the results

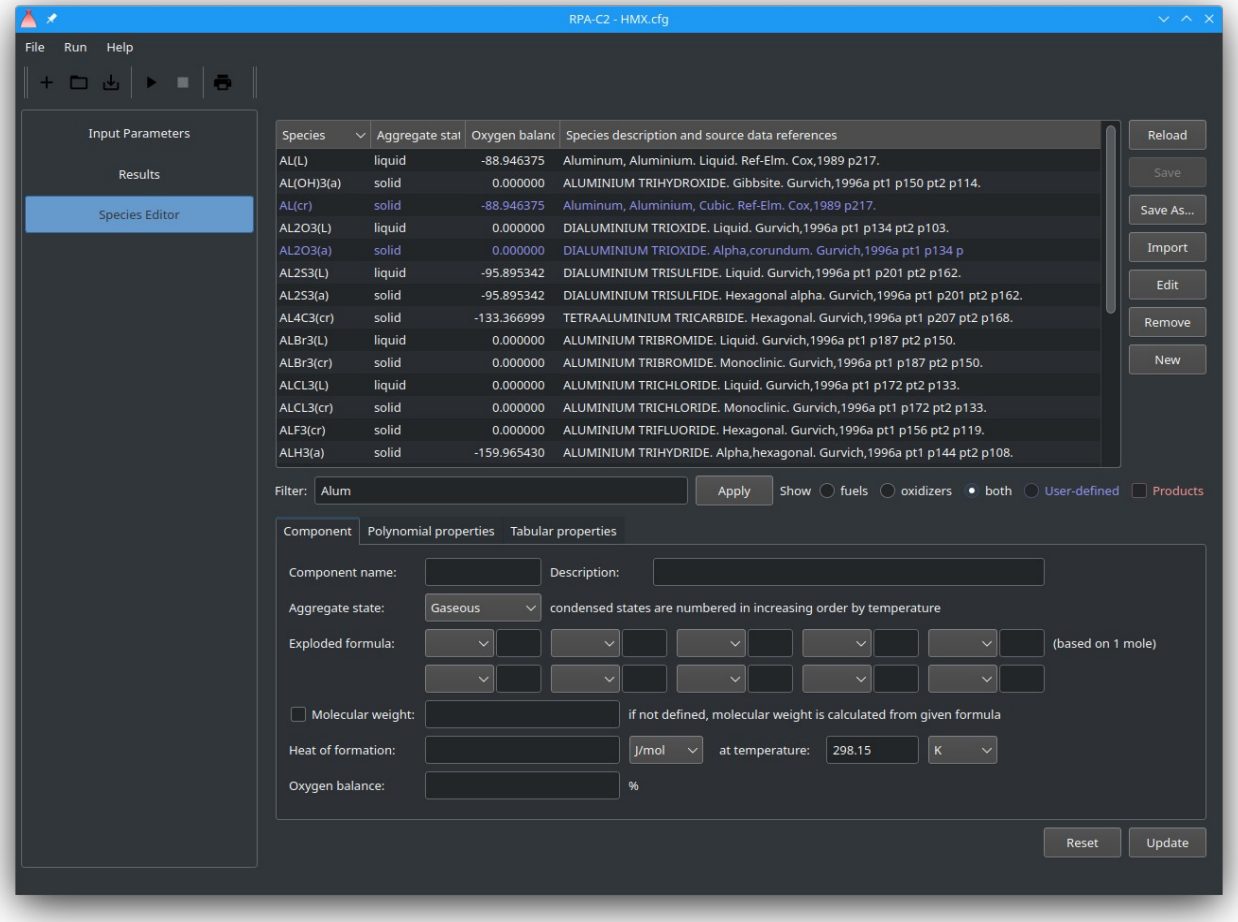
Species Editor

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

The program distribution package contains 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.

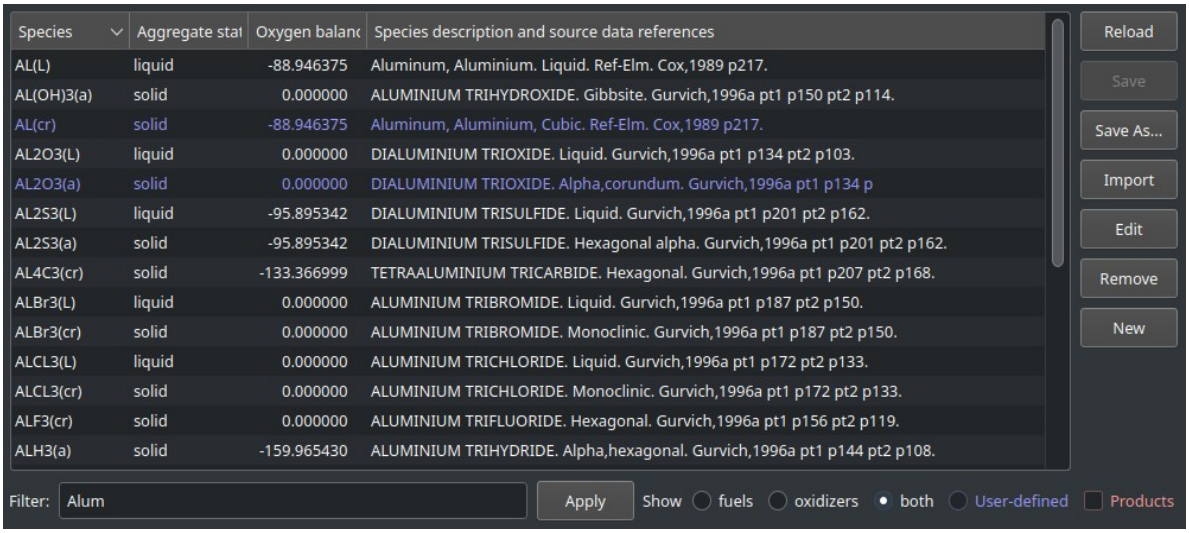
The user can define own species and save it into two additional database files `resources/usr_thermo.inp` and `resources/usr_properties.inp`. These files are not shipped within standard RPA-C distribution packages, and will not be rewritten after program update.

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.



Screen „Species Editor“

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



The user 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 *"Products"* 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 ingredients.

The filter pattern is applied to both columns of the table “Species” and “Species description”.

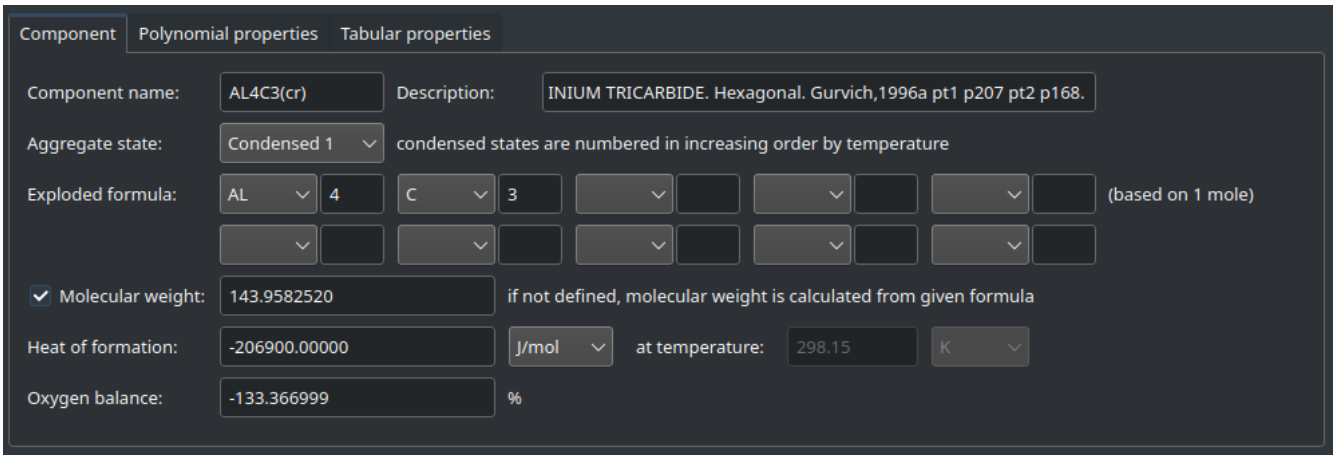
The control buttons at the right side can be used as follows:

- 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: 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**.

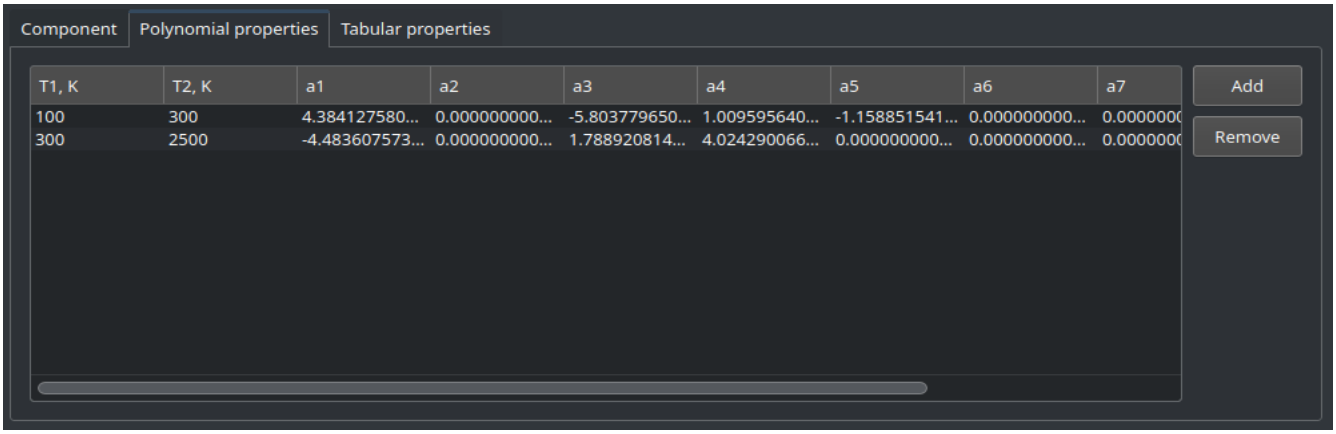
The tab *Component* displays the information about component, its aggregate state, chemical formula, molecular weight, heat of formation, and the temperature the heat of formation is defined for.

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.

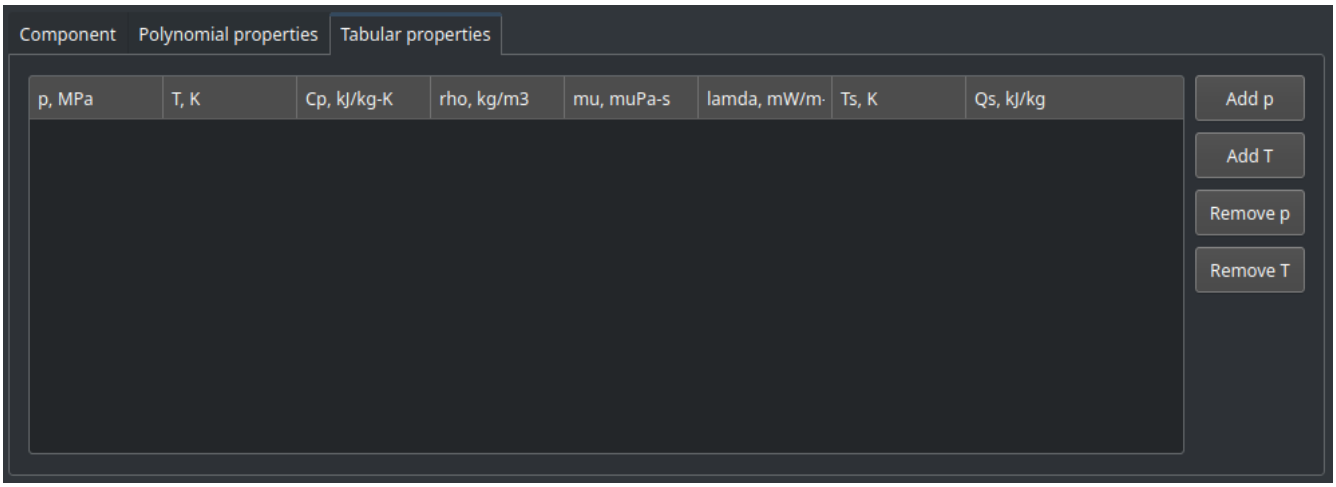
The chemical formula is given as an exploded formula with up to 10 chemical elements, followed by its molecular weight. The heat of formation (enthalpy) can be given in one one the units: `J/mol`, `cal/mol`, `kJ/kg`, `kcal/kg`, `Btu/lbm`, `kcal/lbm`. The heat of formation is followed by the temperature (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.



Polynomial properties

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.



Tabular properties

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
```

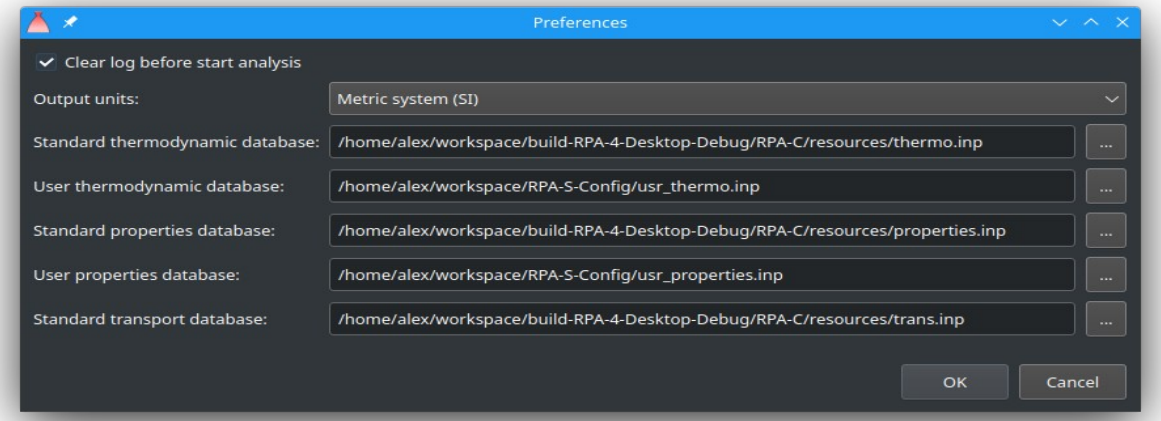
The minimalist data for the component consists of at least two rows:

```
!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 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
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

Input and Output Units

The user 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)

Name	Value	Description
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·s)" to "kg/(m^2·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
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-C, 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 mixtures of ingredients
- run typical combustion problems `(p,H)=const`, `(p,S)=const`, `(p,T)=const`, `(v,U)=const`, `(v,S)=const`, `(v,T)=const`
- calculate HEX
- 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.
`Themro API`	API for searching the thermodynamic database, obtaining species properties, preparing mixtures of ingredients.
`Reaction API`	API 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.
`Analysis API`	API for running analysis defined in prepared configuration files and obtains results.

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

Function	Description
	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 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 %.

Function	Description

	%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]

`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.

Function	Description

	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");
	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 loading, manipulation and writing configuration files.

Object ConfigFile

This object represents the configuration file and provides the functions to access different sub-configurations.

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:

Function	Parameters	Description
		c = ConfigFile("examples/test-1.cfg");

`read()`	-	Reads the assigned file.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();

`read(String path)`	`File path`	Reads the specified file.
		
		Example:
		
		c = ConfigFile();
		c.read("examples/test-1.cfg");

`write()`	-	Writes the configuration into assigned file.
		
		Example:
		
		c = ConfigFile("examples/RD-275.cfg");
		c.setName("test-1");
		c.write();

`write(String path)`	`File path`	Writes the configuration into specified file.
		
		Example:
		
		c = ConfigFile();
		c.setName("test-1");
		c.write("examples/test-1.cfg");

`Number getVersion()`	-	Returns the version of configuration file.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		print("Version: "+c.getVersion());

`setName(String name)`	`Engine name`	Assigns the engine name.

Function	Parameters	Description

		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.setName("RD-275");
		c.write();

`String getName()`	-	Returns the engine name.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		print("Problem name: "+c.getName());

`setInfo(String info)`	`Description`	Assigns the description.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.setInfo("Test case #1");
		c.write();

`String getInfo()`	-	Returns the assigned description.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		print("Case description: "+c.getInfo());

`Object getGeneralOptions()`	-	Returns the associated GeneralOptions object.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		Object g = c.getGeneralOptions();

`Object getIngredients()`	-	Returns the associated Ingredients object.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		Object n = c.getIngredients();

Function	Parameters	Description

`Object getCombustionConditions()`	-	Returns the associated CombustionConditions object.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		Object cc = c.getCombustionConditions();

`Number getCombustionOptionalConditionsSize()`	-	Returns the number of defined optional combustion conditions.
`Object getCombustionOptionalConditions(Number n)`	`n – index of optional combustion parameters`	Returns the object CombustionConditions which represents the optional combustion conditions with given number “n”.
`Object setCombustionOptionalConditions()`	-	Creates and returns new optional combustion conditions (object CombustionConditions).
`clearCombustionOptionalConditions()`	-	Removes all optional combustion conditions.
`Object getHexConditions()`	-	Returns the associated HEXConditions object.

Object GeneralOptions

This object represents the general options of the configuration.

Function	Parameters	Description

`Boolean isMultiphaseFlow()`	-	Returns true, if multiphase flow and phase transitions should be considered.
		
		Example:
		
		c = ConfigFile("examples/test-1.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		if (g.isMultiphaseFlow()) {
		print("Multiphase flow flag is on");
		}

Function	Parameters	Description

`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/test-1.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/test-1.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/test-1.cfg");
		c.read();
		Object g = c.getGeneralOptions();
		g.setIons(false);

Object CombustionConditions

This object represents the combustion conditions and can be used to manipulate both the main combustion conditions and the optional combustion conditions. In the last case the functions for manipulating the list of omitted products are ignored amd have no effect.

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

Function	Parameters	Description
	"MPa")	
Boolean isTemperature()	-	Retuens *true* if temperature is assigned.
Number getTemperature(String unit)	Temperature unit (one of "K", "F", "C")	Returns assigned temperature in givem unit, or in "K" if unit is not specified.
setTemperature(Number t, String unit)	t – temperature unit - temperature unit (one of "K", "F", "C")	Sets assigned temperature specified in specified unit, or in "K" if unit is not specified.
Number getOmitProductsSize()	-	Returns the size of the list of omitted products.
String getOmitProduct(Number n)	Index of requested product on the list	Returns the name of omitted products under given number.
addOmitProduct(String nane)	Name of product	Adds the given product name to the list of omitted products.
ClearOmitProducts()	-	Clears the list of omitted products.

Object HEXConditions

This object represents the configuration of HEX calculation.

Depending on the configured method, part of available functions is ignored and has no effect.

Function	Parameters	Description
String getType()	-	Returns the configured method of HEX calculation: "exact method" or "inert diluent method".
setType(String type)	The method name: "exact method" or "inert diluent method"	Assigns the method of HEX calculation.
Boolean isAssignedLoadDensity()	-	Returns true if load density of mixture is assigned. Available for both Exact method and Inert Diluent method.
Number getAssignedLoadDensity(String unit)	Density unit (one of "kg/m^3" (default), "g/cm^3", "lbm/ft^3", "lbm/in^3")	Returns load density of the misture in specified unit, or in "kg/m^2" if unit is not specified. Available for both Exact method and Inert Diluent method.
setAssignedLoadDensity(Number rho, String unit)	rho – assigned mixtre density	Assigns mixture density in specified unit, or in "kg/m^3" if unit is not specified.

Function	Parameters	Description
	unit - density unit (one of "kg/m^3" (default), "g/cm^3", "lbm/ft^3", "lbm/in^3")	Available for both Exact method and Inert Diluent method.
deleteAssignedLoadDensity()	-	Remove assigned mixtuer density. Available for both Exact method and Inert Diluent method.
Boolean isFreezeOutTemperature()	-	Returns true if freeze-out temperature is assigned. Available for Exact method.
Number getFreezeOutTemperature(String unit)	Temperature unit (one of "K", "F", "C")	Returns assigned freeze-out tempareture in specified unut, or in "K" if unit is not specified. Available for Exact method only.
setFreezeOutTemperature(Number t, String unit)	t – temperature unit - temperature unit (one of "K", "F", "C")	Assigns freeze-out temperature in specified unit, or in "K" if unit is not specified. Available for Exact method only.
deleteFreezeOutTemperature()	-	Removes assigned freeze-out temperature. Available for Exact method only.
Number getReplaceProductsSize()	-	Returns the size of the list with product replacements pairs. Available for Exact method only.
String getReplaceProductKey(Number n)	Index of requested replacement pair	Returns the name of the product under given number. Available for Exact method only.
String getReplaceProductValue(Number n)	Index of requested replacement pair	Returns the name of the replacement product under given number. Available for Exact method only.
addReplaceProduct(String name, String replaceWith)	name – name of product replaceWith – name of replacement product	Adds new replacement pair to the list. Available for Exact method only.
clearReplaceProducts()	-	Removes all replacement pairs from the list.

Function	Parameters	Description
		Available for Exact method only.
`Number getIncludeProductsSize()`	`-`	Returns the size of the list with products to be included into the calculation. Available for Inert Diluent method only.
`String getIncludeProduct(Number n)`	`Index of requested product on the list`	Returns the name of product under given number. Available for Inert Diluent method only.
`addIncludeProduct(String name)`	`Name of product`	Adds specified product to the list. Available for Inert Diluent method only.
`clearIncludeProducts()`	`-`	Removes all products from the list. Available for Inert Diluent method only.
`Number getOmitProductsSize()`	`-`	Returns the size of the list with products to be omitted from the calculation. Available for Inert Diluent method only.
`String getOmitProduct(Number n)`	`Index of requested product on the list`	Returns the name of product under given number. Available for Inert Diluent method only.
`addOmitProduct(String name)`	`Name of product`	Adds specified product to the list. Available for Inert Diluent method only.
`clearOmitProducts()`	`-`	Removes all products from the list. Available for Inert Diluent method only.

Object Component

This object represents the mixture component (ingredient) which can be added into the object Ingredients.

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.
`setRho(Number rho, String unit)`	`rho - density of ingredient`	Assigns density of ingredient in specified unit, or in "kg/m^3" if unit is

Function	Parameters	Description
	unit - density unit (one of "kg/m^3" (default), "g/cm^3", "lbm/ft^3", "lbm/in^3")	not specified.
Number getRho(String unit)	temperature unit (one of "kg/m^3" (default), "g/cm^3", "lbm/ft^3", "lbm/in^3")	Returns ingredient density in specified unit, or in "kg/m^3" if unit is not specified..

Object Ingredients

This object represents the mixtuer of components (ingredients).

Function	Parameters	Description
Number getSpeciesListSize()	-	Returns the number of ingredients in the mixture.
Object getSpecies(Number n)	Index of requested igredient in the mixture	Returns the object *Component* which represents the ingredient under specified number.
addSpecies(Component c)	Object Component which represents the ingredient	Adds the ingredient into the mixture.
Object getMixture()	-	Returns the object *Mixture* which includes all ingredients.

Thermo API

Thermo API is indented for searching the species in the thermodynamic database, obtaining species properties, and preparing mixtures.

Object Database

The object represents the species database. The object has no constructor, and cannot be instantiated explicitly. Instead, the Scripting Utility provides the global variable “database”.

Function	Parameters	Description
Object getSpecies(String name)	Name of species	Returns *Species* object. Example: print(database.getSpecies("H2O2"));

Object Species

The object represents the species (ingredients or reaction products). The object has no constructor, and cannot be instantiated explicitly. The instance of the object can be obtained from objects *Database* or *Mixture*.

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`
		Checks whether the specified temperature is valid. If valid, returns specified temperature. Otherwise

Function	Parameters	Description
	unit (one of "K", "F", "C")	throws an exception.
Number getM()		Returns molecular weight of species.
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(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")	Returns Gibbs energy at specified temperature in desired unit.

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

Object Mixture

The object represents the mixture ingredients or combustion products. The object can be instantiated using constructor or obtained from other objects.

Function	Parameters	Description
Mixture()	-	Constructor. Creates an empty mixture.
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 rho, String rhoUnit, Number r)	name - species name rho - density rhoUnit – density unit (one of "kg/m^3", "g/cm^3", "lbm/ft^3", "lbm/in^3") 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.
Array getFormula(String base)	"g" if number of elements in exploded formula have to be based on 100 g of mixture. Any other string except "g", if	Returns exploded chemical formula of the mixture. Each element of the array represents one chemical element (atom) with properties “name” and “n”.

Function	Parameters	Description
	number of elements in exploded formula have to be based on 1 mole of mixture.	
Number getM()	-	Returns molecular weight of mixture.
Number getH(String unit)	enthalpy unit (one of "J/mol" (default), "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar enthalpy of mixture in desired unit.
Number getU(String unit)	enthalpy unit (one of "J/mol" (default), "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns specific or molar internal energy of mixture in desired unit.
Number getRho(String unit)	one of "kg/m^3" (default), "g/cm^3", "lbm/ft^3", "lbm/in^3"	Returns volumetric density 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.
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")	Prints 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 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 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.

Function	Parameters	Description
`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")`
	
	`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(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-`
		Returns specific or molar enthalpy at specified temperature in desired unit.

Function	Parameters	Description
	mol", "J/kg", "kJ/kg", "Btu/lbm")	
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", "F", "C") unit - Gibbs energy unit (one of "J/mol", "Btu/lb-mol", "J/kg", "kJ/kg", "Btu/lbm")	Returns Gibbs energy at specified temperature in desired unit.

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	Assigns pressure and temperature of combustion and switches the solving

Function	Parameters	Description
	p_unit - pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa") T - temperature T_unit - temperature unit (one of "K", "F", "C")	reaction problem to type (p,T)=const.
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") S - enropy S_unit - enropy unit (one of "J/(mol K)", "Btu/(lb-mol R)", "kJ/(mol K)")	Assigns combustion pressure and enropy of propellant and switches the solving reaction problem to type (p,S)=const.
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,	v - specific volume	Assigns combustion specific volume

Function	Parameters	Description
Number U, String U_unit)	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")	(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 value is false.	Solves the prepared problem. 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.
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.

Function	Parameters	Description
`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")`	Prints 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 capacity 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)",`	Returns specific heat or molar heat capacity of reaction products at constant volume in desired unit.

Function	Parameters	Description
	"Btu/(lb-mol R)")	
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)", "mpoise" or "x10^-4 lb/(ft s)")	Returns viscosity of reacting mixture.
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

Function	Parameters	Description
`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.

Analysis API

Analysis API is indented for executing configured problem and obtaining performance parameters.

Object Combustor

Function	Parameters	Description
`Combustor(Object mix)`	`mix – Mixture object which represents the mixture of ingredients`	Constructor. Creates *Combustor* object using given object *Mixture*.

`setP(Number p, String unit)`	`p - pressure`
	
	`pressure unit (one of "Pa", "psi", "atm", "bar", "at", "MPa")`
		Assigns combustion chamber pressure. Enthalpy of mixture of ingredients is automatically calculated using mixture assigned in constructor.

`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")`
		Assigns combustion chamber pressure and enthalpy of mixture of ingredients.

Function	Parameters	Description
	H - enthalpy H_unit - enthalpy unit (one of "J/mol", "Btu/lb-mol", "kJ/mol")	
omit(Array species)	species – array of species to be omitted from calculation	
solve(Boolean startWithCondensed, Boolean withTransportProperties)	startWithCondensed – true of false withTransportProperties – true or false	Solves the chemical equilibrium problem. 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.
Object getEquilibrium()	-	Returns object *Reaction* for solved chemical equilibrium problem.
Object getDerivatives()	-	Returns object *Derivatives* for solved chemical equilibrium problem.

Object CombustionAnalysis

Function	Parameters	Description
CombustionAnalysis()	-	Constructor. Creates *CombustionAnalysis* object.
run(ConfigFile conf, String units)	conf – object ConfigFile units – name of unuts system to be used for printing out the results into the log (one of "SI" or "US")	Executes the problem configured in given configuration file.
Object getMixture()	-	Returns the object *Mixture* which represents the initial mixture of ingredients. Returns null if executed befire the function *run*.
Object getCombustor(Number n)	index	Returns the object *Combustor* for solved chemical equilibrium. Index 0 corresponds to *Combustor*

Function	Parameters	Description
		created for main combustion parameters Index>0 corresponds to *Combustor* created for optional combastion parameters. Returns null if executed befire the function *run*.
`Number getHEX(String unit)`	`HEX unut (one of "J/kg", "kJ/kg", "kcal/kg", "Btu/lbm")`	Returns calculated HEX in specified unt, or in "J/kg" if unit is not specified. Returns null if executed befire the function *run*.

Scripting examples

All examples can be found in the directory “script-examples” within installation directory of the RPA-C.

analysis.js

```
// Load configuration file
c = new ConfigFile("examples/HMX.cfg");
c.read();

// Create and run combustion analysis
ca = new CombustionAnalysis();
ca.run(c);

if (ca.getCombustorsListSize()>0) {
    printf("Initial mixture:\n");
    ca.getMixture().print();

    tUnit = "K";
    printf("T   = %10.5f %s\n",
      ca.getCombustor(0).getEquilibrium().getT(tUnit), tUnit);

    hexUnit = "kJ/kg";
    printf("HEX = %10.5f %s\n", ca.getHEX(hexUnit), hexUnit);
} else {
    printf("Could not solve!\n");
}
```

mixture.js

```
// Prepare the mixture of ingredients.
mix = new Mixture();
mix.addSpecies("NH4CLO4(cr)", 0.7);
mix.addSpecies("AL(cr)", 0.2);
mix.addSpecies("HTPB+Curative", 0.9, "g/cm^3", 0.1);

printf("Initial mixture (built-in printout)\n");
printf("***********************************\n");
mix.print();

printf("\nInitial mixture (script printout)\n");
printf("***********************************\n");

printf("-------------------------------------------------------------------\n");
printf("%20s\t %10s\t %10s\n", "Propellant Mixture", "Mass", "Mole");
printf("%20s\t %10s\t %10s\n", "", "fractions", "fractions");
printf("-------------------------------------------------------------------\n");
sum1 = 0;
sum2 = 0;
for (i=0; i<mix.size(); ++i) {
    s = mix.getSpecies(i);
    massFraction = mix.getFraction(i, "mass");
    moleFraction = mix.getFraction(i, "mole");
    sum1 += massFraction;
    sum2 += moleFraction;
    printf("%20s\t %10.7f\t %10.7f \n", s.getName(), massFraction,
moleFraction);

}
printf("-------------------------------------------------------------------\n");
printf("%20s\t %10.7f\t %10.7f\n", "Total:", sum1, sum2);
printf("-------------------------------------------------------------------\n");

printf("Exploded chemical formula:\n");

printf("%20s\t ", "based on 1 mole:");
f = mix.getFormula("m");
for (var i=0; i<f.length; i++) {
    printf("(%s)%8.6f ", f[i].name, f[i].n);
}
printf("\n");

printf("%20s\t ", "based on 100 g:");
f = mix.getFormula("g");
for (var i=0; i<f.length; i++) {
    printf("(%s)%8.6f ", f[i].name, f[i].n);
}
printf("\n\n");

omit = ["CU"];
printf("Equivalence ratio: %8.6f (omitted: %s)\n",
mix.getEquivalenceRatio(omit), omit);
```

```
omit = [];
printf("Equivalence ratio: %8.6f (omitted: %s)\n",
mix.getEquivalenceRatio(omit), omit);

printf("\n");
```

combustor.js

```
// Prepare the mixture of ingredients
mix = new Mixture();
mix.addSpecies("NH4CLO4(cr)", 0.7);
mix.addSpecies("AL(cr)", 0.2);
mix.addSpecies("HTPB+Curative", 0.9, "g/cm^3", 0.1);

printf("Initial mixture:\n");
mix.print();

// Solve problem (p,H)=const using object Combustor
printf("\n\nSolve problem (p,H)=const\n");
c1 = new Combustor(mix, true, true);
c1.setP(20.7, "MPa");
c1.solve(true, false);
c1.getEquilibrium().print("SI");
c1.getDerivatives().print("SI");

// Solve problem (p,T)=const using object Combustor
p = 20.7;
T = 1400;
printf("\n\nSolve problem (p,T)=const at p=%4.1f MPa T=%8.3f K\n", p, T);
c2 = new Combustor(mix, true, true);
c2.setPT(p, "MPa", T, "K");
c2.solve(true, false);
c2.getEquilibrium().print("SI");
c2.getDerivatives().print("SI");

// Solve problem (p,H)=const using object Combustor
printf("\n\nSolve problem (p,H)=const at p=%4.1f MPa\n", p);
c3 = new Combustor(mix, true, true);
c3.setP(p, "MPa");
c3.solve(true, false);
c3.getEquilibrium().print("SI");
c3.getDerivatives().print("SI");
```

combustor_nested_analysis.js

```
// Prepare the mixture of ingredients
mix = new Mixture();
mix.addSpecies("NH4CLO4(cr)", 0.7);
```

```
mix.addSpecies("AL(cr)", 0.2);
mix.addSpecies("HTPB+Curative", 0.9, "g/cm^3", 0.1);

printf("Initial mixture:\n");
mix.print();

// Nested analysis

// Array of pressure values in MPa
p = [10, 15, 20];

// Array of temperature values in K
// "-1" means the temperature won't be assigned
// (see the code below)
T = [-1, 1000, 1400, 1800];

for (var i=0; i<p.length; i++) {
    printf("Pressure p=%10.5f MPa\n", p[i]);

    for (var j=0; j<T.length; j++) {
        c = new Combustor(mix, true, true);

        if (T[j]>0) {
            printf("Temperature T=%10.5f K\n", T[j]);
            c.setPT(p[i], "MPa", T[j], "K");
        } else {
            c.setP(p[i], "MPa");
        }

        c.solve(true, false);

        c.getEquilibrium().print("SI");
        c.getDerivatives().print("SI");

        printf("\n*****************************************************\n");
    }

}
```

reaction_products.js

```
c = new ConfigFile("examples/HMX.cfg");
c.read();

ca = new CombustionAnalysis();
ca.run(c);

// Obtain object "Reaction" (aka "Equilibrium")
r = ca.getCombustor(0).getEquilibrium();
```

```
// Obtain mixture of reaction products
products = r.getResultingMixture();

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

sum1 = 0;
sum2 = 0;

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

    massFraction = products.getFraction(i, "mass");
    moleFraction = products.getFraction(i, "mole");;

    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\n",
        s.getName(),
        massFraction,
        moleFraction,
        s.getCondensed()
    );

}

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

custom_log.js

```
// Open the  file "log.txt" in the mode "w" ("write")
var f = new File("log.txt", "w");

// Define variable with the name of configuration file
configName = "examples/HMX.cfg";
```

```
// Open configuration file
c = new ConfigFile(configName);
c.read();
f.printf("# Configuration file: %s\n\n", configName);

// Prepare and run combustion analysis
ca = new CombustionAnalysis();
ca.run(c);

if (ca.getCombustorsListSize()>0) {
    combustor = ca.getCombustor(0);

    r = combustor.getEquilibrium();
    products = r.getResultingMixture();

    unit = "MPa";
    f.printf("p   = %10.5f %s\n",
      combustor.getEquilibrium().getP(unit), unit);

    unit = "K";
    f.printf("T   = %10.5f %s\n",
      combustor.getEquilibrium().getT(unit), unit);

    unit = "kJ/kg";
    f.printf("HEX = %10.5f %s\n", ca.getHEX(unit), unit);

    f.printf("\n# %13s %9s %9s %4s\n",
      "Name", "Mass Frac", "Mole Frac", "Cond");

    sum1 = 0;
    sum2 = 0;

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

        massFraction = products.getFraction(i, "mass");
        moleFraction = products.getFraction(i, "mole");;

        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;
        }

        f.printf("%15s %9.7f %9.7f %4d\n",
            s.getName(),
            massFraction,
```

```
            moleFraction,
            s.getCondensed()
        );

    }

    f.printf("%15s %9.7f %9.7f\n",
        "Summ:",
        sum1,
        sum2
    );

}

// Close the file
f.close();
```

Scilab Plugin

The Scilab plugin is used to run combustion analysis from third-party application Scilab.

Configuration of environment

To use RPA-C plugin for Scilab, the RPA-C dynamic libraries have to be available from the Scilab process. There are two options to achieve that:

- Start Scilab (either GUI or console application) from the installation directory of RPA-C: in Windows command-line console (CMD), go to the installation directory of RPA-C, and start Scilab from there by typing the name of the Scilab executable file an dpressing ENTER

- Add the path to RPA-C installation directory (e.g. “C:\Software\RPA-C”) to the environmental variable PATH.

Using the API

To work properly, the Scilab script has to include the following lines in the very beginning:

```
exec loader.sce
RPAInit()
```

Where the file "`loader.sce`" is a part of RPA-C distribution (you can find it in the root directory of the RPA-C), and "RPAInit()" is am initialization function of the plugin.

When used without parameters (like in two lines above), it will use the paths to thermodynamic libraries, configured in the GUI version of RPA-C (see dialog window Help->Preferences).
It can also accept the following string **Parameters:**

```
RPAInit(stdThermoPath, stdPropertiesPath, stdTransportPropertiesPath,
usrThermoPath, usrPropertiesPath)
```

Components

API provides a set of functions to create/get and work with the RPA-C objects.

Most of the functions to work with the RPA-C object accept the reference to the object as a first parameter.

Configuration API

Configuration API is indented for loading, manipulation and writing configuration files. The configuration file can be created in memory and used within RPA-C plugin without saving as a file in a file system of the operating system.

ConstructConfigFile

Create new configuration object with default values in a memory (no real file is created).

```
ConstructConfigFile()
```

Parameters:

n.a.

Returned value:

Reference to configuration object

Example:

```
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "test.cfg");
```

ConfigFile_read

Read the configuration parameters from the specified file.

```
ConfigFile_read(cfg, path)
```

Parameters:

cfg	Reference to configuration object
path	File path

Example:

```
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "test.cfg");
```

ConfigFile_write

Write the configuration parameters to the specified file, rewriting the file if already exists.

```
ConfigFile_write(cfg, path)
```

Parameters:

cfg	Reference to configuration object
path	File path

Example:

```
cfg = ConstructConfigFile();
ConfigFile_write(cfg, "test.cfg");
```

ConfigFile_fromString

Read the configuration parameters from the JSON string.

```
ConfigFile_fromString(cfg, json)
```

Parameters:

cfg	Reference to configuration object
json	JSON representation of the configuration

Example 1:

```
cfg = ConstructConfigFile();
// Use specified JSON string (e.g. from some external file)
ConfigFile_fromString(cfg, "{""HEX_Options"":{""freezeOutTemperature"":
{""unit"":""K"",""value"":900},""type"":""exact
method""},""application"":""RPA-C"",""combustionConditions"":{""pressure"":
{""unit"":""Mpa"",""value"":20.7}},""combustionOptionalConditions"":
[],""generalOptions"":
{""ions"":false,""multiphase"":true},""info"":"""",""ingredients"":
[],""name"":"""",""version"":2}");
```

Example 2:

```
cfg = ConstructConfigFile();

// Prepare Scilab structure with parameters
cfg_struct.application = "RPA-C";
cfg_struct.version = 2;
cfg_struct.name = "test2";
cfg_struct.info = "";
cfg_struct.ingredients = [];
cfg_struct.combustionConditions.pressure.value = 20.7;
cfg_struct.combustionConditions.pressure.unit = "Mpa";
cfg_struct.combustionOptionalConditions = [];
cfg_struct.generalOptions.ions = %F;
cfg_struct.generalOptions.multiphase = %T;
cfg_struct.HEX_Options.type = "exact method";
cfg_struct.HEX_Options.freezeOutTemperature.value = 900;
cfg_struct.HEX_Options.freezeOutTemperature.unit = "K";

// Use Scilab function toJSON to convert structure to JSON string
json = toJSON(cfg_struct);

ConfigFile_fromString(cfg, json);
```

ConfigFile_toString

Write the configuration parameters to the JSON string.

```
ConfigFile_toString(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

JSON representation of the configuration

Example:

```
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "test.cfg");
json = ConfigFile_toString(cfg);
```

ConfigFile_getName

Get the case name.

```
ConfigFile_getName(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

Case name

Example:

```
cfg = ConstructConfigFile();
name = ConfigFile_getName(cfg);
```

ConfigFile_setName

Assign the case name.

```
ConfigFile_setName(cfg, name)
```

Parameters:

cfg	Reference to configuration object
name	Case name

Example:

```
cfg = ConstructConfigFile();
ConfigFile_setName(cfg, "Test case");
```

ConfigFile_getInfo

Get the case information and comments.

```
ConfigFile_getInfo(cfg)
```

Parameters:

cfg	reference to configuration object

Returned value:

Case information and comments

Example:

```
cfg = ConstructConfigFile();
info = ConfigFile_getInfo(cfg);
```

ConfigFile_setInfo

Assign the case information and comments.

```
ConfigFile_setInfo(cfg, info)
```

Parameters:

cfg	Reference to configuration object
info	Case information and comments

Example:

```
cfg = ConstructConfigFile();
ConfigFile_setInfo(cfg, "This is a test case comment");
```

ConfigFile_getGeneralOptions

Get reference to general options object.

```
ConfigFile_getGeneralOptions(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

Reference to general options object

Example:

```
cfg = ConstructConfigFile();
gopt = ConfigFile_getGeneralOptions(cfg);
```

GeneralOptions_isMultiphase

Return %T if multi-phase reaction should be considered.

```
GeneralOptions_isMultiphase(gopt)
```

Parameters:

gopt	Reference to general object

Returned value:

Boolean value (%T or %F)

Example:

```
cfg = ConstructConfigFile();
gopt = ConfigFile_getGeneralOptions(cfg);
m = GeneralOptions_isMultiphase(gopt);
```

GeneralOptions_setMultiphase

Set multi-phase flag.

```
GeneralOptions_setMultiphase(gopt, m)
```

Parameters:

gopt	Reference to general object
M	Boolean value (%T or %F)

Example:

```
cfg = ConstructConfigFile();
gopt = ConfigFile_getGeneralOptions(cfg);
GeneralOptions_setMultiphase(gopt, %T);
```

GeneralOptions_isIons

Return %T if species ionization should be considered.

```
GeneralOptions_isIons(gopt)
```

Parameters:

gopt	Reference to general object

Returned value:

Boolean value (%T or %F)

Example:

```
cfg = ConstructConfigFile();
gopt = ConfigFile_getGeneralOptions(cfg);
m = GeneralOptions_isIons(gopt);
```

GeneralOptions_setIons

Set ionization effects flag.

```
GeneralOptions_setIons(gopt, m)
```

Parameters:

gopt	Reference to general object
m	Boolean value (%T or %F)

Example:

```
cfg = ConstructConfigFile();
gopt = ConfigFile_getGeneralOptions(cfg);
GeneralOptions_setIons(gopt, %T);
```

ConfigFile_getIngredients

Get reference to ingredients list.

```
ConfigFile_getIngredients(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

Reference to ingredients list

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
```

Ingredients_*getSize*

Return number of configured ingredients.

```
Ingredients_getSize(ing)
```

Parameters:

ing	Reference to ingredients list

Returned value:

Number of configured ingredients

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
s = Ingredients_getSize(ing);
```

Ingredients_isOmitAtomsER

Return %T (true) if parameter “Omit Atoms from ER calculation” is assigned.

```
Ingredients_isOmitAtomsER(ing)
```

Parameters:

ing	Reference to ingredients list

Returned value:

Boolean value (%T or %F)

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
if Ingredients_isOmitAtomsER(ing) then
      ...
end
```

Ingredients_getOmitAtomsER

Return string with comma-separated list of atoms to omit from ER calculation, or empty string.

```
Ingredients_getOmitAtomsER(ing)
```

Parameters:

ing	Reference to ingredients list

Returned value:

String with a comma-separated list of atoms, or empty string

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
atoms = Ingredients_getOmitAtomsER(ing);
```

Ingredients_setOmitAtomsER

Assign string with comma-separated list of atoms to omit from ER calculation.

```
Ingredients_setOmitAtomsER(ing, atoms)
```

Parameters:

ing	Reference to ingredients list
atoms	String with comma-separated list of atoms, or empty string

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
Ingredients_setOmitAtomsER(ing, "Cu,Fe");
```

Ingredients_getComponent

Return reference to component specified by index.

```
Ingredients_getComponent(ing, index)
```

Parameters:

ing	Reference to ingredients list
index	Index of component on teh list Index 0 corresponds to the first component

Returned value:

Reference to component object

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
c = Ingredients_getComponent(ing);
```

Ingredients_*add*Component

Add new component to the list of ingredients.

```
Ingredients_addComponent(ing, c)
```

Parameters:

ing	Reference to ingredients list
c	Reference to component object

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
c = ConstructComponent("Mg(cr)", 0.1);
Ingredients_addComponent(ing, c);
```

Ingredients_*reset*

Removes all components from the list of ingredients.

```
Ingredients_reset(ing)
```

Parameters:

ing	Reference to ingredients list

Example:

```
cfg = ConstructConfigFile();
ing = ConfigFile_getIngredients(cfg);
Ingredients_reset(ing);
```

ConstructComponent

Create component object and return the reference.

```
ConstructComponent(name, massFraction);
```

Parameters:

name	Name of species Can be empty to create "empty" component (no assigned species)
massFraction	Mass fraction of component Can be empty Must be empty if name is empty

Returned value:

Reference to component object

Example 1:

```
c1 = ConstructComponent();
c2 = ConstructComponent("Mg(cr)", 0.1);
```

Example 2:

```
ing = ConfigFile_getIngredients(cfg);
Ingredients_addComponent(ing, ConstructComponent("SiO2(qz/crt)", 0.002985));
Ingredients_addComponent(ing, ConstructComponent("AL2O3(a)", 0.026866));
Ingredients_addComponent(ing, ConstructComponent("GuNO3", 0.506294));
Ingredients_addComponent(ing, ConstructComponent("BCN", 0.463855));
```

Component_getName

Return assigned species name.

```
Component_getName(c)
```

Parameters:

c	Reference to component object

Returned value:

Assigned name of species

Example:

```
c = ConstructComponent("Mg(cr)", 0.1);
name = Component_getName(c);
```

Component_setName

Assign species name.

```
Component_setName(c, name)
```

Parameters:

c	Reference to component object
name	Name of species

Example:

```
c = ConstructComponent();
Component_setName(c, "O2(L)");
```

Component_getMf

Return assigned mass fraction.

```
Component_getMf(c)
```

Parameters:

c	Reference to component object

Returned value:

Assigned mass fraction

Example:

```
c = ConstructComponent("Mg(cr)", 0.1);
name = Component_getMf(c);
```

Component_setMf

Assign mass fraction.

```
Component_setMf(c, mf)
```

Parameters:

c	Reference to component object
mf	Mass fraction

Example:

```
c = ConstructComponent();
Component_setName(c, "O2(L)");
Component_setMf(c, 0.2);
```

ConfigFile_getCombustionConditions

Get reference to main combustion conditions.

```
ConfigFile_getCombustionConditions(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

Reference to main combustion conditions object

Example:

```
cfg = ConstructConfigFile();
cc = ConfigFile_getCombustionConditions(cfg);
```

ConfigFile_getCombustionOptionalConditionsSize

Return number of configured optional combustion conditions.

```
ConfigFile_getCombustionOptionalConditionsSize(cfg)
```

Parameters:

cfg Reference to configuration object

Returned value:

n.a.

Example:

```
s = ConfigFile_getCombustionOptionalConditionsSize(cfg);
```

ConfigFile_clearCombustionOptionalConditionsList

Remove all configured optional combustion conditions.

```
ConfigFile_clearCombustionOptionalConditionsList(cfg)
```

Parameters:

cfg Reference to configuration object

Returned value:

n.a.

Example:

```
ConfigFile_clearCombustionOptionalConditionsList(cfg);
```

ConfigFile_setCombustionOptionalConditions

Add new optional combustion condition.

```
ConfigFile_setCombustionOptionalConditions(cfg)
```

Parameters:

cfg	Reference to configuration object

Returned value:

Reference to new optional combustion conditions object

Example:

```
copt = ConfigFile_setCombustionOptionalConditions(cfg); // Add new;
```

ConfigFile_getCombustionOptionalConditions

Get optional combustion condition specified by index.

```
ConfigFile_getCombustionOptionalConditions(cfg, index)
```

Parameters:

cfg	Reference to configuration object
Index	Index of required optional combustion condition object

Returned value:

Reference to optional combustion condition object

Example:

```
copt = ConfigFile_getCombustionOptionalConditions(cfg, 0);
```

CombustionConditions_getP

Get combustion pressure in required units.

```
CombustionConditions_getP(c, units)
```

Parameters:

c	Reference to combustion conditions object
units	Requred pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"

Returned value:

Pressure in required units

Example:

```
p = CombustionConditions_getP(c, "Mpa");
```

CombustionConditions_setP

Set combustion pressure in required units.

```
CombustionConditions_setP(c, p, units)
```

Parameters:

c	Reference to combustion conditions object
p	Pressure value
units	Requred pressure units: `"Mpa", "Pa", "bar", "atm", "at", "psi"`

Returned value:

n.a.

Example:

```
CombustionConditions_setP(copt, 98.692327, "atm");
```

CombustionConditions_isT

Check whether combustion temperature is specified.

```
CombustionConditions_isT(c)
```

Parameters:

c	Reference to combustion conditions object

Returned value:

Boolean value (`%F or %T`)

Example:

```
b = CombustionConditions_isT(c);
```

CombustionConditions_setT

Set required combustion temperature.

```
CombustionConditions_setT(c, T, units)
```

Parameters:

c	Reference to combustion conditions object
T	Temperature
units	Required temperature units: `"K", "C", "R", "F"`

Returned value:

n.a.

Example:

```
CombustionConditions_setT(c, 1400, "K");
```

CombustionConditions_getT

Return configured combustion temperature.

```
CombustionConditions_getT(c, units)
```

Parameters:

c	Reference to combustion conditions object
units	Required temperature units: `"K", "C", "R", "F"`

Returned value:

Configured combustion temperature in specified units

Example:

```
T = CombustionConditions_getT(c, "K");
```

CombustionConditions_deleteT

Remove configured combustion temperature.

```
CombustionConditions_deleteT(c)
```

Parameters:

c	Reference to combustion conditions object

Returned value:

n.a.

Example:

```
CombustionConditions_deleteT(c);
```

ConfigFile_getHexConditions

Return reference to HEX conditions object.

```
ConfigFile_getHexConditions(cfg)
```

Parameters:

n.a.

Returned value:

Reference to HEX conditions object

Example:

```
h = ConfigFile_getHexConditions(cfg);
```

HEXConditions_getType

Return type of HEX calculation method.

```
HEXConditions_getType(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

HEX calculation method as a string: `"none", "inert diluent method", "exact method"`

Example:

```
type = HEXConditions_getType(h);
```

HEXConditions_setType

Set type of HEX calculation method.

```
HEXConditions_setType(h, type);
```

Parameters:

h	Reference to HEX conditions object
type	Type of HEX calculation: `"none", "inert diluent method", "exact method"`

Returned value:

n.a.

Example:

```
HEXConditions_setType(h, "exact method");
```

HEXConditions_isFreezeOutTemperature

Return %T (true) if freeze-out temperature is specified.

```
HEXConditions_isFreezeOutTemperature(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

Boolean value (%T or %F)

Example:

```
HEXConditions_isFreezeOutTemperature(h);
```

HEXConditions_getFreezeOutTemperature

Return configured freeze-out temperature in specified units.

```
HEXConditions_getFreezeOutTemperature(h, units)
```

Parameters:

h	Reference to HEX conditions object
units	Required temperature units: "K", "C", "R", "F"

Returned value:

Freeze-out temperature in specified units

Example:

```
T = HEXConditions_getFreezeOutTemperature(h, "K");
```

HEXConditions_setFreezeOutTemperature

Set freeze-out temperature in specified units.

```
HEXConditions_setFreezeOutTemperature(h, T, units)
```

Parameters:

h	Reference to HEX conditions object
T	Temperature value
units	Required temperature units: "K", "C", "R", "F"

Returned value:

n.a.

Example:

```
HEXConditions_setFreezeOutTemperature(h, 1400, "K");;
```

HEXConditions_isAssignedLoadDensity

Return %T (true) if assigned load density is specified.

```
HEXConditions_isAssignedLoadDensity(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

Boolean value (%T of %F)

Example:

```
HEXConditions_isAssignedLoadDensity(h);
```

HEXConditions_setAssignedLoadDensity

Set assigned load density in specified units.

```
HEXConditions_setAssignedLoadDensity(h, rho, units)
```

Parameters:

h	Reference to HEX conditions object
rho	Load density value
units	Required density units: "g/cm^3", "g/cm3", "kg/m^3", "kg/m3"

Returned value:

n.a.

Example:

```
HEXConditions_setAssignedLoadDensity(h, 1, "g/cm^3");
```

HEXConditions_getAssignedLoadDensity

Return assigned load density in specified units.

```
HEXConditions_getAssignedLoadDensity(h, units)
```

Parameters:

h	Reference to HEX conditions object
units	Required density units: "g/cm^3", "g/cm3", "kg/m^3", "kg/m3"

Returned value:

Load density value in specified units

Example:

```
rho = HEXConditions_getAssignedLoadDensity(h, "g/cm^3");
```

HEXConditions_getReplaceProductsSize

Return number of replace products.

```
HEXConditions_getReplaceProductsSize(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

Number of replace products

Example:

```
HEXConditions_getReplaceProductsSize(h);
```

HEXConditions_addReplaceProduct

Add replace product.

```
HEXConditions_addReplaceProduct(h, p1, p2)
```

Parameters:

h	Reference to HEX conditions object
p1	Replaced product name
p2	Replacement product name

Returned value:

n.a.

Example:

```
HEXConditions_addReplaceProduct(h, "H2O", "H2O(L)");
```

HEXConditions_getReplaceProductKey

Return the name of replaced product identified by index.

```
HEXConditions_getReplaceProductKey(h, index)
```

Parameters:

h	Reference to HEX conditions object
index	Index of replacement paar on the list (starting with 0)

Returned value:

Name of replaced product

Example:

```
HEXConditions_getReplaceProductKey(h, 0);
```

HEXConditions_getReplaceProductValue

Return the name of replacement product identified by index.

```
HEXConditions_getReplaceProductValue(h, index)
```

Parameters:

h	Reference to HEX conditions object
index	Index of replacement paar on the list (starting with 0)

Returned value:

Name of replacement product

Example:

```
HEXConditions_getReplaceProductValue(h, 0);
```

HEXConditions_clearReplaceProducts

Clear the list of replacement products.

```
HEXConditions_clearReplaceProducts(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

n.a.

Example:

```
HEXConditions_clearReplaceProducts(h);
```

HEXConditions_getIncludeProductsSize

Return number of included products.

```
HEXConditions_getIncludeProductsSize(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

Number of included products

Example:

```
HEXConditions_getIncludeProductsSize(h);
```

HEXConditions_addIncludeProduct

Add product name to the list of included products.

```
HEXConditions_addIncludeProduct(h, p)
```

Parameters:

h	Reference to HEX conditions object

p	Product name

Returned value:

n.a.

Example:

```
HEXConditions_addIncludeProduct(h, "H2O");
```

HEXConditions_getIncludeProduct

Return the included product name specified by index.

```
HEXConditions_getIncludeProduct(h, index)
```

Parameters:

h	Reference to HEX conditions object
index	Product index, starting with 0

Returned value:

Product name

Example:

```
HEXConditions_getIncludeProduct(h, 0);
```

HEXConditions_clearIncludeProducts

Clear the list of included products.

```
HEXConditions_clearIncludeProducts(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

n.a.

Example:

```
HEXConditions_clearIncludeProducts(h);
```

HEXConditions_getOmitProductsSize

Get number of omitted products.

```
HEXConditions_getOmitProductsSize(h)
```

Parameters:

h	Reference to HEX conditions object

Returned value:

Number of omitted products

Example:

```
HEXConditions_getOmitProductsSize(h);
```

HEXConditions_addOmitProduct

Add product name to the list of omitted products.

```
HEXConditions_addOmitProduct(h, p)
```

Parameters:

h	Reference to HEX conditions object
p	Product name

Returned value:

n.a.

Example:

```
HEXConditions_addOmitProduct(h, "H2O");
```

HEXConditions_getOmitProduct

Get name of omitted product specified by index.

```
HEXConditions_getOmitProduct(h, index)
```

Parameters:

h	Reference to HEX conditions object
index	Product index, starting with 0

Returned value:

n.a.

Example:

```
HEXConditions_getOmitProduct(h, 0);
```

HEXConditions_clearOmitProducts

Clear the list of omitted products.

```
HEXConditions_clearOmitProducts(h);
```

Parameters:

h	Reference to HEX conditions object

Returned value:

n.a.

Example:

```
HEXConditions_clearOmitProducts(h);
```

Mixture API

ConstructMixture

Construct the mixture object.

```
ConstructMixture()
```

Parameters:

n.a.

Returned value:

Reference to new mixture object

Example:

```
m = ConstructMixture();
```

DeleteMixture

Delete the reference to the mixture object.

```
DeleteMixture(m)
```

Parameters:

m	Reference to mixture object

Returned value:

n.a.

Example:

```
DeleteMixture(m);
```

Mixture_size

Return number of species in the mixture.

```
Mixture_size(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Number of species in mixture

Example:

```
// Gas Yield (mol/kg)
products = Equilibrium_getResultingMixture(e);
v = 1000 / Mixture_getM(products);
v_c = 0;
for j=0:Mixture_size(products)-1
      s = Mixture_getSpecies(products, j);
            if Species_isCondensed(s) then
                  v_c = v_c + (Mixture_getFraction(products, j, "mole") * v);
            end
      end
g = v - v_c;
```

Mixture_add

Add other mixture to this mixture, assign specified mass fraction, and return reference to species object.

```
s = Mixture_add(m, mix, mf)
```

Parameters:

m	Reference to mixture object
mix	Reference to another mixture object
mf	Mass fraction of another mixture in this mixture

Returned value:

n.a.

Example:

```
Mixture_add(m, mix, 0.3);
```

Mixture_add

Add species to the mixture, assign specified mass fraction, and return reference to species object.

```
s = Mixture_add(m, name, mf)
```

Parameters:

m	Reference to mixture object
name	Species name
mf	Mass fraction of species in the mixture

Returned value:

Reference to the species object added to the mixture

Example:

```
s = Mixture_add(m, "H2O(L)", 1.0);
```

DeleteSpecies

Delete the reference to the species object.

```
DeleteSpecies(s)
```

Parameters:

s	Reference to the species object

Returned value:

n.a.

Example:

```
DeleteSpecies(s);
```

Species_getName

Return species name.

```
Species_getName(s)
```

Parameters:

s	Reference to the species object

Returned value:

Species name

Example:

```
name = Species_getName(s);
```

Species_isReactantOnly

Return %T (true) if species could not be usually used as a propellant component.

```
Species_isReactantOnly(s)
```

Parameters:

s	Reference to the species object

Returned value:

Boolean value

Example:

```
Species_isReactantOnly(s);
```

Species_isIon

Return %T (true) if species is ionized.

```
Species_isIon(s)
```

Parameters:

s	Reference to the species object

Returned value:

Boolean value

Example:

```
Species_isIon(s);
```

Species_getCharge

Return the charge of ionized species.

```
Species_getCharge(s)
```

Parameters:

s	Reference to the species object

Returned value:

Charge of species

Example:

```
c = Species_getCharge(s);
```

Species_getValence

Return the valency of species.

```
Species_getValence(s)
```

Parameters:

s	Reference to the species object

Returned value:

Valency of species

Example:

```
v = Species_getValence(s);
```

Species_isCondensed

Return %T (true) if species is condensed.

```
Species_isCondensed(s)
```

Parameters:

s	Reference to the species object

Returned value:

Boolean value

Example:

```
Species_isCondensed(s);
```

Species_getDHf298_15

Return $H^0(298.15)$ - heat of formation at the temperature 298.15 K and pressure 1 bar in desired units.

```
Species_getDHf298_15(s, units)
```

Parameters:

s	Reference to the species object
units	Result units: `"J/mol", "J/kg", "kJ/kg"`

Returned value:

Heat of formation

Example:

```
hf = Species_getDHf298_15(s, "J/mol");
```

Species_getDH298_15_0

Return $H^0(298.15) – H^0(0)$ in desired units, if available.

```
Species_getDH298_15_0(s, units)
```

Parameters:

s	Reference to the species object
units	Result units: `"J/mol", "J/kg", "kJ/kg"`

Returned value:

$H^0(298.15) – H^0(0)$

Example:

```
dh = Species_getDH298_15_0(s, "J/mol");
```

Species_getT0

Return standard temperature of the species in desired units.

```
Species_getT0(s, units)
```

Parameters:

s	Reference to the species object
units	Temperature units: `"K", "C", "R", "F"`

Returned value:

Standard temperature

Example:

```
T0 = Species_getT0(s, "K");
```

Species_getP0

Return standard pressure of the species in desired units.

```
Species_getP0(s, units)
```

Parameters:

s	Reference to the species object
units	Pressure units: `"Mpa", "Pa", "bar", "atm", "at", "psi"`

Returned value:

Standard pressure

Example:

```
p0 = Species_getP0(s, "Pa");
```

Species_getMinimumT

Return minimum temperature of the species in desired unit.

```
Species_getMinimumT(s, units)
```

Parameters:

s	Reference to the species object
units	Temperature units: `"K", "C", "R", "F"`

Returned value:

Minimum temperature

Example:

```
Tmin = Species_getMinimumT(s, "K");
```

Species_getMaximumT

Return maximum temperature of the species in desired unit.

```
Species_getMaximumT(s, units)
```

Parameters:

s	Reference to the species object
units	Temperature units: `"K", "C", "R", "F"`

Returned value:

Maximum temperature

Example:

```
Tmax = Species_getMaximumT(s, "K");
```

Species_getM

Return molecular weight of species.

```
Species_getM(s)
```

Parameters:

s	Reference to the species object

Returned value:

Molecular weight of species

Example:

```
M = Species_getM(s);
```

Species_getR

Get gas constant in desired unit (applicable for gaseous species only).

```
Species_getR(s, units)
```

Parameters:

s	Reference to the species object
units	Gas constant units: `"J/(mol K)", "J/(kg K)", "kJ/(kg K)"`

Returned value:

Gas constant in desired unit

Example:

```
R = Species_getR(s, "J/(mol K)");
```

Species_getCp

Return specific heat or molar heat capacity (depending on desired units) at specified temperature and constant pressure in desired units.

```
Species_getCp(s, T, tunits, runits)
```

Parameters:

s	Reference to the species object
T	Temperature value
tunits	Temperature units: `"K", "C", "R", "F"`
runits	Result units: `"J/(mol K)", "J/(kg K)", "kJ/(kg K)"`

Returned value:

Specific heat or molar heat capacity (depending on desired units)

Example:

```
cp = Species_getCp(s, 1300, "J/(kg K)");
```

Species_getH

Return specific or molar enthalpy (depending on desired units) at specified temperature in desired units.

```
Species_getH(s, T, tunits, runits)
```

Parameters:

s	Reference to the species object
T	Temperature value
tunits	Temperature units: `"K", "C", "R", "F"`
runits	Result units: `"J/mol", "J/kg", "kJ/kg"`

Returned value:

Specific or molar enthalpy

Example:

```
h = Species_getH(s1, 1300, "K", "J/mol");
```

Species_getS

Return specific or molar entropy (depending on desired units) at specified temperature in desired units.

```
Species_getS(s, T, tunits, runits)
```

Parameters:

s	Reference to the species object
T	Temperature value
tunits	Temperature units: "K", "C", "R", "F"
runits	Result units: "J/(mol K)", "J/(kg K)", "kJ/(kg K)"

Returned value:

Specific or molar entropy

Example:

```
s = Species_getS(s, 1300, "K", "J/(mol K)");
```

Species_getG

Get Gibbs energy of species at specified temperature.

```
Species_getG(s, T, tunits, runits)
```

Parameters:

s	Reference to the species object
T	Temperature value
tunits	Temperature units: "K", "C", "R", "F"
runits	Result units: "J/mol", "J/kg", "kJ/kg"

Returned value:

Gibbs energy of species at specified temperature

Example:

```
g = Species_getG(s, 1300, "K", "J/mol");
```

Mixture_getFraction

Return mass fraction assigned to the species identified by given index.

```
Mixture_getFraction(m, index)
```

Parameters:

m	Reference to mixture object
index	Species index

Returned value:

Mass fraction

Example:

```
mf = Mixture_getFraction(m, 0);
```

Mixture_setFraction

Assign mass fraction to the species identified by given index.

```
Mixture_setFraction(m, index, mf)
```

Parameters:

m	Reference to mixture object
index	Species index

Returned value:

n.a.

Example:

```
Mixture_setFraction(m, 0, 0.7);
```

Mixture_checkFractions

Return %T (true) if mass fractions assigned correctly (that is, the sum of all mass fractions is equals to 1.0).
If parameter

```
Mixture_checkFractions(m, fix)
```

Parameters:

m	Reference to mixture object
fix	If %T (true), the mass fractions will be re-assigned automatically to get teh sum=1.0

Returned value:

Boolean value

Example:

```
Mixture_checkFractions(m, %T);
```

Mixture_getSpecies

Return reference to species object identified by given index.

```
Mixture_getSpecies(m, index)
```

Parameters:

m	Reference to mixture object
index	Species index, starting with 0

Returned value:

n.a.

Example:

```
s = Mixture_getSpecies(m, 0);
```

Mixture_getValence

Get total mixture valency.

```
Mixture_getValence(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Mixture valency

Example:

```
v = Mixture_getValence(m);
```

Mixture_getEquivalenceRatio

Get mixture equivalence ratio.

```
Mixture_getEquivalenceRatio(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Equivalence ratio

Example:

```
eq = Mixture_getEquivalenceRatio(m);
```

Mixture_getOxygenBalance

Get mixture oxygen balance.

```
Mixture_getOxygenBalance(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Oxygen balance

Example:

```
ob = Mixture_getOxygenBalance(m);
```

Mixture_getMolesGas

Get moles of gas in mixture in desired units.

```
Mixture_getMolesGas(m, units)
```

Parameters:

m	Reference to mixture object
units	Result units: "mol/kg", "mol/g", "mol/100g"

Returned value:

Moles of gas

Example:

```
// Get the density of the initial mixture
rho = Mixture_getRho(mix, "kg/m^3");
T = Equilibrium_getT(e, "K");
products = Equilibrium_getResultingMixture(e);
g = Mixture_getMolesGas(products, "mol/kg");    // mol/kg
volGasYield = rho * g / 10000;                  // mol/100cm³
MSIFx = T * g / 1000;                           // mol·K/g
```

Mixture_getM

Get molecular weight of mixture.

```
Mixture_getM(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Molecular weight

Example:

```
M = Mixture_getM(m);
```

Mixture_getH

Get specific or molar enthalpy of mixture in desired units.

```
Mixture_getH(m, units)
```

Parameters:

m	Reference to mixture object

units	Result units: `"J/mol", "J/kg", "kJ/kg"`

Returned value:

Specific or molar enthalpy of mixture

Example:

```
h = Mixture_getH(m, "J/mol");
```

Mixture_getU

Get specific or molar internal energy of mixture in desired units.

```
Mixture_getU(m, units)
```

Parameters:

m	Reference to mixture object
units	Result units: `"J/mol", "J/kg", "kJ/kg"`

Returned value:

Specific or molar internal energy of mixture

Example:

```
u = Mixture_getU(m, "J/mol");
```

Mixture_getRho

Get density of mixture in desired units.

```
Mixture_getRho(m, units)
```

Parameters:

m	Reference to mixture object
units	Result units: `"g/cm^3", "g/cm3", "kg/m^3", "kg/m3"`

Returned value:

Density of mixture

Example:

```
rho = Mixture_getRho(m, "kg/m^3");
```

Mixture_getFormula

Get reference to exploded formula object.

```
Mixture_getFormula(m, basedOn)
```

Parameters:

m	Reference to mixture object
basedOn	Type of exploded formula: "%", "kg", "g"

Returned value:

Reference to exploded formula object

Example:

```
f = Mixture_getFormula(m, "%");
```

DeleteFormula

Delete reference to exploded formula object.

```
DeleteFormula(f);
```

Parameters:

f	Reference to exploded formula object

Returned value:

n.a.

Example:

```
DeleteFormula(f);
```

Formula_size

Number of chemical elements in exploded formula.

```
Formula_size(f)
```

Parameters:

f	Reference to exploded formula object

Returned value:

Number of chemical elements

Example:

```
n = Formula_size(f);
```

Formula_getElement

Get name of chemical element identified by index.

```
Formula_getElement(f, index)
```

Parameters:

f	Reference to exploded formula object
index	Index of chemical element in exploded formula, starting with 0

Returned value:

Name of chemical element

Example:

```
name = Formula_getElement(f, 0);
```

Formula_getNumber

Get amount of chemical element in exploded formula.

```
Formula_getNumber(f, index)
```

Parameters:

f	Reference to exploded formula object
index	Index of chemical element in exploded formula, starting with 0

Returned value:

Amount of chemical element

Example:

```
n = Formula_getNumber(f, 0);
```

Equilibrium API

ConstructEquilibrium

Create equilibrium object based on provided mixture of ingredients.

```
ConstructEquilibrium(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Reference to equilibrium object

Example:

```
e = ConstructEquilibrium(m);
```

DeleteEquilibrium

Delete equilibrium object.

```
DeleteEquilibrium(e)
```

Parameters:

e	Reference to equilibrium object

Returned value:

n.a.

Example:

```
DeleteEquilibrium(e);
```

Equilibrium_setP

Assign combustion pressure, switching the solving reaction problem to type (p,H)=const. Enthalpy of mixture of ingredients is automatically calculated using mixture assigned in constructor.

```
Equilibrium_setP(e, p, punits)
```

Parameters:

e	Reference to equilibrium object
p	Pressure value
punits	Pressure units: `"Mpa", "Pa", "bar", "atm", "at", "psi"`

Returned value:

n.a.

Example:

```
Equilibrium_setP(e, 20, "MPa");
```

Equilibrium_setPH

Assign combustion pressure and specified enthalpy, switching reaction problem to type (p,H)=const.

```
Equilibrium_setPH(e, p, punits, H, hunits)
```

Parameters:

e	Reference to equilibrium object
p	Pressure value
punits	Pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"
H	Enthalpy value
hunits	Enthalpy units: "J/mol", "kJ/mol"

Returned value:

n.a.

Example:

```
Equilibrium_setPH(e, 20, "MPa", Mixture_getH(m, "J/mol"), "J/mol");
```

Equilibrium_setPT

Assign pressure and temperature of combustion, switching reaction problem to type (p,T)=const.

```
Equilibrium_setPT(e, p, punits, T, tunits)
```

Parameters:

e	Reference to equilibrium object
p	Pressure value
punits	Pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"
T	Temperature value
tunits	Temperature units: "K", "C", "R", "F"

Returned value:

n.a.

Example:

```
Equilibrium_setPT(e, 20, "MPa", 1274.6012, "K");
```

Equilibrium_solve

Solve the configured equilibrium problem and get the reference to derivatives object.

In some cases the problem does not converge because condensed species not included before first iteration. To solve such a problems, set parameter z to true.

```
Equilibrium_solve(e, z)
```

Parameters:

e	Reference to equilibrium object
z	If %T (true) include condensed species into the reaction Optional parameter; defailt value is %T

Returned value:

Reference to derivatives object

Example:

```
d = Equilibrium_solve(e);

d = Equilibrium_solve(e, %F);
```

Derivatives_getCp

Get specific heat or molar heat capacity of reaction products at constant pressure in desired units.

```
Derivatives_getCp(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: "J/(kg K)", "kJ/(kg K)", J/(mol K)

Returned value:

Specific heat or molar heat capacity Cp

Example:

```
Cp = Derivatives_getCp(d, "J/(mol K)");
```

Derivatives_getCv

Get specific heat or molar heat capacity of reaction products at constant volume in desired units.

```
Derivatives_getCv(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: "J/(kg K)", "kJ/(kg K)", J/(mol K)

Returned value:

Specific heat or molar heat capacity Cv

Example:

```
Cv = Derivatives_getCv(d, "J/(mol K)");
```

Derivatives_getR

Get gas constant of reaction products in desired units.

```
Derivatives_getR(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: "J/(kg K)", "kJ/(kg K)", J/(mol K)

Returned value:

Gas constant R

Example:

```
R = Derivatives_getR(d, "J/(mol K)");
```

Derivatives_getK

Get isentropic exponent of reaction products.

```
Derivatives_getK(d)
```

Parameters:

d	Reference to derivatives object

Returned value:

Isentropic exponent

Example:

```
k = Derivatives_getK(d);
```

Derivatives_getGamma

Get specific heat ratio of reaction products.

```
Derivatives_getGamma(d)
```

Parameters:

d	Reference to derivatives object

Returned value:

Specific heat ratio of reaction products

Example:

```
gamma = Derivatives_getGamma(d);
```

Derivatives_getA

Get velocity of sound in desired units.

```
Derivatives_getA(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: `"m/s"`

Returned value:

Velocity of sound

Example:

```
a = Derivatives_getA(d, "m/s");
```

Derivatives_getRho

Get density of reaction products in desired units.

```
Derivatives_getRho(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: `"kg/m^3", "g/m^3"`

Returned value:

Density of reaction products

Example:

```
rho = Derivatives_getRho(d, "kg/m^3");
```

Derivatives_getRhoGas

Get density of gaseous reaction products in desired units.

```
Derivatives_getRhoGas(d, units)
```

Parameters:

d	Reference to derivatives object
units	Result units: "kg/m^3", "g/m^3"

Returned value:

Density of gaseous reaction products

Example:

```
rho_gas = Derivatives_getRhoGas(d, "kg/m^3");
```

Derivatives_getZ

Get mass fraction of condensed reaction products.

```
Derivatives_getZ(d)
```

Parameters:

d	Reference to derivatives object

Returned value:

Mass fraction of condensed reaction products

Example:

```
z = Derivatives_getZ(d);
```

Derivatives_getM

Get molecular weight of reaction products.

```
Derivatives_getM(d)
```

Parameters:

d	Reference to derivatives object

Returned value:

Molecular weight of reaction products

Example:

```
M = Derivatives_getM(d);
```

Equilibrium_getP

Get assigned pressure of reaction.

```
Equilibrium_getP(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"

Returned value:

Pressure of reaction

Example:

```
p = Equilibrium_getP(e, "MPa");
```

Equilibrium_getT

Get the temperature of reaction.

```
Equilibrium_getT(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Temperature units: "K", "C", "R", "F"

Returned value:

Temperature of reaction

Example:

```
T = Equilibrium_getT(e, "K");
```

Equilibrium_getH

Get specific or molar enthalpy of reaction products.

```
Equilibrium_getH(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Enthalpy units: "J/mol", "kJ/mol", "J/kg", "kJ/kg"

Returned value:

Enthalpy of reaction products

Example:

```
H = Equilibrium_getH(e, "J/mol");
```

Equilibrium_getU

Get specific or molar internal energy of reaction products.

```
Equilibrium_getU(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Internal energy units: `"J/mol", "kJ/mol", "J/kg", "kJ/kg"`

Returned value:

Internal energy of reaction products

Example:

```
U = Equilibrium_getU(e, "J/mol");
```

Equilibrium_getS

Get specific or molar entropy of reaction products.

```
Equilibrium_getS(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Entropy units: `"J/(mol K)", "J/(kg K)", "kJ/(kg K)"`

Returned value:

Entropy of reaction products

Example:

```
S = Equilibrium_getS(e, "J/(mol K)");
```

Equilibrium_getG

Get Gibbs energy of reaction products.

```
Equilibrium_getG(e, units)
```

Parameters:

e	Reference to equlibrium object
units	Gibbs energy units: `"J/mol", "kJ/mol", "J/kg", "kJ/kg"`

Returned value:

Gibbs energy of reaction products

Example:

```
G = Equilibrium_getG(e, "J/mol");
```

Equilibrium_hasCondensedPhase

Get %T (true) if reaction products contains condensed species.

```
Equilibrium_hasCondensedPhase(e)
```

Parameters:

e	Reference to equlibrium object

Returned value:

Boolean value (%T or %F)

Example:

```
Equilibrium_hasCondensedPhase(e);
```

Equilibrium_getResultingMixture

Get reference to Mixture object, containing all products of reaction.

```
Equilibrium_getResultingMixture(e)
```

Parameters:

e	Reference to equlibrium object

Returned value:

Reference to Mixture object

Example:

```
rm = Equilibrium_getResultingMixture(e);
```

Combustor API

ConstructCombustor

Create Combustor object using given object Mixture with ingrediants.

```
ConstructCombustor(m)
```

Parameters:

m	Reference to mixture object

Returned value:

Reference to Combustor object

Example:

```
c = ConstructCombustor(m);
```

DeleteCombustor

Delete combustor object.

```
DeleteCombustor(c);
```

Parameters:

c	Reference to combustor object

Returned value:

n.a.

Example:

```
DeleteCombustor(c);
```

Combustor_setP

Assign combustion chamber pressure, switching the solving reaction problem to type (p,H)=const.. Enthalpy of mixture of ingredients is automatically calculated using mixture assigned in constructor.

```
Combustor_setP(c, p, punits)
```

Parameters:

c	Reference to combustor object
p	Pressure value
punits	Pressure units: `"Mpa", "Pa", "bar", "atm", "at", "psi"`

Returned value:

n.a.

Example:

```
Combustor_setP(c, 20, "Mpa");
```

Combustor_setPH

Assign combustion pressure and specified enthalpy, switching reaction problem to type (p,H)=const.

```
Combustor_setPH(c, p, punits, H, hunits)
```

Parameters:

c	Reference to combustor object
p	Pressure value
punits	Pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"
H	Enthalpy value
hunits	Enthalpy units: "J/mol", "kJ/mol"

Returned value:

n.a.

Example:

```
Combustor_setPH(c, 20, "MPa", Mixture_getH(m, "J/mol"), "J/mol");
```

Combustor_setPT

Assign pressure and temperature of combustion, switching reaction problem to type (p,T)=const.

```
Combustor_setPT(c, p, punits, T, tunits)
```

Parameters:

c	Reference to combustor object
p	Pressure value
punits	Pressure units: "Mpa", "Pa", "bar", "atm", "at", "psi"
T	Temperature value
tunits	Temperature units: "K", "C", "R", "F"

Returned value:

n.a.

Example:

```
Combustor_setPT(c, 20, "MPa", 1274.6012, "K");
```

Combustor_solve

Solve the chemical equilibrium problem.

In some cases the problem does not converge because condensed species not included before first iteration. To solve such a problems, set parameter z to true.

```
Combustor_solve(c, z)
```

Parameters:

c	Reference to combustor object
z	If %T (true) include condensed species into the reaction Optional parameter; defailt value is %T

Returned value:

n.a.

Example:

```
Combustor_solve(c);

Combustor_solve(c, %T)
```

Combustor_getEquilibrium

Get reference to object Equilibrium for solved chemical equilibrium problem.

```
Combustor_getEquilibrium(c)
```

Parameters:

c	Reference to combustor object

Returned value:

Reference to equilibrium object

Example:

```
e = Combustor_getEquilibrium(c);
```

Combustor_getDerivatives

Get regerence to object Derivatives for solved chemical equilibrium problem.

```
Combustor_getDerivatives(c)
```

Parameters:

c	Reference to combustor object

Returned value:

Reference to derivatives object

Example:

```
d = Combustor_getDerivatives(c);
```

Combustion Analysis API

ConstructCombustionAnalysis

Create combustion analysis object.

```
ConstructCombustionAnalysis()
```

Parameters:

n.a.

Returned value:

Reference to combustion analysis object

Example:

```
ca = ConstructCombustionAnalysis();
```

DeleteCombustionAnalysis

Delete combustion analysis object.

```
DeleteCombustionAnalysis(ca)
```

Parameters:

ca	Reference to combustion analysis object.

Returned value:

n.a.

Example:

```
DeleteCombustionAnalysis(ca);
```

CombustionAnalysis_setPrintResults

Configure the verbose logging mode.

```
CombustionAnalysis_setPrintResults(ca, print);
```

Parameters:

ca	Reference to combustion analysis object
print	If %T (true), set the verbose logging mode while running the analysis

Returned value:

n.a.

Example:

```
ca = ConstructCombustionAnalysis();
CombustionAnalysis_setPrintResults(ca, %F);
CombustionAnalysis_run(ca, cfg);
```

CombustionAnalysis_run

Execute the problem configured in given configuration file.

```
CombustionAnalysis_run(ca, cfg);
```

Parameters:

ca	Reference to combustion analysis object.
cfg	Reference to configuration object.

Returned value:

n.a.

Example:

```
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "test.cfg");
ca = ConstructCombustionAnalysis();
CombustionAnalysis_run(ca, cfg);
```

CombustionAnalysis_isHEX

Return %T (true) if HEX was configured and calculated.

```
CombustionAnalysis_isHEX(ca)
```

Parameters:

ca	Reference to combustion analysis object.

Returned value:

Boolean value (%T or %F)

Example:

```
CombustionAnalysis_isHEX(ca);
```

CombustionAnalysis_getHEX

Get calculated HEX in specified unts.

```
CombustionAnalysis_getHEX(ca, units)
```

Parameters:

ca	Reference to combustion analysis object.
units	Result units: `"J/kg", "kJ/kg", "kcal/kg", "Btu/lbm"`

Returned value:

n.a.

Example:

```
hex = CombustionAnalysis_getHEX(ca, "J/kg");
```

CombustionAnalysis_isHEXInterpolated

Return %T (true) if HEX was interpolated.

```
CombustionAnalysis_isHEXInterpolated(ca)
```

Parameters:

ca	Reference to combustion analysis object.

Returned value:

Boolean %T if HEX is interpolated

Example:

```
CombustionAnalysis_isHEXInterpolated(ca);
```

CombustionAnalysis_getCombustorsListSize

Return number of combustors, including the combustor for the main combustion conditions as well as all combustors for optional combustion condistions.

```
CombustionAnalysis_getCombustorsListSize(ca)
```

Parameters:

ca	Reference to combustion analysis object.

Returned value:

Number of available combustors

Example:

```
size = CombustionAnalysis_getCombustorsListSize(ca);
```

CombustionAnalysis_getEquilibrium

Return reference to equilibrium object for specified index.

```
CombustionAnalysis_getEquilibrium(ca, index);
```

Parameters:

ca	Reference to combustion analysis object.
index	Combustion condition index. Index 0 corresponds to main combustion conditions. Index > 0 corresponds to optional combustion conditions.

Returned value:

Reference to equilibrium object

Example:

```
e = CombustionAnalysis_getEquilibrium(ca, 0);

if CombustionAnalysis_getCombustorsListSize(ca)>0 then
      e_opt = CombustionAnalysis_getEquilibrium(ca, 1);
end
```

CombustionAnalysis_getDerivatives

Return reference to derivatives object for specified index.

```
CombustionAnalysis_getDerivatives(ca, index);
```

Parameters:

ca	Reference to combustion analysis object.
index	Combustion condition index. Index 0 corresponds to main combustion conditions. Index > 0 corresponds to optional combustion conditions.

Returned value:

Reference to derivatives object

Example:

```
d = CombustionAnalysis_getDerivatives(ca, 0);

if CombustionAnalysis_getCombustorsListSize(ca)>0 then
      d_opt = CombustionAnalysis_getDerivatives(ca, 1);
end
```

Scilab examples

All examples can be found in the directory “scilab-examples” within installation directory of the RPA-C.

example1.sce

exec loader.sce

```
RPAInit()

// clear console
clc
// clear memory
clear
// close all plots
xdel(winsid())

tic()

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

// Load configuration from the file
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "scilab-examples/ScilabTest.cfg")

// Create separate copy of mixture with ingredients to get some
// propertiesof the mixture: density and equivalence ratio
// We coud get it from CombustionAnalysis, but only after the analysis run,
// which is in the loop in this example. So, to improve the performance,
// we create another copy of mixtue outside of the loop
mix = ConstructMixture();
ingredients = ConfigFile_getIngredients(cfg);
for i=0:Ingredients_getSize(ingredients)-1
    ing = Ingredients_getComponent(ingredients, i);
    s = Mixture_add(mix, Component_getName(ing), Component_getMf(ing));
end

// Get the density of the mixture in kg/m3
rho = Mixture_getRho(mix, "kg/m^3");

// Equivalence ratio
er = Mixture_getEquivalenceRatio(mix,
Ingredients_getOmitAtomsER(ingredients));

//*************************************
// Run amalysis

p=logspace(-1,2,30)
```

```
for i=1:size(p,'*')

    cc = ConfigFile_getCombustionConditions(cfg);
    CombustionConditions_setP(cc, p(i), "MPa"); // Update main combustion
conditions

    ca = ConstructCombustionAnalysis();
    CombustionAnalysis_setPrintResults(ca, %F);
    CombustionAnalysis_run(ca, cfg);

    if CombustionAnalysis_getCombustorsListSize(ca)>0 then

        e = CombustionAnalysis_getEquilibrium(ca, 0);

        // Pressure
        _p = Equilibrium_getP(e, "MPa");

        // Flame Temperature in K
        T = Equilibrium_getT(e, "K");

        // Gas Yield (mol/kg)
        products = Equilibrium_getResultingMixture(e);
        v = 1000 / Mixture_getM(products);      // Mole number in 100 gm of
whole mixture
        v_c = 0;                                // Mole number in 100 gm of
condenced mixture
        for j=0:Mixture_size(products)-1
            s = Mixture_getSpecies(products, j);
            if Species_isCondensed(s) then
                v_c = v_c + (Mixture_getFraction(products, j, "mole") * v);
            end
        end
        g = v - v_c;

        // Condensed fraction
        vc = 0;
        d = CombustionAnalysis_getDerivatives(ca, 0);
        vc = Derivatives_getZ(d);

        mprintf("%2d: p=%f T=%f rho=%f er=%f g=%f vc=%f\n", i, _p, T, rho,
er, g, vc);

        data(:,i) = [T, rho, er, g, vc];

    end
    DeleteCombustionAnalysis(ca);
end

d=toc()
disp(d)
```

```
//************************************
// Plot diagrams

xdel()
f=figure("background",-2,"figure_position", [0 0],"figure_size",[1800 1000]);
drawlater()

subplot(1,3,1)
plot(p,data(1,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Flame Temperature',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Flame Temperature (K)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

subplot(1,3,2)
plot(p,data(5,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Gas Yield',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Gas yield (mol/kg)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

subplot(1,3,3)
plot(p,data(5,:).*data(1,:)/1000,'bo-','thickness',2)
xgrid(33,1,8)
title('Massic Sifx',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Massic Sifx (mol.K/g)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

drawnow()
```

example2.sce

exec loader.sce

```
RPAInit()

// clear console
clc
// clear memory
clear
// close all plots
xdel(winsid())

tic()

//************************************
// Create configuration in memory
```

```
cfg = ConstructConfigFile();
ConfigFile_setName(cfg, "Test")

ingredients = ConfigFile_getIngredients(cfg);
Ingredients_addComponent(ingredients, ConstructComponent("BCN", 0.537));
Ingredients_addComponent(ingredients, ConstructComponent("GuNO3", 0.248));
Ingredients_addComponent(ingredients, ConstructComponent("CuDABT", 0.200));
Ingredients_addComponent(ingredients, ConstructComponent("AL2O3(a)",
0.015));
Ingredients_setOmitAtomsER(ingredients, "CU");

gopt = ConfigFile_getGeneralOptions(cfg);
GeneralOptions_setMultiphase(gopt, %T);
GeneralOptions_setIons(gopt, %T);

cc = ConfigFile_getCombustionConditions(cfg);
CombustionConditions_setP(cc, 20.7, "MPa");               // Will be changed
in the loop below

hexc = ConfigFile_getHexConditions(cfg);
HEXConditions_setType(hexc, "none");

// Configuration is cerated
//************************************

// Create separate copy of mixture with ingredients to get some
// propertiesof the mixture: density and equivalence ratio
// We coud get it from CombustionAnalysis, but only after the analysis run,
// which is in the loop in this example. So, to improve the performance,
// we create another copy of mixtue outside of the loop
mix = ConstructMixture();
ingredients = ConfigFile_getIngredients(cfg);
for i=0:Ingredients_getSize(ingredients)-1
    ing = Ingredients_getComponent(ingredients, i);
    s = Mixture_add(mix, Component_getName(ing), Component_getMf(ing));
end

// Get the density of the mixture in kg/m3
rho = Mixture_getRho(mix, "kg/m^3");

// Equivalence ratio
er = Mixture_getEquivalenceRatio(mix,
Ingredients_getOmitAtomsER(ingredients));

//************************************
// Run amalysis

p=logspace(-1,2,30)

for i=1:size(p,'*')
```

```
    cc = ConfigFile_getCombustionConditions(cfg);
    CombustionConditions_setP(cc, p(i), "MPa"); // Update main combustion
conditions

    ca = ConstructCombustionAnalysis();
    CombustionAnalysis_setPrintResults(ca, %F);
    CombustionAnalysis_run(ca, cfg);

    if CombustionAnalysis_getCombustorsListSize(ca)>0 then

        e = CombustionAnalysis_getEquilibrium(ca, 0);

        // Pressure
        _p = Equilibrium_getP(e, "MPa");

        // Flame Temperature in K
        T = Equilibrium_getT(e, "K");

        // Gas Yield (mol/kg)
        products = Equilibrium_getResultingMixture(e);
        v = 1000 / Mixture_getM(products);      // Mole number in 100 gm of
whole mixture
        v_c = 0;                                // Mole number in 100 gm of
condenced mixture
        for j=0:Mixture_size(products)-1
            s = Mixture_getSpecies(products, j);
            if Species_isCondensed(s) then
                v_c = v_c + (Mixture_getFraction(products, j, "mole") * v);
            end
        end
        g = v - v_c;

        // Condensed fraction
        vc = 0;
        d = CombustionAnalysis_getDerivatives(ca, 0);
        vc = Derivatives_getZ(d);

        mprintf("%2d: p=%f T=%f rho=%f er=%f g=%f vc=%f\n", i, _p, T, rho,
er, g, vc);

        data(:,i) = [T, rho, er, g, vc];

    end
    DeleteCombustionAnalysis(ca);
end

d=toc()
disp(d)

//************************************
// Plot diagrams
```

```
xdel()
f=figure("background",-2,"figure_position", [0 0],"figure_size",[1800 1000]);
drawlater()

subplot(1,3,1)
plot(p,data(1,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Flame Temperature',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Flame Temperature (K)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

subplot(1,3,2)
plot(p,data(5,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Gas Yield',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Gas yield (mol/kg)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

subplot(1,3,3)
plot(p,data(5,:).*data(1,:)/1000,'bo-','thickness',2)
xgrid(33,1,8)
title('Massic Sifx',"font_style",8,"fontsize",3)
xlabel("Pressure (MPa)","font_style",8,"fontsize",2)
ylabel("Massic Sifx (mol.K/g)","font_style",8,"fontsize",2)
a=gca();a.log_flags="lnn"

drawnow()
```

example3.sce

exec loader.sce

```
RPAInit()

// clear console
clc
// clear memory
clear
// close all plots
xdel(winsid())

tic()

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

// Load configuration from the file
cfg = ConstructConfigFile();
ConfigFile_read(cfg, "scilab-examples/ScilabTest.cfg")
```

```
// Find imgredients "GuNO3" and "BCN"
ingredients = ConfigFile_getIngredients(cfg);
for i=0:Ingredients_getSize(ingredients)-1
    c = Ingredients_getComponent(ingredients, i)
    if Component_getName(c)=="GuNO3" then
        c_gun = c
    end
    if Component_getName(c)=="BCN" then
        c_bcn = c
    end
end

if ~Pointer_isComponent(c_gun) then
    disp("Could not find GuNO3!");
    exit;
end
if ~Pointer_isComponent(c_bcn) then
    disp("Could not find BCN!");
    exit;
end

c_total_mf = Component_getMf(c_gun) + Component_getMf(c_bcn);

//*************************************
// Run amalysis

y1=.4
y2=.6
x1=1
x2=30

for i=x1:x2
    txg(i)=((y1-y2)/(x1-x2))*i+(-(x2*y1-x1*y2)/(x1-x2))
    txb(i)=c_total_mf-txg(i)

    // Re-assign the mass fractions
    Component_setMf(c_gun, txg(i));
    Component_setMf(c_bcn, txb(i));

    ca = ConstructCombustionAnalysis();
    CombustionAnalysis_setPrintResults(ca, %F);
    CombustionAnalysis_run(ca, cfg);

    if CombustionAnalysis_getCombustorsListSize(ca)>0 then

        // Get current initial mixture (ingredients)
        mix = CombustionAnalysis_getMixture(ca);

        // Get the density of the mixture in kg/m3
        rho = Mixture_getRho(mix, "kg/m^3");

        // Equivalence ratio
```

```
        er = Mixture_getEquivalenceRatio(mix,
Ingredients_getOmitAtomsER(ingredients));

        e = CombustionAnalysis_getEquilibrium(ca, 0);

        // Pressure
        p = Equilibrium_getP(e, "MPa");

        // Flame Temperature in K
        T = Equilibrium_getT(e, "K");

        // Gas Yield (mol/kg)
        products = Equilibrium_getResultingMixture(e);
        v = 1000 / Mixture_getM(products);      // Mole number in 100 gm of
whole mixture
        v_c = 0;                                // Mole number in 100 gm of
condenced mixture
        for j=0:Mixture_size(products)-1
            s = Mixture_getSpecies(products, j);
            if Species_isCondensed(s) then
                v_c = v_c + (Mixture_getFraction(products, j, "mole") * v);
            end
        end
        g = v - v_c;

        // Condensed fraction
        vc = 0;
        d = CombustionAnalysis_getDerivatives(ca, 0);
        vc = Derivatives_getZ(d);

        mprintf("%2d: Mf=%f+%f=%f p=%f T=%f rho=%f er=%f g=%f vc=%f\n", i,
txg(i), txb(i), c_total_mf, p, T, rho, er, g, vc);

        data(:,i) = [T, rho, er, g, vc];

    end
    DeleteCombustionAnalysis(ca);
end

d=toc()
disp(d)

//************************************
// Plot diagrams

xdel()
f=figure("background",-2,"figure_position", [0 0],"figure_size",[1800 1000]);
drawlater()

subplot(2,3,1)
plot(txg',data(1,:),'bo-','thickness',2)
```

```
xgrid(33,1,8)
title('Flame Temperature',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Flame Temperature (K)","font_style",8,"fontsize",2)

subplot(2,3,2)
plot(txg',data(4,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Gas Yield',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Gas yield (mol/kg)","font_style",8,"fontsize",2)

subplot(2,3,3)
plot(txg',data(4,:).*data(1,:)/1000,'bo-','thickness',2)
xgrid(33,1,8)
title('Massic Sifx',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Massic Sifx (mol.K/g)","font_style",8,"fontsize",2)

subplot(2,3,4)
plot(txg',data(2,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Density',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Density (g/m3)","font_style",8,"fontsize",2)

subplot(2,3,5)
plot(txg',data(3,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Equivalent Ratio',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Equivalent ratio","font_style",8,"fontsize",2)

subplot(2,3,6)
plot(txg',data(5,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Condensed phase',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Condensed Phase (%)","font_style",8,"fontsize",2)

drawnow()
```

example4.sce

exec loader.sce

```
RPAInit()

// clear console
clc
```

```
// clear memory
clear
// close all plots
xdel(winsid())

tic()

//*************************************
// Create configuration in memory

cfg = ConstructConfigFile();
ConfigFile_setName(cfg, "Test")

// We will change mass fractions for these two ingredients in the loop
// so assign them to separate variables to access later easier
c_gun = ConstructComponent("GuNO3", 0.248);
c_bcn = ConstructComponent("BCN", 0.537);
c_total_mf = Component_getMf(c_gun) + Component_getMf(c_bcn);

ingredients = ConfigFile_getIngredients(cfg);
Ingredients_addComponent(ingredients, ConstructComponent("CuDABT", 0.200));
Ingredients_addComponent(ingredients, ConstructComponent("AL2O3(a)",
0.015));
Ingredients_addComponent(ingredients, c_gun);
Ingredients_addComponent(ingredients, c_bcn);
Ingredients_setOmitAtomsER(ingredients, "CU");

gopt = ConfigFile_getGeneralOptions(cfg);
GeneralOptions_setMultiphase(gopt, %T);
GeneralOptions_setIons(gopt, %T);

cc = ConfigFile_getCombustionConditions(cfg);
CombustionConditions_setP(cc, 20.7, "MPa");               // Will be changed
in the loop below

hexc = ConfigFile_getHexConditions(cfg);
HEXConditions_setType(hexc, "none");

//*************************************
// Run amalysis

y1=.4
y2=.6
x1=1
x2=30

for i=x1:x2
    txg(i)=((y1-y2)/(x1-x2))*i+(-(x2*y1-x1*y2)/(x1-x2))
    txb(i)=c_total_mf-txg(i)

    // Re-assign the mass fractions
```

```
    Component_setMf(c_gun, txg(i));
    Component_setMf(c_bcn, txb(i));

    ca = ConstructCombustionAnalysis();
    CombustionAnalysis_setPrintResults(ca, %F);
    CombustionAnalysis_run(ca, cfg);

    if CombustionAnalysis_getCombustorsListSize(ca)>0 then

        // Get current initial mixture (ingredients)
        mix = CombustionAnalysis_getMixture(ca);

        // Get the density of the mixture in kg/m3
        rho = Mixture_getRho(mix, "kg/m^3");

        // Equivalence ratio
        er = Mixture_getEquivalenceRatio(mix,
Ingredients_getOmitAtomsER(ingredients));

        e = CombustionAnalysis_getEquilibrium(ca, 0);

        // Pressure
        p = Equilibrium_getP(e, "MPa");

        // Flame Temperature in K
        T = Equilibrium_getT(e, "K");

        // Gas Yield (mol/kg)
        products = Equilibrium_getResultingMixture(e);
        v = 1000 / Mixture_getM(products);      // Mole number in 100 gm of
whole mixture
        v_c = 0;                                // Mole number in 100 gm of
condenced mixture
        for j=0:Mixture_size(products)-1
            s = Mixture_getSpecies(products, j);
            if Species_isCondensed(s) then
                v_c = v_c + (Mixture_getFraction(products, j, "mole") * v);
            end
        end
        g = v - v_c;

        // Condensed fraction
        vc = 0;
        d = CombustionAnalysis_getDerivatives(ca, 0);
        vc = Derivatives_getZ(d);

        mprintf("%2d: Mf=%f+%f=%f p=%f T=%f rho=%f er=%f g=%f vc=%f\n", i,
txg(i), txb(i), c_total_mf, p, T, rho, er, g, vc);

        data(:,i) = [T, rho, er, g, vc];
```

```
    end
    DeleteCombustionAnalysis(ca);
end

d=toc()
disp(d)

//************************************
// Plot diagrams

xdel()
f=figure("background",-2,"figure_position", [0 0],"figure_size",[1800 1000]);
drawlater()

subplot(2,3,1)
plot(txg',data(1,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Flame Temperature',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Flame Temperature (K)","font_style",8,"fontsize",2)

subplot(2,3,2)
plot(txg',data(4,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Gas Yield',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Gas yield (mol/kg)","font_style",8,"fontsize",2)

subplot(2,3,3)
plot(txg',data(4,:).*data(1,:)/1000,'bo-','thickness',2)
xgrid(33,1,8)
title('Massic Sifx',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Massic Sifx (mol.K/g)","font_style",8,"fontsize",2)

subplot(2,3,4)
plot(txg',data(2,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Density',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Density (g/m3)","font_style",8,"fontsize",2)

subplot(2,3,5)
plot(txg',data(3,:),'bo-','thickness',2)
xgrid(33,1,8)
title('Equivalent Ratio',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Equivalent ratio","font_style",8,"fontsize",2)

subplot(2,3,6)
plot(txg',data(5,:),'bo-','thickness',2)
xgrid(33,1,8)
```

```
title('Condensed phase',"font_style",8,"fontsize",3)
xlabel("Guni rate","font_style",8,"fontsize",2)
ylabel("Condensed Phase (%)","font_style",8,"fontsize",2)

drawnow()
```

example5.sce

exec loader.sce

```
RPAInit()

// clear console
clc
// clear memory
clear
// close all plots
xdel(winsid())

tic()

//*************************************************************
// Set initial data

// Initial ingredients
ingredients_list = list();

ingredients_list(1).name="GuNO3";
ingredients_list(1).massFraction = 0.248;

ingredients_list(2).name = "BCN";
ingredients_list(2).massFraction = 0.537;

ingredients_list(4).name="CuDABT";
ingredients_list(4).massFraction = 0.19;

ingredients_list(3).name="KClO4(a)";
ingredients_list(3).massFraction = 0.01;

ingredients_list(5).name="AL2O3(a)";
ingredients_list(5).massFraction = 0.015;

omitAtomsER = "CU";

// cost of species in same order as in introduction
cost=[
    3.5715    // GuNO3
    7.50      // BCN
    3.00      // CuDABT
```

```
    13.71     // KClO4
    3.00      // AL2O3(a)
];

// Set ranges for each element of the formulation
levels=[
    40.     49.    // GuNO3
    32.     40.4   // BCN
    7.0      9.0   // CuDABT
    7.5      9.5   // KClO4
    0.5      2.0   // AL2O3(a)
]/100;

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

// Number of species in the list of
species = length(ingredients_list);

if (species<>size(cost,'r')) then
    mprintf("Invalid initial ingredients configuration!");
    return;
end

if (species<>size(levels,'r')) then
    mprintf("Invalid initial ingredients configuration!");
    return;
end

// Indices in matrices "D" and "data"
er_index=species+      1;
rho_index=species+     2;
oxBal_index=species+   3;
cost_index=species+    4;
T_index=species+       5;
g_index=species+       6;
vc_index=species+      7;
hex_index=species+     8;
// Add new here like:
// new_param_index=species+ 9;

// Update if indices for new parameters added, e.g.
max_index=new_param_index
max_index = hex_index

// Set number of steps for each element
n=10

// Create matrix of levels for each element
for i=1:species
    A(i,:)=grand(1,n,"unf",levels(i,1),levels(i,2))
end
B=zeros(species,n);
```

```
// Create all formulations with all permutations
for i=1:n
    S=sum(A(1:species,i))
    B(:,i)=[A(1,i)/S A(2,i)/S A(3,i)/S A(4,i)/S A(5,i)/S A(6,i)/S];
end
clear i1 i2 i3 i4 i5 i6 n

//Remove non useful configurations
j=0
for i=1:max(size(B))
    if (B(1,i)>levels(1,1) && B(1,i)<levels(1,2)) then
        if (B(2,i)>levels(2,1) && B(2,i)<levels(2,2)) then
            if (B(3,i)>levels(3,1) && B(3,i)<levels(3,2)) then
                if (B(4,i)>levels(4,1) && B(4,i)<levels(4,2)) then
                    if (B(5,i)>levels(5,1) && B(5,i)<levels(5,2)) then
                        j=j+1
                        C(:,j)=B(:,i)
                    end
                end
            end
        end
    end
end
clear i j R S levels

// Define config file
cfg = ConstructConfigFile();
ingredients = ConfigFile_getIngredients(cfg);
// List "components_list" will be used to modify mass fractions in the
analysis loop
components_list = list();
for i = 1:species
    // Store the reference to configured component in the list for future
use
    components_list(i) = ConstructComponent(ingredients_list(i).name,
ingredients_list(i).massFraction);
    Ingredients_addComponent(ingredients, components_list(i));
end
Ingredients_setOmitAtomsER(ingredients, omitAtomsER);

gopt = ConfigFile_getGeneralOptions(cfg);
GeneralOptions_setMultiphase(gopt, %T);
GeneralOptions_setIons(gopt, %T);

cc = ConfigFile_getCombustionConditions(cfg);
CombustionConditions_setP(cc, 20.7, "MPa");

hexc = ConfigFile_getHexConditions(cfg);
HEXConditions_setType(hexc, "exact method");
HEXConditions_setFreezeOutTemperature(hexc, 900, "K");

// Calculate properties of the initial mixture
// using object "Mixture"
```

```
mix = ConstructMixture();
for i = 1:species
    s = Mixture_add(mix, ingredients_list(i).name,
ingredients_list(i).massFraction);
end
j=0
for i=1:size(C,2)
    for s=1:species
        Mixture_setFraction(mix, s-1, C(s,i))
    end
    Mixture_checkFractions(mix, %T)

    er = Mixture_getEquivalenceRatio(mix, omitAtomsER);
    rho = Mixture_getRho(mix,"kg/m^3");
    oxBal = Mixture_getOxygenBalance(mix);

    if(er > 0.8)&&(er < 1.2) then
        j=j+1
        D(1:species,j)=C(1:species,i)
        D(er_index,j)=er
        D(rho_index,j)=rho
        D(oxBal_index,j)=oxBal;
    end
end
DeleteMixture(mix)
clear mix
if(j==0) then
    mprintf("Matrix D was not defined!")
    return;
end

// Calculate cost of each formulation
for i=1:max(size(D,'c'))
    D(cost_index,i)=sum(D(1:species,i).*cost(1:species))
end
clear cost

//***********************************************************
// Run analysis

data=zeros(max_index,size(D,2));
mprintf("Number of Runs = %d",size(D,2));

winH=waitbar('In progress');
for i=1:size(D,2)
    waitbar(i/size(D,2),winH);
    tic()

    for s=1:species
        // Here we are using the list "components_list"
        // prepared when config filw was defined
        Component_setMf(components_list(s), D(s,i));
    end
```

```
    // Perform combustion analysis
    ca = ConstructCombustionAnalysis();
    CombustionAnalysis_run(ca, cfg);

    // Get equilibrium and derivatives
    e=CombustionAnalysis_getEquilibrium(ca, 0);
    d = CombustionAnalysis_getDerivatives(ca, 0);

    // Get flame temp
    T = Equilibrium_getT(e, "K");
    data(T_index,i)=T

    // Get density
    data(rho_index,i)=D(rho_index,i)

    // Get equivalence ratio
    data(er_index,i)=D(er_index,i)

    // Get gas yield
    products = Equilibrium_getResultingMixture(e);
    g = Mixture_getMolesGas(products, "mol/kg");    // mol/kg
    gv = rho * g / 10000;                           // mol/100cm³
    msifx = T * g / 1000;                           // mol·K/g
    data(g_index,i)=g

    // Condensed fraction
    vc = Derivatives_getZ(d);
    data(vc_index,i) = vc;

    // Heat of Explosion
    hex = CombustionAnalysis_getHEX(ca, "kJ/kg");
    data(hex_index,i) = hex;
    data(1:species,i) = D(1:species,i)
    data(cost_index,i) = D(cost_index,i)

    // Get oxygen balance
    data(oxBal_index,i)=D(oxBal_index,i)*100;

    DeleteCombustionAnalysis(ca);

    duration(i)=toc()
    disp([i duration(i)])
end

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

close(winH);
save('data-a.dat','data')
save('duration-a.dat','duration')
diap(strcat(["Total time : ",msprintf('%5.2f',sum(duration)/60)," mn"]))
disp(strcat(["Average time : ",msprintf('%5.2f',mean(duration))," sec"]))
```

```
xdel();
plot(data(oxBal_index,:),data(er_index,:),'r.');
[a,b,s]=reglin(data(oxBal_index,:),data(er_index,:));
xx=linspace(min(data(oxBal_index,:)),max(data(oxBal_index,:)),100);
yy=a*xx+b;
plot(xx,yy,'b');
disp([(1.1-b)/a;(0.9-b)/a]);
xgrid(33,1,8);
```