RPA-C

RPA-C v.2 API for Scilab

Table of Contents

Configuration of environment

To use RPA-C v.2 API 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_addComponent

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
```