Manual for `Optima++`

Version 2.1.0

Máté Papp, Tamás Varga, Ágota Busai, István Gyula Zsély,
Tibor Nagy, Tamás Turányi

2021.05.03.

Authors

The following people have contributed to the development of `Optima++`

- Máté Papp
- Tamás Varga
- Ágota Busai
- Viktor Samu
- Carsten Olm
- István Gy. Zsély
- Tibor Nagy
- Tamás Turányi

`Optima++` is being developed at the ELTE Reaction Kinetics Laboratory, and development is active and ongoing as of this release.

Operating systems

`Optima++` is currently developed under Ubuntu. It is tested under Windows 7 and Windows 10. The code is expected to run on unix-type systems (including OS X) but no extensive testing was carried out. Any reports of issues or successful usage on different systems is welcome.

External libraries

`Optima++` uses the `tinyxml2` library (by Lee Thomason - www.grinninglizard.com) for XML processing which is freely available under the zlib licence.
`Optima++` uses the `Eigen` library (eigen.tuxfamily.org) for matrix-vector operations, which is freely available licensed primarily under MPL2 with certain components licensed under LGPL3+.
`Optima++` uses the `fmt` library (https://github.com/fmtlib/fmt) for text formatting which is distributed under the MIT license.
`Optima++` uses the `range-v3` library (by Eric Niebler - https://github.com/ericniebler/range-v3) which are under the are under the Boost Software License.
`Optima++` uses the `yaml-cpp` library (https://github.com/jbeder/yaml-cpp) for processing input files of `Zero-rk` which is licensed under MIT license.
`Optima++` uses the `boost` C++ libraries (https://www.boost.org/, version 1.75.0) which is freely available under the Boost Software License.
The external libraries were added as submodules to the `Optima++` git repository and they are located in the `thirdparty` directory.
The licenses of the external libraries can be found in their respective directories of the `Optima++` source code distribution.

The graphical interface of `Optima++` uses the open source version of the `Qt Framework` (https://www.qt.io/) which uses the GNU Lesser General Public License v.3 (with some parts provided under GPL license).
The graphical interface of `Optima++` uses the `qcustomplot` library which is freely available under the GPL license.

Contents

Introduction

`Optima++` is a general framework for manipulating experimental data related to combustion chemistry, carrying out simulations of such experiments, performing model optimization and analysis, and providing auxiliary features for the above tasks. While the main focus is on combustion chemistry, most features should be applicable in any field of gas phase kinetics.

It is written in C++, and utilizes `FlameMaster` (http://www.itv.rwth-aachen.de/en/downloads/flamemaster/), `OpenSmoke++` (http://creckmodeling.chem.polimi.it/111-category-most-recent-publications/354-most-recent-publications-cuoci-cpc2015) or `Zero-rk` (https://github.com/LLNL/zero-rk) for simulations.

The code is developed and used primarily on unix-type systems, but is generally compatible with Windows systems too.

Installation

Installation using the online installer

Step 1 Download the installer suitable for your operating system from http://respecth.hu/.

Step 2 Run the installer and follow the installer's instructions.

Step 3 After the installation process finishes, you can start the graphical user interface of `Optima++` by running `Optima++.exe` executable located in the `GUI\bin\` folder of your Optima++ installation.

Building from source

For compiling `Optima++` on unix-type systems (including OS X) a compiler which supports the c++17 standard is required. We suggest using `gcc-10` (`g++`) or newer. The compilation of `Optima++` was also tested with clang-9 on Ubuntu. On Windows `MinGW` is recommended which is a port of `gcc` and is most conveniently used together with `Code::Blocks` (On windows the `gcc-8` filesystem library seems to be broken, use `gcc-7` or `gcc-9` instead). Building the code can be performed using `ccmake` in a command line (or with the cmake-gui on windows).

The libraries used in `Optima++` are included in the distribution package and their path relative to the project directory is currently hardcoded in the `CMakelists.txt`, so changing them is not recommended.

After compilation the executable file should be found in `bin/Release` directory. (If you build `Optima++` in a separate folder, copy the `bin/Release/OptimaPP` to the source directory.)

To use `FlameMaster` with `Optima++`, you first must download it from http://www.itv.rwth-aachen.de/en/downloads/flamemaster/, and compile it following the instructions in the `FlameMaster` package. To download `FlameMaster` you need to request a password from Prof. Heinz Pitsch. The request form is available on the `FlameMaster` download page.

NOTE: `Optima++` will only function correctly with `FlameMaster V4.0 BETA or newer`. Also, compilation must be carried out using the `-DOPTIMAPP=ON` option with `cmake`.

IMPORTANT NOTICE: The `ScanMan` executable of `FlameMaster` seems to behave incorrectly if compiled with "`Release`" build settings. If problems occur during compilation of mechanisms, try compiling `ScanMan` with "`Debug`" build settings.

After compilation, place the `FlameMan`, `ScanMan` and `CreateBinFile` executables into the `executables/` directory of the `Optima++` package. On Windows systems the executable names should have the `.exe` extension, and in case CygWin was used to compile `FlameMaster`, the `cygwin1.dll`, `cygstdc++-6.dll` and `cyggcc_s-seh-1.dll` files

should also be copied from the `/bin` subdirectory of `cygwin` to the `executables` directory, or added to the `PATH` system variable.

Compilation of `FlameMaster` mechanisms also required `perl` to be installed.

To use `OpenSmoke++` with `Optima++`, you must request the OpenSMOKE++ 0.12.0 version from prof. Alberto Cuoci (alberto.cuoci@polimi.it). Copy the content of OpenSMOKE++ Suite bin folder to the executables/opensmoke directory. On linux the `LD_LIBRARY_PATH` of `OpenSmoke++` is hardcoded in optima, the content of OpenSMOKE++ Suite lib folder should be copied to the executables/opensmoke/lib directory.

To use `Zero-rk`with `Optima++`, download the source code from https://github.com/LLNL/zero-rk, compile and install it according the `Installation Notes` section on their GitHub page.
After installation, set the path in the 5^{th} line of

```
executables/zero-rk_steady.py
executables/zero-rk_unsteady.py
```

files to point to the python library folder of `Zero-rk`.

Usage

Depending on the tasks to be performed Optima++ may use four sources of information:

1. RKD Format XML files for the description of experimental conditions.
2. Chemkin-II or FlameMaster format chemical reaction mechanism files.
3. An Optima++ input file describing the tasks to carry out.
4. Input file of the integrator settings.

1. RKD Format XML files

Optima++ accepts all valid RKD Format XMLs up to version 2.3. The conformity of the file to the appropriate ReSpecTh Kinetics Data Format Specification is strictly checked when the files are read (to be more precise, only the structure of the XML will be verified). Only error free files are accepted. The current specification can be downloaded from this link: http://respecth.chem.elte.hu/respecth/reac/ReSpecTh_Kinetic_Data_Format_Specification_v2.2.pdf.
Thousands of RKDF XML files can be downloaded from the ReSpecTh repository: www.respecth.hu.

2. Chemical mechanism files

Chemkin-II or FlameMaster format chemical mechanism files are needed as the source of the chemical reaction, thermodynamic and transport information. For its specification please refer to the documentation of the appropriate solver package.

3. Optima++ input file

To carry out any task with Optima++ either input files are required or there is a possibility to call Optima++ from command line for limited tasks (see section Calling Optima++ from command line).
To run Optima++ with the input file "INPUT_FILE" navigate to the package directory and give the following command on unix-type systems

```
bin/Release/OptimaPP INPUT_FILE
```

or on Windows

```
bin\Release\OptimaPP.exe INPUT_FILE
```

The executable should be run from the root directory of the package, as certain files are needed from the res/ and settings/ directories included in the release package, and the paths are currently hardcoded.

Input files are built from input blocks which are started with an appropriate keyword and terminated with an END keyword, with the exception of certain single-line blocks.

Each block has individual syntax, but they typically contain lines defining settings for the block and lines defining tasks to be carried out by Optima++.

NOTE: When referring to files in inputs of Optima++ all paths should be relative to the directory from where Optima++ is being run, which is not necessarily the directory where the executable is.

A typical layout of an input block is as follows:

```
BLOCK_KEYWORD
  SETTINGS_LINE_1 on
  SETTINGS_LINE_2 45

  DO_THIS ON_THIS
  DO_THIS ON_THIS_TOO
END
```

Any number of input blocks can be present in a single input file. Blocks are executed in strict order, meaning that every task will be completely executed in a block before moving onto tasks defined by the next block in the input file. This is so, that it is possible to have blocks depend on previous ones, without having to call Optima++ multiple times. Such a strict ordering is not enforced for tasks within blocks, but is usually adhered to.

The individual input blocks are described in the following sections, with respect to their syntax and functionality.

4. Integrator settings

Many functions of Optima++ involve simulations of combustion experiments. The solver packages that can be used for the simulations generally have several settings such as numerical tolerances for the integration that is performed during the simulations or parameters of the diffusion model used for 1D flame simulations.

Optima++ is capable of reading, interpreting and utilizing integrator settings. At startup Optima++ attempts to read the res/SettingsList file (path relative to the running directory), which should contain the list of integrator settings files that are to be read and interpreted, and potentially used in tasks. Such a SettingsList file is provided with the Optima++ package, along with the settings files it refers to, that covers most typical cases and provides reasonable settings.

Each line of the SettingsList file should contain four string tokens, which are the path and name of a settings file, the settings tag associated with the file, the abbreviation of the respective solver package and the type of measurement for which the settings should be used.

Examples

```
! FM settings
settings/FM_0D_settings   default    FM   ignition
```

```
settings/FM_0D_settings   default    FM   conc_prof
! OS settings
settings/OS_0D_settings   default    OS   ignition
settings/OS_0D_settings   default    OS   outlet_conc
! CKII settings
settings/CKII_0D_settings default    CKII ignition
settings/CKII_0D_settings default    CKII outlet_conc
```

The files that are referred to on each line must contain all integrator settings for the appropriate model type according to the syntax of the respective solver package. The example files provided in the settings directory of the `Optima++` release package show all settings that are required.

NOTE: A single file can be used for multiple purposes, as in the above example.

The currently handled solvers are `FM` (`FlameMaster`), `OS` (`OpenSmoke++`) and `CKII` (`Chemkin-II`). The valid experiment type strings are the following:

`ignition`	0D ignition delay simulation
`ignition_VTIM`	0D ignition delay simulation with volume-time history (usually RCM)
`conc_prof`	0D concentration-time profile simulation
`outlet_conc`	0D outlet concentration simulation
`psr`	perfectly stirred reactor simulation
`burning_velocity`	1D laminar flame simulation

All functions of `Optima++` that require chemical kinetic simulations or printing of input files will, by default, try to use the settings with the "`default`" tag that is for the appropriate solver and simulation type. This can be overridden with the `SETTINGS_TAG` keyword in any of the appropriate blocks, followed by the string tag that should be used. If no settings were read with the requested tag for the required solver package and/or simulation type then `Optima++` will attempt to use the "`default`" tag and if even those are not available, the "`inbuilt_default`" tag will be used, which refers to a set of hardcoded integrator settings (see class `IntegratorSettingsHandler` in the source code for the inbuilt defaults).

Input blocks

In this section the individual input blocks of the Optima++ input files are described. You can read information on the usage, the syntax, possible keywords, find examples and sometimes the description of their output.

TXT_TO_XML

Create XML datafiles according to the ReSpecTh Kinetics Data Format Specification version 2.3 from simple and easy to edit plaintext files.

Usage

In a TXT_TO_XML block each line should contain two strings. The first should be the full path of the input plaintext file and the second the full path of the desired output file. If there are further strings within a line, the line will be ignored.

Each source file will be read individually, and if the source file defines a valid and complete RKD file then a new RKD file will be created with the given output path and name.

The path to the output files must exist, as no directories will be created based on this input block. The paths given in the block are relative to the directory from where the program is called.

Single files can also be processed using the alternate syntax that can be seen below.

Processing a single file

```
TXT_TO_XML path_to_input/text_file_1 path_to_output/xml_file_1
```

Processing multiple files

```
TXT_TO_XML
SHOWWARNING
  path_to_input/text_file_1 path_to_output/xml_file_1
  path_to_input/text_file_2 path_to_output/xml_file_2
  path_to_input/text_file_3 path_to_output/xml_file_3
END
```

List of TXT_TO_XML keywords

SHOWWARNING

By default Optima++ only reports the error messages of a TXT_TO_XML job, but in certain cases it could be useful to have all the warnings as well. The SHOWWARNING keyword enables the reporting of warnings.
In the single-line case it is not possible to use this keyword.

Format of the input file

The input text files to be converted into XML files must follow the format described in this section. Example forms are provided in the release for all experiment types that are defined in the RKDFS v2.3.

The text files are processed line-by-line, and any text following a "!" or "%" character will be considered a comment and not interpreted.

The keywords and their usage is described in the table below. The arguments following the keywords are given in *italics*. Spaces in strings will be kept in the XML files. In case of species names, spaces can cause incorrect handling of species during the interpretation of XML files. Both decimal and scientific notations are accepted number formats.

`File author:` *string*
The string following the keyword will be used to identify the author of the XML file (which does not imply that the experimental data were obtained by this author).
`File DOI:` *string*
The string following the keyword will be used to specify a unique digital object identifier (DOI) that belongs to the dataset.
`Specification version:` *string*
The number following the keyword will be used to specify the ReSpecTh Kinetics Data Format Specification version the created file adheres to. The latest released specification version is 2.3. Two integers separated by a dot will be understood as `major_version.minor_version` and will be inserted into the XML file accordingly.
`File version:` *string*
The number following the keyword can be used to track the version history of a certain file. It should be set to 1.0 by default. A higher number should be chosen whenever a file was edited at a later stage. Two integers separated by a dot will be understood as `major_version.minor_version` and will be inserted into the XML file accordingly.
`Reference description:` *string*
The unformatted string following the keyword will be stored as the bibliographic reference of the measurements stored in the XML file. This may contain the name of the authors of the corresponding publication, journal name, page numbers, etc.
`Reference DOI:` *string*
The string following the keyword will be stored as the DOI of the corresponding publication. Providing this information is optional, and will not influence the interpretation of the XML files.
`Reference location:` *string*

The string following the keyword may help the users of XML files to locate the data in the corresponding publication (e.g. “Main article” or “Supplementary Material”). Providing this information is optional, and will not influence the interpretation of the XML files.

`Reference table:` *string*

The string following the keyword may help the users of XML files to locate the data in the corresponding publication (e.g. “Table 1, high-pressure series”). Providing this information is optional, and will not influence the interpretation of the the XML files.

`Reference figure:` *string*

The string following the keyword may help the users of XML files to locate the data in the corresponding publication (e.g. “Figure 2, blue circle”). Providing this information is optional, and will not influence the interpretation of the XML files.

`Experiment type:` *string*

The string following the keyword is used to define the experiment type.
The allowed types are:

- `ignition delay measurement`
- `laminar burning velocity measurement`
- `outlet concentration measurement`
- `concentration time profile measurement`
- `jet stirred reactor measurement`
- `burner stabilized flame speciation measurement`
- `rate coefficient determination`

`Apparatus:` *string*

The string following the keyword is used to provide the apparatus type in which the experiment was carried out (e.g. shock tube). Providing this information is optional, and will not influence the interpretation of the XML files.

`Operation mode:` *string*

The string following the keyword is used to provide the operating mode of the experimental apparatus in which the experiment was carried out (e.g. reflected shock). Providing this information is optional, and will not influence the interpretation of the XML files.

`Method:` *string*

The string following the keyword is used to describe the experimental method that was used in a direct measurement (e.g. “laser photolysis, laser-induced fluorescence”) or the method used in a theoretical determination study (e.g. “VTST”). Providing this information is optional, and will not influence the interpretation of the XML files.

`Common experimental conditions:` *varied*

The keyword can be followed by multiple lines, each specifying an experimental condition that is common across all individual experiments that are described within the file.
The lines should follow the following format (the order of the sub-keywords is arbitrary):

`Name:`*string* `Label:`*string* `Source_type:`*string* `Species:`*string*
`Value:`*number* `Unit:`*string* `Chemical_name:`*string* `CAS:`*string* `InChI:`*string*
`SMILES:`*string* `Reference:`*string* `Kind:`*string* `Bound:`*string*

The string after "`Name:`" defines what type of physical property is given (e.g. `temperature`, `pressure`).

The string after "`Label:`" is the short notation of the physical property (e.g. for a `temperature` property, the label is usually `T`).

The string after "`Source_type:`" defines the source type of the given physical property (`reported`, `digitized`, `estimated` or `calculated`).

The string after "`Species:`" defines which species a composition or concentration type of property refers to. For other types of properties it should not be used.

The number after "`Value:`" provides the numeric value of the property, and the string after "`Unit:`" provides the corresponding unit.
Valid property types and corresponding units are summarized in the table at the end of this section.

If the property is `concentration` or `composition` type then the string after "`CAS:`", "`InChI:`" and "`SMILES:`" can be used to identify species. The string after "`Chemical_name:`" denotes the chemical name of a certain species.

If the property name type is "`uncertainty`", the string after "`Reference:`" contains the name of the property to which the specified uncertainty refers, the string after "`Kind:`" describes the kind of uncertainty (`absolute` or `relative`), and the string after "`Bound:`" specifies the type of uncertainty bound(s) (`plus`, `minus` or `plusminus`).

The string after "`Kind:`" defines the nature of a physical property (e.g. if a pressure rise is `relative`).

`Varied experimental conditions and measured results:` *varied*

The keyword can be followed by multiple lines, each specifying an experimental condition that was varied between the individual experiments, or an experimental result.
The lines should follow the following format (the order of the sub-keywords is arbitrary):
`Name:`*string* `Source_type:`*string* `Species:`*string* `Unit:`*string* `ID:`*string*
`Chemical_name:`*string* `CAS:`*string* `InChI:`*string* `SMILES:`*string*
`Reference:`*string* `Kind:`*string* `Bound:`*string* `Label:`*string*

The string after "`Name:`" defines what type of physical property is given (e.g. `temperature`, `pressure`).

The string after "`Source_type:`" defines the source type of the given physical property (`reported`, `digitized`, `estimated` or `calculated`).

The string after "`Species:`" defines which species a composition or concentration type of property refers to. For other types of properties it should not be used.

The string after "`Unit:`" defines the unit for the numeric values that will be given later in the file.

The string after "ID:" provides an identifier for the property which will be used to identify which values correspond to which property.

The string after "Label:" defines a label for the species which is the suggested plot label for the given property. As no plotting is carried out by Optima++ providing a label is not necessary and the given label is not used (only stored in the XML files).

If the property is concentration or composition type then the string after "CAS:", "InChI:" and "SMILES:" can be used to identify species. The string after "Chemical_name:" denotes the chemical name of a certain species.

If the property name type is "uncertainty", the string after "Reference:" contains the name of the property to which the specified uncertainty refers, the string after "Kind:" describes the kind of uncertainty (absolute or relative), and the string after "Bound:" specifies the type of uncertainty bound(s) (plus, minus or plusminus).

The string after "Kind:" defines the nature of a physical property (e.g. if a pressure rise is relative).

Varied values:

A header line must follow the keyword, which must contain the IDs that were defined for the varied properties. The order of the ID strings defines the order in which the numerical values are provided for the varied properties. Common ID names are e.g. "x1, x2,...", but any name can be chosen.
The header line must be followed by lines with as many numerical values as many IDs were provided in the header, and in the corresponding order. Each line defines an experimental data point.

An example is provided below:

```
Varied values:
x1       x2
25.64    0.0466
37.34    0.0529
45.11    0.0591
49.89    0.0653
51.28    0.0713
50.00    0.0773
42.45    0.0832
33.30    0.0891
22.77    0.0948
```

Ignition definition:

The definition of the ignition delay can be specified with this keyword.
The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

MeasuredQuantity:*string* Type:*string* Amount:*number* Unit:*string*

The string following "`MeasuredQuantity:`" defines what physical property was used to define the ignition delay. This can be pressure, temperature or the concentration of a species. These are to be denoted by "`p`", "`T`", and the name of the species (e.g. "`OH`"), respectively. Alternative names of excited species are accepted according the following lists:

Excited CH: `CH*` `CHEX` `CHV` `CH(E)` `CH(A)`

Excited OH: `OH*` `OHEX` `OHV` `OH(E)` `OH(A)`

Excited CO2: `CO2*` `CO2EX` `CO2V` `CO2(E)` `CO2(A)`

In case there is no alternatively named species in the mechanism the non-excited species is used in the calculation of the ignition delay time (this fallback results a warning message only).

The string following type "`Type:`" defines which feature of the measured physical property is considered for the ignition delay. Valid values are summarized below:

- `max`
- `d/dt max`
- `baseline max intercept from d/dt`
- `baseline min intercept from d/dt`
- `concentration`
- `relative concentration`

See the RKD Format Specification for a detailed description of the types.

The number following "`Amount:`" defines the absolute or relative concentration value the target species has to reach for an ignition to occur. This attribute can only be used when the value of type is "`concentration`" or "`relative concentration`". The string following "`Unit:`" defines the corresponding unit.

`Time shift definition:`

The time shifting procedure can be specified with this keyword for a concentration time profile measurement.

The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

`MeasuredQuantity:`*string* `Type:`*string* `Amount:`*number*

The string following "`MeasuredQuantity:`" defines what physical property was used to define the time shift. This must be the name of a single measured species.

The string following type "`Type:`" defines which part of the specified species profile is matched with the experiments. The valid types are summarized below:

- `half`
- `inflexion`
- `relative`

See the RKDFS manual for a detailed description of the types.

The number following "`Amount:`" defines the absolute or relative concentration value the target species has to reach for ignition to occur. This attribute can only be used when the value of type is "`concentration`" or "`relative concentration`".

`Volume-time profile:`

A volume-time history can be defined with this keyword for an ignition delay experiment (usually an RCM experiment).

The keyword must be followed by lines defining a time and a volume property, in an identical way to property lines after "`Varied experimental conditions and measured results:`".

These lines must be followed by "`Profile:`", after which a header line must appear with the IDs defined here. This must be followed by the numeric values for the time–volume pairs.

An example can be seen below:

```
Volume-time profile:
dataGroupID: dg1   dataGroupLabel: V-t history   dataPointLink: 1
Type: time         Unit: s          ID: x4
Type: volume       Unit: cm3        ID: x5
Profile:
x4              x5
0.000000e+000   1.000000e+000
1.000000e-006   9.998782e-001
7.564500e-004   9.886456e-001
1.004150e-003   9.831007e-001
```

`Reaction:`

For direct rate coefficient determinations this keyword must be used to specify the reaction of which the rate coefficient is described.

The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

`Reaction string:`*`string`* `Order:`*`integer`* `Bulkgas:`*`string`*

The string following "`Reaction string:`" is the reaction string. The names of species on the same side of the reaction must be separated by "`+`" characters, and the reactant and product sides must be separated by an "`=`" sign, which can be bordered by "`>`" or "`<`" characters.
The reaction string must contain "`LP`" or "`HP`" before the first species separated by a space, if the rate coefficients measured are at the low pressure or high pressure limit respectively, for pressure dependent reaction rate coefficients (e.g. "`LP H+O2+M=HO2+M`").

The integer following "`Order:`" defines the order of the reaction. This should match with the reaction string, including high/low pressure limit specification. For fall-off reactions use the lower order (i.e. that which corresponds to the high-pressure limit).

The string following "`Bulkgas:`" specifies the major diluent gas for the experiment/theoretical determination. This only has significance for pressure-dependent rate coefficients and low-pressure limit rate coefficients. When calculating the rate coefficient the bulkgas will be taken into account through third-body collision efficiency effects, as if the whole gas composition was made up by the bulkgas.

NOTE: It is also possible to define a detailed composition through the common or varied conditions, and in this case the bulkgas will be completely ignored.

Summary of valid units

When printing XML files from text files, `Optima++` does not check explicitly is the specified units are valid within the RKDFS. As of version 2.1.0, the given units are printed into the XML files and if incorrect or unhandled units are given, then any solver input files printed from XML will not be correct or complete (a warning will be printed in such a case). Therefore it is important to use only those units that are handled within the RKDFS.

In the following table a summary of the unit strings that are currently handled within the RKDFS is given. Here all strings are given in the *exact way* as it should appear in the file. This means that exponents are not typed as superscript, and the micro (μ) prefix should be typed as "u" to guarantee that these can be typed in plain text files.

Property type	Valid units
temperature	K
pressure	Pa, kPa, MPa, Torr, torr, bar, mbar, atm
volume	m3, dm3, cm3, mm3, L
time	s, ms, us, ns, min
residence time	s, ms, us, ns, min
distance	m, dm, cm, mm
ignition delay	s, ms, us, ns, min
length	m, dm, cm, mm
density	g m-3, g dm-3, g cm-3, g mm-3, kg m-3, kg dm-3, kg cm-3, kg mm-3
flow rate	g m-2 s-1, g dm-2 s-1, g cm-2 s-1, g mm-2 s-1, kg m-2 s-1, kg dm-2 s-1, kg cm-2 s-1, kg mm-2 s-1
flame speed	m/s, dm/s, cm/s, mm/s, m s-1, dm s-1, cm s-1, mm s-1
composition	mole fraction, percent, ppm, ppb
concentration	mol/m3, mol/dm3, mol/cm3, mol m-3, mol dm-3, mol cm-3, molecule/m3, molecule/dm3, molecule/cm3, molecule m-3, molecule dm-3, molecule cm-3
rate coefficient	s-1, m3 mol-1 s-1, dm3 mol-1 s-1, cm3 mol-1 s-1, m3 molecule-1 s-1, dm3 molecule-1 s-1, cm3 molecule-1 s-1, m6 mol-3 s-1, dm6 mol-2 s-1, cm6 mol-2 s-1, m6 molecule-2 s-1, dm6 molecule-2 s-1, cm6 molecule-2 s-1
pressure rise	ms-1, s-1
all relative properties	unitless

CHECK_XML

Check the conformity of the XML datafiles to the appropriate ReSpecTh Kinetics Data Format Specification.

Usage

There are two legal styles for a CHECK_XML job. The first is a single-line command, where one argument follows the keyword. The other one is a CHECK_XML block, where each line should contain one string: the full path of the input XML file.
In both cases if there are further strings within a line (after the removal of potential comments), the line will be ignored.

Optima++ checks species appearing in the input XML files and their identifiers based on the speciesDatabase.xml file. The program warns if a given species is missing from the database or if there are conflicting identifiers.

Processing a single file

```
CHECK_XML path_to_input/xml_file_1
```

Processing multiple files

```
CHECK_XML
SHOWWARNING
  path_to_input/xml_file_1
  path_to_input/xml_file_2
  path_to_input/xml_file_3
END
```

List of CHECK_XML keywords

SHOWWARNING

By default Optima++ only reports the error messages of a CHECK_XML job, but in certain cases it could be useful to have all the warnings as well. The SHOWWARNING keyword enables the reporting of warnings.
In the single-line case it is not possible to use this keyword.

XML_TO_TXT

Create plain text files from ReSpecTh format files that can be used by `TXT_TO_XML`. The general purpose of this is to avoid directly editing XML files when data is to be modified, and rather edit more easily readable text files or convert the data in a non-tagged format which can be useful for people who are not familar with XML files. The conformity of the file to the appropriate ReSpecTh Kinetics Data Format Specification is strictly checked during the conversion (to be more precise, only the structure of the XML will be verified).

Usage

In an `XML_TO_TXT` block each line should contain two strings. The first should be the full path of the input XML file and the second the full path of the desired output file. If there are further strings within a line, the line will be ignored.

Each source file will be read individually. If the input file is a valid RKD file, then a plaintext file will be created with the requested output path. The file will contain the same information as the RKD input file and can be converted back using the `TXT_TO_XML` function of `Optima++`.

NOTE: When converting an RKD file using `XML_TO_TXT` and back with `TXT_TO_XML` the files might not be exactly identical, but semantically they will be the same. If `chemName`, `CAS`, `InChI` or `SMILES` for a species is missing `Optima++` automatically adds this information based on the species' `preferredKey` if it can be found in the `speciesDatabase.xml`.

The path to the output files must exist, as no directories will be created based on this input block. The paths given in the block are relative to the directory from where the program is called.

Single files can also be processed using the alternate syntax that can be seen below.

Processing a single file

```
XML_TO_TXT path_to_input/xml_file_1 path_to_output/txt_file_1
```

Processing multiple files

```
XML_TO_TXT
SHOWWARNING
  path_to_input/xml_file_1 path_to_output/txt_file_1
  path_to_input/xml_file_2 path_to_output/txt_file_2
  path_to_input/xml_file_3 path_to_output/txt_file_3
END
```

The file `speciesDatabase.xml` is used by `Optima++` to complement missing unambiguous identifiers for species. The `speciesDatabase.xml` file is located in the `Optima++` main directory and can be extended by the users using any ASCII text editor.

Its format must follow this example:

```
<database>
<species preferredKey="C3H6">
        <chemName>propylene</chemName>
        <CAS>115-07-1</CAS>
        <InChI>1S/C3H6/c1-3-2/h3H,1H2,2H3</InChI>
        <SMILES>CC=C</SMILES>
    </species>
<species preferredKey="C3H8">
        <chemName>propane</chemName>
        <CAS>74-98-6</CAS>
        <InChI>1S/C3H8/c1-3-2/h3H2,1-2H3</InChI>
        <SMILES>CCC</SMILES>
    </species>
</database>
```

List of XML_TO_TXT keywords

SHOWWARNING

By default Optima++ only reports the error messages of an XML_TO_TXT job, but in certain cases it could be useful to have all the warnings as well. The SHOWWARNING keyword enables the reporting of warnings.
In the single-line case it is not possible to use this keyword.

XML_TO_CKII

Create Chemkin-II input files from ReSpecTh format files. Inputs can be created for SENKIN (ignition delay, outlet concentration and concentration-time profile measurements), JSR and PREMIX (only burning velocity measurements currently).
The conformity of the file to the appropriate ReSpecTh Kinetics Data Format Specification is strictly checked during the conversion.

NOTE: Direct rate coefficient determination type XMLs cannot be used to create Chemkin-II input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_CKII block each line should contain two strings. The first should be the full path of the input plaintext file and the second the path of the directory where the output files should be printed. The path to the output files must exist, as no directories will be created based on this input block.

NOTE: Unlike TXT_TO_XML and XML_TO_TXT, here the path of the output directory must be specified and the name of the output file not. This is due to the fact that most RKD files contain the definition more than one experiment, therefore usually more than one output file will be produced.

The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file

```
XML_TO_CKII path_to_input/xml_file_1 path_to_output/output_directory
```

Processing multiple files

```
XML_TO_CKII
  path_to_input/xml_file_1 path_to_output/output_directory
  path_to_input/xml_file_2 path_to_output/output_directory
  path_to_input/xml_file_3 somewhere/else
END
```

XML_TO_FM

Create FlameMaster input files from ReSpecTh format files. Inputs can be created for 0D, PSR and freely propagating flame configurations.

NOTE: Direct rate coefficient determination type XMLs cannot be used to create FlameMaster input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_FM block each line should contain two strings. The first should be the full path of the input plaintext file and the second the path of the directory where the output files should be printed. The path to the output files must exist, as no directories will be created based on this input block.

NOTE: Unlike TXT_TO_XML and XML_TO_TXT, here the path of the output directory must be specified and the name of the output file not. This is due to the fact that most RKD files contain the definition more than one experiment, therefore usually more than one output file will be produced.

The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file

```
XML_TO_FM path_to_input/xml_file_1 path_to_output
```

Processing multiple files

```
XML_TO_FM
  path_to_input/xml_file_1 path_to_output
  path_to_input/xml_file_2 path_to_output
  path_to_input/xml_file_3 path_to_output
END
```

XML_TO_OS

Create OpenSmoke++ input files from ReSpecTh format files. Inputs can be created for 0D, PSR and freely propagating flame configurations.

NOTE: Direct rate coefficient determination type XMLs cannot be used to create OpenSmoke++ input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_OS block each line should contain two strings. The first should be the full path of the input plaintext file and the second the path of the directory where the output files should be printed. The path to the output files must exist, as no directories will be created based on this input block.

NOTE: Unlike TXT_TO_XML and XML_TO_TXT, here the path of the output directory must be specified and the name of the output file not. This is due to the fact that most RKD files contain the definition more than one experiment, therefore usually more than one output file will be produced.

The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file

```
XML_TO_OS path_to_input/xml_file_1 path_to_output
```

Processing multiple files

```
XML_TO_OS
  path_to_input/xml_file_1 path_to_output
  path_to_input/xml_file_2 path_to_output
  path_to_input/xml_file_3 path_to_output
END
```

EXP_INFO

Print the experimental conditions of measurements from ReSpecTh Kinetics Data Format Specification XML files.

Usage

When using EXP_INFO, two output files (using INFO_OUTPUT and RAW_OUTPUT keywords) and a list of input XML files must be defined. The XML files must be given with their full paths.

List of EXP_INFO keywords

INFO_OUTPUT

The file followed INFO_OUTPUT keyword will contain a summary of the experimantal condition ranges, one line per XML file.

RAW_OUTPUT

The file followed RAW_OUTPUT keyword will containd detailed print of conditions, one line per experimental point.

MECHMOD

Read and manipulate complex chemical kinetic mechanisms. The mechanism modification capabilities of the MECHMOD block are similar to those of the MECHMOD program written by Tamás Turányi (downloadable from www.respecth.hu) but not all features are available yet. A full coverage is planned for future releases of Optima++.

Apart from manipulation of mechanisms the MECHMOD block must be used to read and interpret mechanisms before they can be utilized by Optima++ for simulations.

Usage

When using MECHMOD, a new mechanism must be defined with the USE_NAME keyword followed by the name to be used for the mechanism. All operations within a MECHMOD block will be performed on this mechanism, and it can be referred to in subsequent blocks by the name assigned with USE_NAME.

After defining a mechanism, contents must be read into the mechanism using MECH_FILE followed by the path to a Chemkin-II format mechanism file. The file will be read and checked for errors. If no errors are found, then the interpreted mechanism will contain all species and reaction information and further operations will be possible on it.

Therefore, all MECHMOD blocks should follow the following basic layout:

```
MECHMOD
  USE_NAME myMechanism
  MECH_FILE path/myMechanismFile.inp
  !
  ! Do things with the mechanism here
  !
END
```

NOTE: Some mechanism contanins negative A parameter. This is not allowed in Optima++ because this parameter is stored internally in logarithmic ($\ln A$) form. By default, Optima++ terminates if a negative A parameter is found. You can change this behavior by passing the -forceM or -forceMechRead keyword as a commandline argument:

```
bin/Release/OptimaPP path/to/inputfile --forceMechRead
```

This is recommended only if you use precompiled mechanism, otherwise the negative A parameter will be passed to the solver as NaN.

List of MECHMOD keywords

USE_NAME

Specify name for the mechanism to be used. This name can be used to refer to the mechanism in all subsequent blocks. USE_NAME must be used to carry out any task with MECHMOD.

MECH_FILE

Read and interpret a Chemkin-II format mechanism file. Currently this is the only way to read contents into a mechanism defined with USE_NAME and must be used before any further operations can be carried out.

NOTE: The mechanism file must contain the thermodynamic data for all defined species. This does not adhere exactly to the Chemkin-II format, where it is possible to provide thermodynamic data in separate files.

NOTE: Thermodynamic data is read only for species that are actually defined within the mechanism (in the SPECIES block). Further NASA polynomials will be ignored and not interpreted.

NOTE: Most, but not all features of the Chemkin-II format are supported. Apart from basic reaction definitions the following keywords of the Chemkin-II format are interpreted by Optima++: DUPLICATE (or DUP); REVERSE (or REV); LOW; TROE; SRI; PLOG. Special reactants M and hv are interpreted as generic third bodies and photons respectively. Photons in photochemical reactions are preserved as reactants or products, but otherwise photochemical reactions are not handled by Optima++.

Example
```
MECH_FILE path/mechanismFile
```

TRAN_FILE

Read and interpret a Chemkin-II format transport file. A mechanism must already be correctly read through MECH_FILE for this keyword to do anything, as the transport properties are assigned to species defined in the previously read mechanism file.

Example
```
TRAN_FILE path/transportFile
```

PRINT_CKII_MECH

Write the mechanism into a file in Chemkin-II format, including all thermodynamic properties. In case no modifications were made to the mechanism read using MECH_FILE (and potentially TRAN_FILE), then the contents of the mechanisms should be identical, except that thermodynamic properties for non-defined species will not be written and the formatting of the files can differ slightly.

Example
`PRINT_CKII_MECH path/targetMechanismFile`

`PRINT_CKII_THERMO`

Write the thermodynamic properties of all species into a `Chemkin-II` format file. The thermodynamic properties of species will be given in NASA polynomial format, and order in the same way as they were in the mechanism. Also, NASA polynomials will be printed only for those species that were actually defined in the `SPECIES` block of the mechanism.

Example
`PRINT_CKII_THERMO path/targetThermoFile`

`PRINT_CKII_TRAN`

Write the transport properties of all species into a `Chemkin-II` format file. The transport properties of species will be ordered in the same way as they were in the mechanism. Also, transport data will be printed only for those species that were actually defined in the `SPECIES` block of the mechanism.

Example
`PRINT_CKII_TRAN path/targetTransportFile`

`PRINT_FM_MECH`

Write the mechanism into a file in `FlameMaster` format. Note that the `FlameMaster` mechanism file contains only the kinetic information, but no thermodynamic or transport data.

Example
`PRINT_FM_MECH path/targetMechanismFile`

`COMPILE_CKII`

Compile the mechanism into a `Chemkin-II` format binary file. This does not include the transport properties, as `Chemkin-II` uses a separate binary file for transport properties.

NOTE: This functionality uses `ckinterp` to carry out the compilation, which is part of the `Chemkin-II` package. As `Chemkin-II` is not freely distributable, `ckinterp` is not provided in the `Optima++` package. To be able to use `Chemkin-II`, please place the `ckinterp` (or `ckinterp.exe` on Windows) into the `Optima++` package root.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Example
`COMPILE_CKII path/targetChemkinBinary`

`FORCE_USE_CKII_BIN`

Use precompiled `Chemkin-II` mechanism binary file from input.

Example
`FORCE_USE_CKII_BIN path/inputChemkinBinary`

`COMPILE_CKII_TRANBIN`

Create the `Chemkin-II` format binary output file containing transport properties.

Example
`COMPILE_CKII_TRANBIN path/targetChemkinTransportBinary`

`FORCE_USE_CKII_TRANBIN`

Use precompiled `Chemkin-II` transport binary file from input.

Example
`FORCE_USE_CKII_TRANBIN path/inputChemkinTransportBinary`

`COMPILE_FM`

Compile the mechanism into a `FlameMaster` format binary file. This includes transport properties if they were provided using `TRAN_FILE`. It should be noted that the transport properties are not necessary for compiling a `FlameMaster` mechanism, but if they are not provided, the mechanism will only be usable for 0D simulations.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Example
`COMPILE_FM path/targetFlameMasterBinary`

`COMPILE_FM_FROM_CKII`

Compile a `FlameMaster` binary mechanism from `Chemkin-II` format mechanism file.

Example
`COMPILE_FM_FROM_CKII path/targetFlameMasterBinary`

`FORCE_USE_FM_BIN`

Use precompiled `FlameMaster` binary mechanism file from input.

Example
`FORCE_USE_FM_BIN path/inputFlameMasterBinary`

`COMPILE_OS`

Compile the mechanism into a `OpenSmoke++` format xml file. This includes transport properties if they were provided using `TRAN_FILE`. It should be noted that the transport properties are not necessary for compiling a `OpenSmoke++` mechanism, but if they are not provided, the mechanism will only be usable for 0D simulations.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Example
`COMPILE_OS path/targetOpenSmokeKineticsFolder`

`FORCE_USE_PRECOMPILED_OS_MECH`

Use precompiled `OpenSmoke++` mechanism folder from input. (The folder should contain the following two files: `kinetics.xml` and `reaction_names.xml`.)

Example
`FORCE_USE_PRECOMPILED_OS_MECH path/inputOpenSmokeKineticsFolder`

`SWITCH_A_UNIT`

Change the unit of the A Arrhenius parameters in the mechanism, and convert all parameter values appropriately. The valid units are those that are available in `Chemkin-II`. In all cases the volume units are cm^3 and the time units are seconds.

NOTE: The exact units of the A Arrhenius parameters always depend on the order of the respective reaction.

Valid units are: `MOLES` ($\mathrm{cm}^3\,\mathrm{mol}\,\mathrm{s}$); `MOLECULES` ($\mathrm{cm}^3\,\mathrm{molecule}\,\mathrm{s}$).

Example
`SWITCH_A_UNIT MOLECULES`

WARNING: This keyword will only be usable with `PRINT_CKII_MECH` for writing `Chemkin-II` format mechanism files.

`SWITCH_E_UNIT`

Change the unit of the E Arrhenius parameters in the mechanism, and convert all parameter values appropriately.

Valid units are: `KELVINS`; `CAL/MOLE`; `KCAL/MOLE`; `JOULES/MOLE`; `KJOULES/MOLE`.

Example
`SWITCH_E_UNIT KJOULES/MOLE`

WARNING: This keyword will only be usable with `PRINT_CKII_MECH` for writing `Chemkin-II` format mechanism files.

`LUMPED_MECH`

Store reactions containing nonintegral stoichiometric coefficients. (This feature is on by default from version 2.1.0.)

`NO_LUMPED_MECH`

Remove each reaction containing nonintegral stoichiometric coefficients from the mechanism.

`REMOVE_SPECIES`

Remove one or more species from the mechanism, including all thermodynamic and transport properties and all reactions in which it is a reactant or product.

Species to be removed must be named on the same line as the `REMOVE_SPECIES` keyword and be separated by whitespace characters (except newline).

Example
```
REMOVE_SPECIES C3H8 C3H7 C3H6
```

`REMOVE_REACTION`

Remove one or more reactions from the mechanism.

Reactions to be removed must be named on the same line as the `REMOVE_REACTION` keyword and be separated by whitespace characters (except newline). The reaction string must not contain whitespace characters, and each species name must be written out separated by "`+`" signs (i.e. "`H+H`" should be used instead of "`2H`"). The reactants and products must be separated by "`=`", "`=>`" or "`<=>`". The different separators between reactants and products are treated as identical (i.e. "`H+O2=>OH+O`" is equivalent to "`H+O2=OH+O`").

The reactants and products can be given in an arbitrary order, and parantheses will be ignored. In case of duplicate reactions all instances will be removed.

Example
```
REMOVE_REACTION H+H+M=H2+M HO2+OH=H2O+O2
```

`KEEP_SUBMECH`

Specify one or more species to keep in the mechanism. All reactions will be removed from the mechanism that contain none of the specified species. Therefore, it is generally expected that some reactions will remain in the mechanism that have not been specified after `KEEP_SUBMECH`.

Species to be kept must be named on the same line as the `KEEP_SUBMECH` keyword and be separated by whitespace characters (except newline).

Example
```
KEEP_SUBMECH H O OH
```

`RENAME_SPECIES`

Renames a species in the thermodynamic and transport properties and all reactions in which it is a reactant or product. This can be useful e.g. to match species names used in XML files.

The `RENAME_SPECIES` keyword is followed by the old and the new name of the species, respectively. All three strings have to be separated by whitespace characters (except newline).

NOTE: Avoid using characters `C`, `H` and `O` in species names if they're not describing the elemental composition. This would affect the equivalence ratio calculated by the code, which is relevant for stepping in between conditions during the creation of a `FLAME_DATABASE`.

Example
```
RENAME_SPECIES CH3HCO CH3CHO
```

MECHTEST

MECHTEST can be used to perform simulations at the conditions of experiments using a detailed chemical mechanism.

The following experiment types can be used:

- Ignition delay measurement
- Laminar flame speed measurement
- Outlet concentration measurement
- Concentration time profile measurement
- Jet stirred reactor measurement
- Direct rate coefficient determination

Burner stabilized flame speciation measurement are not handled in the present release, but are planned for future releases.

Usage

To use MECHTEST several things need to be defined, including the mechanism and integrator settings to be used. The mechanism to be used can be defined with MECHANISM followed by a mechanism name defined in MECHMOD block (see USE_NAME).

List of MECHTEST keywords

MECHANISM

Specify the mechanism to be used for the simulation of the experiments. The name must match with one defined in a MECHMOD block with USE_NAME. All modifications carried out on the mechanism are kept for the simulations.

To use a mechanism for simulations it must also have been compiled with an appropriate COMPILE keyword in the MECHMOD block.

SOLVER

Specify solver package to be used. By default FlameMaster is used.

Accepted values are FM, OS and CKII, however CKII requires a modified version of the Chemkin-II package which is not distributed.

NOTE: The evaluation of direct rate coefficient determinations is independent of the selected solver package.

SETTINGS_TAG

Specify the settings tag to be used. See section 4. Integrator settings for details about configuration of solver settings. By default the "default" tag is used, and if such settings are not available the inbuilt defaults of Optima++ are used.

`THREAD_LIMIT`

Specify the number of parallel threads to be used by `Optima++` for the simulations. Default value is 1.

`GROUP_BY_POINTS`

The content of the mechtestResults file will be ordered as the datapoints occurs in the xml file (or in the order of the point numbers if specified after the `POINTS` keyword).

`GROUP_BY_SPECIES`

The content of the mechtestResults file will be groupped by species. (This is the default behaviour.)

`NOCLEANUP`

`Optima++` can produce enourmously large number of temporary files in the `JOBS` and `FMLOGS` directories. By default these files are cleaned up after finishing the corresponding jobs, but in certain cases it could be useful to keep them for further investigation. This case use the `NOCLEANUP` keyword.

`TIME_LIMIT`

Specify the maximum time in seconds allowed to the solvers. If the `TIME_LIMIT` is exceeded, the solver process is killed and the job is markes as `FAILED`.

NOTE: This option is enabled only if `Optima++` was compiled with `-DUSE_TIMELIMIT`. It works only on linux and with `OS` solver.

`NODISK`

By default, the input and output files used by the solver packages are saved to the `INPUTS`, `JOBS` and `LOGS` directories. With this option these files will be saved to a temporary filesystem in the memory without any physical disk usage.

NOTE: This option is limited to `FM` and `OS` solvers and works only on linux operating systems.

`FORCE(_NO)_RCM_SUB`

During the postprocessing step of the RCM simulation's results the time point belonging to the minimum of the volume history is substracted from the final ignition delay only if the minimum point is not in the last 5% of the volume-time history. Whith the `FORCE_RCM_SUB` keyword this time shifting is always performed. Use the `FORCE_NO_RCM_SUB` keyword to disable the time shifting.

`LOAD_DATABASE`

Laminar flame speed measurements require pre-existing solutions from a flame database. Use `LOAD_DATABASE` keyword to specify the name of the flame solution database to be used for the flame simulations. See `FLAME_DATABASE` for details on creating and utilizing flame databases.

`NAME`

Mark experiments defined in an RKD datafile for simulation. The specified file must be a valid RKD Format file. The name of the file must be followed by the `POINTS` keywords and the numbers of the experimental datapoints (according to their order in the respective file) separated by non-newline whitespace characters. The "`all`" (or "`ALL`") keyword can also be provided to use all datapoints from a file. The experimental datapoints can be selected by ranges (for example the "`1-10 15-20`" expression selects the first 10 datapoints and form the 15^{th} point to the 20^{th} point, respectively). The `POINTS` keyword optionally can be followed by `EXCLUDE` keyword and the numbers (or ranges) of the datapoints to be excluded from the calculations.

NOTE: Concentration-time profile measurements contain a single experiment and datapoints refer to different times at which concentrations were measured. This means that no matter how many points are selected, the same amount of simulation time is required, unlike other types of dataset where a single datapoint is a single experiment, and has to be simulated individually.

NOTE: The range expression should not contain any whitespaces between the range delimiter numbers and the "`-`" sign.

Examples

```
NAME myDatafiles/experiment_1.xml POINTS 3 4 6 7
NAME myDatafiles/experiment_2.xml POINTS 1 2 3
NAME myDatafiles/experiment_3.xml POINTS 1-5
NAME myDatafiles/experiment_4.xml POINTS all
NAME myDatafiles/experiment_5.xml POINTS all EXCLUDE 2-5
```

Output

A file called `mechTestResults` is created in the directory from where `Optima++` is called. If a file with the same name and path already exists then the new content will be appended to it.

First a header is printed describing the mechanism, integrator settings tag and solver package used for the simulations. Then the experimental and simulation results are printed line by line. On each line the name of the experiment file, the number of the current datapoint, the measured species (if applicable), and the measured and simulated values are given.

The units of the printed results (both measured and simulated) depend on the experiment type:

Experiment type	Unit of results
`Ignition delay measurement`	s
`Laminar flame speed measurement`	cm/s

`Outlet concentration measurement`
`Concentration time profile measurement`
`Jet stirred reactor measurement`
	Depending on whether mole fractions or absolute concentrations are given in the respective RKD datafile:
	mole fraction OR mol/m^3

`Direct rate coefficient determination`	Depending on the order of the respective reaction: s^{-1} OR $cm^3mol^{-1}s^{-1}$ OR $cm^6mol^{-2}s^{-1}$

Example

```
Mechanism used: sandiego
Integrator settings collection used: default
Solver package used: FM
res/x00000066.xml          1       - 2.89000e-005  5.87242e-005
res/x00000066.xml          2       - 2.03500e-004  3.47538e-004
res/x00000066.xml          3       - 3.93500e-004  5.30334e-004
```

SENSITIVITY

Carry out local sensitivity analysis of a given model at the conditions of selected experiments. The sensitivity analysis is carried out on the simulations of the measured values (i.e. the burning velocities if that was measured in a selected experiment) with respect to all A Arrhenius parameters in the selected model including low pressure limit Arrhenius parameters for falloff reactions. The calculations are performed using a brute-force method and the normalized sensitivity coefficients are provided.

The normalized sensitivity coefficients are calculated according to the following formula:

$$Sn_{ij} = \frac{A_j}{Y_i} \frac{Y_i' - Y_i}{A_j' - A_j},$$

where Y_i is the i-th simulation result on which the sensitivity analysis is carried out, and A_j is the j-th A Arrhenius parameter that is investigated.

Usage

The MECHANISM, LOAD_DATABASE, SOLVER, NODISK, SETTINGS_TAG, THREAD_LIMIT keywords are to be used in the same way as in a MECHTEST block. The NAME keyword is also used in an identical way but here it marks experiments for sensitivity analysis. The REACTION block can be used in many new ways compared to that of described in Reaction blocks of Optimization.

List of SENSITIVITY keywords

Keywords with identical functions as in MECHTEST are not listed here.

SPLIT

In simulation tasks where a single experiment must be simulated using several different parameter sets Optima++ uses a single thread by default. This means that simulations using different parameter sets are carried out one after another for a single experiment. In certain cases it can be beneficial to split the task so that simulation of a single experiment are also multithreaded.

SPLIT can be used to specify how many sub-sets to create from the complete parameter set. For example using SPLIT 4 in a case where 200 parameter sets are to be simulated, results in 4 separate simulation tasks (that Optima++ can run parallel) with 50 parameter sets each.

Using SPLIT only has an effect on how Optima++ manages the simulations internally. All results, both numerically and format-wise, are independent of it.
By default no splitting of tasks is done (which is equivalent to SPLIT 1).

PERTURBATION

Specifes the perturbation factor used in the brute-force method local sensitivity analysis. For example, PERTURBATION 0.10 represents a perturbation by 10%.

PERTURBATION is 0.05 by default (i.e. a perturbation by 5%) and the code also back to this value if nothing or a negative value is specified.

The perturbation factor used is printed to the output of the job. Note that "(default)" is added in brackets to mark that the code fell back to the default value. If PERTURBATION was specified correctly, only the used value is printed.

REACTION

Specifies the reaction(s) and parameter(s) for which the sensitivity analysis will be performed. Without any REACTION block each preexponential factor in the reaction mechanism (including the low pressure ones) is perturbed.

The REACTION block should be constructed in the same way as described in Reaction blocks of Optimization, but with the following differences:

- REACSTRING and PARAMETERS keywords are compulsary, REACNUM and RANGE are optional.
- REACSTRING may contain a POSIX type regular expression. Note, that matching of REACNUM and REACSTRING is checked after interpreting the POSIX type regular expression (if used).
- RANGE keyword can be used to specify a reaction number range.
- All specified parameters MUST exist in each selected reaction.

Possible types of REACSTRING:

- normal reaction string (e.g. H+O2=OH+O)
- species selector independetly if it is a reactant or product (e.g. ${species C2H4})
- species selector among the reactants (e.g. ${reactant C2H4})
- species selector among the products (e.g. ${product C2H4})
- a POSIX type regular expression ${REGEXP} (e.g. ${.*C[0-9]+.*} fits to reactions in which any number follows character 'C')
- to select all reactions use *

Example

```
! Selects all reactions which contain C2H4 as reactant
! and calculates sensitivity for parameters A and E
REACTION
REACSTRING  ${reactant C2H4}
PARAMETERS  A E
END
```

Example

```
! Selects all reactions in the reaction number range 1 to 10 (inclusive)
! and calculates sensitivity for parameter E
REACTION
RANGE       1 10
```

```
REACSTRING *
PARAMETERS E
END
```

NOCLEANUP

Optima++ can produce enourmously large number of temporary files in the JOBS and FMLOGS directories. By default these files are cleaned up after finishing the corresponding jobs, but in certain cases it could be useful to keep them for further investigation. This case use the NOCLEANUP keyword.

Output

A file called sensitivityResults is created in the directory from where Optima++ is called. If a file with the same name and path already exists then the new content will be appended to it.

Similarly to the output of a MECHTEST block first a header is printed. It describes the mechanism, integrator settings tag, solver package used for the simulations and the perturbation factor used for the sensitivity calculations.

In the following lines the basic information about the experiment, and the normalized sensitivity coefficients are printed. The output lines contain the name of the used RKD file, the number of the datapoint of the present experiment in the used RKD file, the name of the measured species if applicable, the measured result, the simulated result obtained with the unperturbed parameters and finally the normalized sensitivity coefficients.

The normalized sensitivity coefficients with respect to the A-factors are printed in the same order as the respective reactions occur in the used mechanism. These are followed by the low-pressure limit A-factors, again in the same order in which the respective falloff reactions occur (non-falloff reactions do not have a low-pressure limit A-factor).

The units of the experimental and simulated results are the same as in the output of a MECHTEST block. See the description of the MECHTEST block for details.

OPTIMIZATION

The objective function

Carry out the optimization of certain parameters of a model against selected data. The optimization algorithm used by Optima++ is described in [1]. The algorithm involves the minimization of the least-squares type error function:

$$E(\mathbf{p}) = \sum_{i=1}^{N} \frac{1}{N_i} \sum_{j=1}^{N_i} \left(\frac{Y_{ij}^{\mathrm{mod}}(\mathbf{p}) - Y_{ij}^{\mathrm{exp}}}{\sigma(Y_{ij}^{\mathrm{exp}})} \right)^2 ,$$

$$\text{where } Y_{ij}^{\mathrm{mod/exp}} = \begin{cases} y_{ij}^{\mathrm{mod/exp}} & \text{if } \sigma(y_{ij}^{\mathrm{exp}}) \approx \text{ constant for all } j \\ \ln y_{ij}^{\mathrm{mod/exp}} & \text{if } \sigma(\ln y_{ij}^{\mathrm{exp}}) \approx \text{ constant for all } j \end{cases} \qquad i = 1, \ldots, N.$$

Here $\mathbf{p}$ is the vector of the parameters selected for optimizations, N is the number of datasets and N_i is the number of data points in the i-th dataset. The values y_{ij}^{exp} and $\sigma(y_{ij}^{\mathrm{exp}})$ are the j-th measured (experimental) data point and its standard deviation, respectively, in the i-th dataset. For the indirect measurement data, the simulated (modeled) value is y_{ij}^{mod}, and is obtained from a simulation using the detailed mechanism investigated. For the direct measurements, the corresponding modeled value y_{ij}^{mod} is calculated at a given temperature, pressure, and bath gas composition. In the formula of $E(\mathbf{p})$, values Y_{ij}^{mod} and Y_{ij}^{exp} were compared, which were derived from y_{ij}^{mod} and y_{ij}^{exp} values depending on the nature of error distribution characteristic for the type of experiment in which data set i was determined.

Optimization targets can be both indirect measurements either direct rate coefficient determinations (experimental or theoretical). The targets and the associated standard deviations must be defined using the NAME keyword, and each line with NAME defines one dataset. For details see the keyword descriptions.

During optimization a random sample of the selected rate parameters is created, and the E error function is evaluated at each parameter set. This is done by performing the simulations of each optimization target with each parameter set. The parameter set which provided the lowest E value is selected as the currently best value, and the following round of sampling will use it as the mean value.

Estimation of the covariance matrix

The estimation of the covariance matrix of the currently best parameters after each round of sampling and evaluation can be requested. If this is done, the new covariance matrix is used for the following round of sampling. The method for the calculation of the covariance matrix of parameters is described in [1] and the modifications introduced regarding the usage of systematic errors in [2] are also used in Optima++.

Focusing during parameter sampling

The optimization algorithm also includes an additional scaling of the covariance matrix for sampling purposes, which is colloquially referred to in the code and keywords as

"focusing". Before each random sampling of the parameters the standard deviations of each parameter is scaled with the following factor:

$$\log f_\sigma = -\frac{2N_{\text{focus}}}{N_{\text{parameters}}} \log N_{\text{sample}} \quad \text{or} \quad f_\sigma = N_{\text{sample}}^{-2N_{\text{focus}}/N_{\text{parameters}}},$$

where f_σ is the scaling factor with which the standard deviations of the sampled parameters are multiplied, N_{sample} is number of parameter sets generated within a sample, N_{focus} is the current "focus level", and $N_{\text{parameters}}$ is the number of parameters that are being sampled. After each iteration if none of the randomly generated parameter sets provided a smaller E error function value than in the previous iteration, then the focus level is increased, thus decreasing the range of the following random sampling. Conversely if the error function could be decreased, the focus level is decreased.

The keywords FOCUSLEVEL, FOCUS_MIN, FOCUS_MAX, FOCUS_BACKSTEP and FOCUS_FORWARDSTEP can be used to control the initial value of the focus level and the later changes. See the description of the keywords for details.

Usage

The MECHANISM, LOAD_DATABASE, SOLVER, SETTINGS_TAG, NODISK, TIME_LIMIT, THREAD_LIMIT and SPLIT keywords are to be used in the same way as in a MECHTEST or SENSITIVITY block. The NAME keyword is used in a similar way, but in an OPTIMIZATION block information on the uncertainties of the data must also be provided.

Keywords with identical functions as in MECHTEST and SENSITIVITY are not listed here.

Keywords in OPTIMIZATION block

SAMPLE_SIZE

Specify the number of random samples to generate and use for simulations during a single optimization cycle.

SEED

Specify the seed value for the initialization of the pseudo-random number generator. If it is 0 or not provided, the number of seconds since 00:00 hours, Jan 1, 1970 UTC will be used as seed value.

ITERATION

Specify how many optimization cycles are to be carried out at maximum.

FOCUSLEVEL

Specify the level of focusing at which to start the sampling. For details see the Focusing subsection of OPTIMIZATION.

FOCUS_MIN

Specify the lowest focus level that can occur during the optimization process. For details see the Focusing subsection of OPTIMIZATION.

FOCUS_MAX

Specify the highest focus level that can occur during the optimization process. For details see the Focusing subsection of OPTIMIZATION.

FOCUS_BACKSTEP

Specify the degree of reducing the focus level after an optimization cycle in which the value of the error function could be decreased compared to the previous cycle.

FOCUS_FORWARDSTEP

Specify the degree of increasing the focus level after an optimization cycle in which the value of the error function could not be decreased compared to the previous cycle.

CALCCOV

Request the calculation of a posterior covariance matrix of the parameters after each optimization cycle. The covariance matrix is calculated based on the currently best parameter set.

NOCLEANUP

Optima++ can produce enourmously large number of temporary files in the JOBS and FMLOGS directories. By default these files are cleaned up after finishing the corresponding jobs, but in certain cases it could be useful to keep them for further investigation. This case use the NOCLEANUP keyword.

NAME

Mark experiments defined in an RKD datafile to be used as optimization targets. The specification of the uncertainties and point selection within the file must be done on the same line on which NAME occurs. The points selected using a NAME keyword constitute one dataset, which has importance during the weighting of the error function.

Uncertainties must be assigned using the SIGM keyword followed by the numerical value of the assigned standard deviation of the experiment. If there are multiple measurements associated with a single experiment (e.g. H_2, O_2 and H_2O concentrations were measured in a single experiment) then the same number of standard deviation values must be given.
The scale of the standard deviations must be defined using the SIGMSCALE keyword, followed by "abs" for absolute scale errors, or "absln" for absolute errors on a natural logarithmic scale. If absolute errors are used the units of the standard deviations must also be defined using the SIGMUNIT keyword followed by the appropriate unit string (for valid unit strings, see Summary of valid units subsection of TXT_TO_XML block

description). SIGMUNIT must not be used together with "absln" as in this case the assigned errors are unitless.

The points to be used from the specified file must be selected using the POINTS keyword followed by the listing of the numbers of the points to be used or "all" (or "ALL"). Point ranges and the EXCLUDE keyword can be used as described in the MECHTEST block.

Examples

```
! Typical setup for ignition measurement
NAME experiment_1.xml SIGM 0.15 SIGMSCALE absln POINTS 3 4 6 7
! Typical setup for concentration measurement with two measurements in one datapoint
NAME experiment_2.xml SIGM 1E-4 4E-5 SIGMSCALE abs SIGMUNIT mole fraction POINTS 1 2 3
! Typical setup for burning velocity measurements
NAME experiment_3.xml SIGM 2.50 SIGMSCALE abs SIGMUNIT cm s-1 POINTS all
NAME experiment_4.xml SIGM 2.50 SIGMSCALE abs SIGMUNIT cm s-1 POINTS all EXCLUDE 1 2
```

NOTE: The SIGM, SIGMSCALE and SIGMUNIT keywords can also be used inside the MECHTEST block and the error function values will be calculated for each datasets and will be written into the "errfValues" file in the output folder. The pointwise error function values will be saved to "errfValues_by_points" file.

REACTION

Parameters can be selected for optimization using REACTION sub-blocks within an OPTIMIZATION block. Similarly to other input blocks in Optima++ it must begin with the keyword itself (i.e. REACTION) and terminated with an END keyword. Several features must be defined within a REACTION block, using keywords exclusive to it. See Reaction blocks subsection for details.

Reaction blocks

Reaction blocks must be used to select parameters for optimization. Apart from selecting the parameters, initial values and uncertainty ranges must also be defined.

NOTE: Currently only parameters of reactions can be optimized, but optimization of thermodynamic and transport parameters is also planned for future releases.

Keywords in REACTION blocks

REACNUM

Specify the number of the reaction in the mechanism that is defined in the present reaction block. The specified reaction from the mechanism that is being optimized will be checked if it has an equivalent reaction string as defined by the REACSTRING keyword within the reaction block.

Example

```
REACNUM 43    ! This is reaction 43 - CH4+OH=H2O+CH3
```

`REACSTRING`

Specify the reaction string of the reaction that is defined in the present reaction block. The string will be checked if it matches with the reaction specified by `REACNUM` in the mechanism. The string does not need to match exactly with what appears in the mechanism, but must have the same meaning. E.g. "`OH+OH=HO2+H`" is considered equivalent to "`2OH=H+HO2`", but not to "`OH+OH=H2O2`".

NOTE: Spaces are allowed in the reaction string. All non-comment strings on the line starting with the `REACSTRING` keyword will be considered to be part of the reaction string.

Example

```
REACSTRING CH4+OH=H2O+CH3    ! This is reaction 43 - CH4+OH=H2O+CH3
```

`PARAMETERS`

Specify the parameters of the reaction that are to be optimized. The `PARAMETERS` keyword must be followed by the space separated list of the names of the selected parameters. The following strings are valid:

- `A`
- `lnA`
- `n`
- `E`
- `LPA`
- `LPlnA`
- `LPn`
- `LPE`
- `m thirdBodyName`

The `A`, `n` and `E` strings refer to the respective Arrhenius parameters. Parameters with an `LP` prefix refer to the low-pressure limit parameters.

NOTE: The low-pressure limit parameters can only be used for fall-off reactions, i.e. a reaction where both a high- and low-pressure limit rate coefficient is defined. If a reaction is defined with "`+M`" as a reactant but no fall-off behavior then the simple `A`, `n` and `E` parameters instead of `LPA`, `LPn` or `LPE`.

For A Arrhenius parameters, either `A` or `lnA` can be defined. The numerical values will have to be provided on the appropriate scale (linear or logarithmic, respectively), but otherwise the two types of definition are identical in function.

The `m` string refers to a third body collision efficiency parameter. It must be followed by the name of respective collider species as it appears in the used mechanism and separated with a space (in place of the "`thirdBodyName`" on the list above).

NOTE: The third body collision efficiency must already be defined in the mechanism for the respective species for the selected reaction, even if with unity efficiency.

Examples

```
PARAMETERS A n E ! All Arrhenius parameters of a reaction
PARAMETERS A E LPA LPE! High- and low-pressure A and E
PARAMETERS LPlnA LPn LPE m AR ! LP params and a third body efficiency
```

UNC_MEAN

Define the mean of the uncertainty range within which the optimization of the parameters will be carried out.

UNC_MEAN must be followed by as many numerical values as many parameters were defined in the reaction block, on the same line. As the number of parameters must be known, the PARAMETERS keyword must precede UNC_MEAN.

Example

```
UNC_MEAN 1.60E+07 1.8 1400.0
```

UNC_COVMAT

Define the covariance matrix of the parameters that represents the uncertainty range around the mean value of the parameter values defined with UNC_MEAN. Starting with the line after UNC_COVMAT, an n-by-n matrix must be given, where n is the number of parameters defined in the reaction block. As the number of parameters must be known, the PARAMETERS keyword must precede UNC_MEAN.

Example

```
UNC_COVMAT
  24.7803636544931    -3.09077749533357    3784.78679570716
 -3.09077749533357    0.385646993305109   -473.059491605493
  3784.78679570716    -473.059491605493    584938.139905027
```

INIT

Define the initial values of the parameters that will serve as the starting point of the optimization, i.e. the mean values for the first random sampling.

NOTE: While the meaning of INIT is different from UNC_MEAN, the numeric values can identical.

Example

```
INIT 1.60E+07 1.8 1400.0
```

INIT_COVMAT

Define the covariance matrix according to which the first round of random sampling will be carried out during the optimization.

Example

```
INIT_COVMAT
  24.7803636544931    -3.09077749533357    3784.78679570716
 -3.09077749533357    0.385646993305109   -473.059491605493
  3784.78679570716    -473.059491605493    584938.139905027
```

`TEMPRANGE`

Define the temperature range in which the uncertainty range of the rate coefficients (defined with `UNC_MEAN` and `UNC_COVMAT`) must be enforced during the random sampling.

The keyword `TEMPRANGE` must be followed on the same line by two numeric values which must be the lower and upper temperature limits respectively, given in K.

Example
```
TEMPRANGE 800 2500    ! 800-2500 K range
```

`SIGMARANGE`

Define the cutoff range in terms of standard deviations for the sampled parameters.

NOTE: Using a `SIGMARANGE` of less than 2 is generally not recommended as it will become very difficult to obtain valid samples.

Example
```
SIGMARANGE 3
```

`UNIFORM`

If the keyword `UNIFORM` appears in a reaction block, then the parameters will be sampled according to a uniform distribution. Otherwise, the sampling is done according to a normal distribution.

Uncertainty and sampling of rate parameters

Optimization of detailed kinetic models typically involves the optimization of Arrhenius parameters of important reaction steps.

Such an optimization requires bounds on the investigated parameter space, however it is important to consider that the Arrhenius parameters themselves are not physically meaningful, only the rate coefficients defined by them. `Optima++` uses the characterization of the uncertainty parameters as described by Nagy and Turányi [3]. The uncertainty of rate coefficients is considered to be generally temperature dependent, but it can be characterized by the covariance matrix of its Arrhenius parameters which is temperature independent.

Characterizing uncertainties of rate coefficients

Rate coefficients are expressed via the extended Arrhenius expression:

$$k = A \cdot T^n \cdot \exp\left(-\frac{E}{RT}\right).$$

It is practical to linearize the equation, by taking the logarithm of it:

$$\ln k = \ln A + n \ln T - \frac{E}{RT}.$$

Since the above equation is linear (at a given temperature), there must be a linear relationship between the variance of $\ln k$ and the variances and covariances of the parameters of the equation, i.e. $\ln A$, n and E. For convenience E/R can be used as a parameter instead of E to avoid dependence on the choice of energy units, and the following notations can be introduced:

$$\kappa := \ln k, \quad \alpha := \ln A, \quad \varepsilon := \frac{E}{R}.$$

Using the above notations, the covariance of the Arrhenius parameters can be written in the following form:

$$\mathbf{\Sigma_p} = \begin{bmatrix} \sigma_\alpha^2 & r_{\alpha n}\sigma_\alpha\sigma_n & r_{\alpha\varepsilon}\sigma_\alpha\sigma_\varepsilon \\ r_{\alpha n}\sigma_\alpha\sigma_n & \sigma_n^2 & r_{n\varepsilon}\sigma_n\sigma_\varepsilon \\ r_{\alpha\varepsilon}\sigma_\alpha\sigma_\varepsilon & r_{n\varepsilon}\sigma_n\sigma_\varepsilon & \sigma_\varepsilon^2 \end{bmatrix}.$$

Where r-s denote the correlations between the respective parameters. The following vector can be introduced for convenience:

$$\mathbf{\Theta} = \begin{bmatrix} 1 & \ln T & -T^{-1} \end{bmatrix}^\intercal.$$

The following relation exists between the covariance matrix of the Arrhenius parameters and the variance of the rate coefficient at a given T temperature:

$$\sigma^2(\ln k) = \mathbf{\Theta}^\intercal \mathbf{\Sigma_p} \mathbf{\Theta}.$$

The matrix-vector multiplications can be expanded, giving the following form for the above equation:

$$\sigma(\ln k)^2 = \sigma_\alpha^2 + \sigma_n^2 \ln^2 T + \sigma_\varepsilon^2 T^{-2} + 2r_{\alpha n}\sigma_\alpha\sigma_n \ln T - 2r_{\alpha\varepsilon}\sigma_\alpha\sigma_\varepsilon T^{-1} - 2r_{n\varepsilon}\sigma_n\sigma_\varepsilon T^{-1} \ln T.$$

Based on the above equations the variance and standard deviation of a rate coefficient can be calculated from the covariance matrix of its Arrhenius parameters. The same equation can be used if not all Arrhenius parameters are used for a particular rate coefficient (e.g. only A and n are non-zero), but in this case the variance of the non-used parameters can be considered to be zero and a truncated from of the respective equations can be used.

Application of uncertainty ranges during optimization

In `Optima++` the uncertainty in which an optimization of Arrhenius parameter can be carried out is defined via a mean value of the parameters and a covariance matrix, by using the `UNC_MEAN` and `UNC_COVMAT` keywords. These can be used to calculate the standard deviation of the respective lnk value at all temperatures. Additionally, a cutoff in terms of standard deviations has to be defined and a temperature range where the uncertainty range is considered valid. These can be defined with the `SIGMARANGE` and `TEMPRANGE` keywords, respectively.

Each parameter set that is generated by `Optima++` during the random sampling phase of an optimization is checked, if they provide rate coefficients that fall outside the uncertainty limits, then the respective element of the sample is discarded and a new one is generated, until all rate coefficients are within their respective uncertainty bounds at all temperatures within the specified temperature range.

Random sampling of rate parameters

There are two sampling methods available for rate parameters in `Optima++`.

The first is sampling according to a normal distribution defined by the mean values of the parameters and their covariance matrix. After generating a normal sample the Arrhenius parameters belonging to the same reaction are grouped together and are used to calculate the corresponding rate coefficient in the temperature range of the uncertainty limits (see `TEMPRANGE`).

The second option is uniform sampling of the Arrhenius parameters. In this case the sampling is carried out, by creating uniform samples of rate coefficients at as many temperatures as many Arrhenius parameters are used for the respective rate coefficient.

In case of three parameters the temperature values are the two extreme values of the temperature range and the middle of it on the inverse temperature scale. For two parameters only the extreme values are used. If only the A Arrhenius parameter is sampled, the rate coefficient and its uncertainty is temperature independent and the chosen temperature is arbitrary.

The uniform rate coefficient samples are made into groups of 3 or 2 depending on the number of parameters, which define a set of Arrhenius parameters which provide a sample for the given reaction. It has been shown in [4] that the distribution of Arrhenius parameters generated this way is also uniform.

Output

An output folder is created in the `outputs/` directory, with the same name as the input file. The different output contents are discussed in the following sections.

optimizationMonitor

A file called `optimizationMonitor` is created in the output folder. After each iteration a line is printed into this file, with the number of the iteration, the current focus level, the lowest error function value that was achieved in the iteration and the relative improvement in percent. A similar line is also printed for the initial parameter set.

Example

```
n_It  n_Focus     Errf_value Improvement
init       10  3.344485e+002    0.00000%
   1        8  1.773861e+002  -46.96159%
```

optimalParameters

A directory called `optimalParameters/` is created in the output folder, and files with the name `optResults_iterationNumber` are created during each iteration (with `iterationNumber` replaced by the appropriate number). These files contain the optimal

parameter set after an iteration and the corresponding covariance matrix that was used for sampling in that iteration (*not* the newly calculated one if it exists).

The order of the parameters is the same as they were provided in the REACTION blocks.

NOTE: If CALCCOV is not used, then the covariance matrix will be the same as provided with the INIT_COVMAT-s of the REACTION blocks.

Example

```
Optimal parameters
 3.694889967458e+001 -5.533827744981e-001  8.393545487220e+003
Previous covariance matrix
 2.780392398426e+001 -3.462492639425e+000  4.194526761001e+003
-3.462492639425e+000  4.313436605710e-001 -5.234283631604e+002
 4.194526761001e+003 -5.234283631604e+002  6.404338925486e+005
```

If CALCCOV is used, then the covariance matrices are printed into the `optResults_iterationNumber` file during each iteration (with `iterationNumber` replaced by the appropriate number).

In addition to the covariance matrix calculated during the iteration, the contributions from the statistic errors and the systematic errors are also printed to this file.

parameterSets

A directory called `parameterSets/` is created in the output folder, and files with the name `errfValues_iterationNumber` are created during each iteration (with `iterationNumber` replaced by the appropriate number).

The files contain the parameter values of the sample, with one set on each line. The columns are in the same order as they are given in REACTION blocks.

errorFunctionValues

A directory called `errorFunctionValues/` is created in the output folder, and files with the name `errfValues_iterationNumber` are created during each iteration (with `iterationNumber` replaced by the appropriate number).

In the first line the total error function values are printed for each parameter. The order of the values for the parameter sets is the same as in the `parameterSets/parameters_X` files. In the following lines the contributions to the total error function value from the individual datasets are printed.

Example

```
ErrfValue    1.676209e+002    5.025798e+002    1.605620e+002
x10000008    3.829500e+000    8.314917e+000    5.834598e-001
x10000025    1.355385e+001    2.481779e+001    8.122800e+000
```

FLAME_DATABASE

FLAME_DATABASE can be used to build up a database of flame simulation solutions. In this version FLAME_DATABASE can be used for FlameMaster and OpenSmoke++ simulations only.

For Chemkin-II simulations the user has to build up the collection of restart files outside of Optima++. The restart files have to place in the directory PREMIX_RESTARTS and their naming must follow this convention: rest.bin_m_USENAME_f_EXPID.

USENAME: the name of the mechanism defined with USE_NAME

EXPID: the identifier of the experimental point including the name of the XML and the number of experimental point. E.g. use EXPID x23003038_p4 for the 4th point of x23003038.xml.

Usage

The MECHANISM, SOLVER, SETTINGS_TAG, THREAD_LIMIT keywords are to be used in the same way as in a MECHTEST block. The NAME keyword is also used in an identical way but here it marks experiments for flame simulations to be included in the solution database.

The first solution file has to be copied by the user to the database since FlameMaster can perform flame simulations only using an existing solution file. With OpenSmoke++, the calculation can be performed with empty database.

List of FLAME_DATABASE keywords

Keywords with identical functions as in MECHTEST are not listed here.

CREATE_DATABASE

As a first step of flame simulations with FlameMaster the appropriate directory structure of a new database has to be created and at least one solution file have to be provided. Using CREATE_DATABASE Optima++ performs these tasks with the name given after the keyword. One sample solution file for a hydrogen flame coming with Optima++ is copied to the new database, but the user can replace it with more appropriate one(s) before using the database (modify file solutions and copy the new solution files in directory solutionFiles). Please, do not specify any simulations if you use CREATE_DATABASE keyword, they won't be performed.

FORCE_RECALCULATE

By default if a solution file correspond to the conditions of a given experiment is found in the database Optima++ doesn't carry out a simulation, but takes the stored burning velocity from the database. Using FORCE_RECALCULATE Optima++ takes the most similar solution from the database as a guess, but performs the simulation.

LOAD_DATABASE

Laminar flame speed measurements require pre-existing solutions from a flame

database. Use LOAD_DATABASE keyword in the FLAME_DATABASE block to utilized a flame database for the forthcoming MECHTEST, SENSITIVITY or OPTIMIZATION blocks. You have to specify the name of the flame solution database after this keyword.

NUMBER_OF_RECALCULATIONS

For properly converged flame simulations FlameMaster sometimes needs to run the simulations more than once. Use NUMBER_OF_RECALCULATIONS keyword to force Optima++ for multiple calculations with FlameMaster for one condition. The default value is 1.

NOCLEANUP

Optima++ can produce enourmously large number of temporary files in the JOBS and FMLOGS directories. By default these files are cleaned up after finishing the corresponding jobs, but in certain cases it could be useful to keep them for further investigation. This case use the NOCLEANUP keyword.

STORE_ONLY

If specified, the solutions will be saved to the database, but the calculations are performed without using the database. By default Optima++ takes the most similar solution from the database as a guess. If the conditions in the solution files of the database are far from the conditions defined in the given experiment(s), the OpenSmoke++ calculation can be slower than without the guess solution. In this case, the STORE_ONLY keyword can be used.

CONVERT_TO_FM

If specified, after loading the database, all entries will be converted to FlameMaster style flame database elements. It must be used together with SOLVER OS keywords.

Example

```
CONVERT_TO_FM name_of_the_new_database
```

XMLMOD

XMLMOD can be used to format, modify or update ReSpecTh Kinetics Data Format Specification xml files.

Example

```
XMLMOD
  [[additional keywords]]
  path_to_input/xml_file_1 path_to_output/xml_file_1
  path_to_input/xml_file_2 path_to_output/xml_file_2
  path_to_input/xml_file_3 path_to_output/xml_file_3
  path_to_input2/.*xml  path_to_output_folder2/
END
```

Without any additional keyword, XMLMOD rearranges the xml files to a more easily readable format.

List of XMLMOD keywords

COMPACT

Formats the xml files with the minimum naumber of white spaces.

UPDATE_BIBLIOGRAPHY_DETAILS

Updates or creates the <details> element in <bibliographyLink> block from a BibTeX file. The BibTeX entry is identified by the DOI number defined inside the <referenceDOI> text element. If the DOI occurs multiple times in the BibTeX file, a warning message is shown and the last occurence is used in the updating process.

Example

```
UPDATE_BIBLIOGRAPHY_DETAILS  path/to/bibtex_file.bib
```

UPGRADE_TO_V2.2

The UPGRADE_TO_V2.2 command performs some automatic upgrade from ReSpecTh Kinetics Data Format Specification version 1.0 to version 2.2.

- In the old version preferredKey attribute was used instead of description element in <bibliographyLink> block. The description tag is inserted automatically.
- The CAS, InChI, SMILES and chemName attributes are added to speciesLink elements.

DEFAULT_VALUES

Sets the content or attribute of an xml entity to a specified text if the entity is not present in the xml file. The parameters can be provided by key-value pairs, separated with ':'. The key can be any txt form element (see TXT_TO_XML block for details) or an xml selector (see the xml selectors section).

Example

```
DEFAULT_VALUES
    ! set the file author to ELTE, Budapest, Hungary
    File author: ELTE, Budapest, Hungary
    ! set the file version to 2.0
    File version: 2.0
    Specification version: 2.2
    Source_type: TODO
    ! set the file DOI to '10.24388/' followed by the name of the xml file
    File DOI: 10.24388/${xmlName}
    ! add a comment with the current date
    Comment: Updated on ${exec date}
END
```

OVERRIDE

Overwrites content or attribute of an xml entity with a specified text. The possible parameters are identical to the parameters descibed in DEFAULT_VALUES block.

Example

```
OVERRIDE
   ! change the label attribute of each temperature property to 'T'
   ${?property[name=temperature]}Label: T
   ! replace the preferredKey attribute AR with Ar in each
   ! speciesLink element
   ${?speciesLink[preferredKey=AR]}Name\: composition Species: Ar
END
```

DATA_FILE

The update information can be provided in a csv file. The first column contains the the name of the xml file and from the second, each column contains the text to be inserted into the file. The first line contains the updateable property. It can be any txt form element or an xml selector.

Example

```
xml       ; Reference DOI:                ; ${?property[id=x1]}Source_type:
x10004001 ; 10.1016/s0010-2180(70)80008-6 ; reported
x10004002 ; 10.1016/s0010-2180(70)80008-6 ; digitized
x10004003 ; 10.1016/s0010-2180(70)80008-6 ; reported
x10004017 ; 10.1007/s001930100108         ; calculated
```

Descripton of the possible xml selectors/commands

Each command should be surrounded by `"${"` and `"}"`. It is evaluated for each xml and the `$` expression is substituted with the evaluation result.

`Query selector (${?expr})`

If the expression is true, then the substitution is applied.

```
${?dataGroup>property[id=x1]}Source_type
```

For example, the expression above is applied only for the `Source_type` elements which are inside a property tag with `id="x1"` attribute wich has a `dataGroup` parent element.

`Text selector (${&expr})`

Inserts the content of the selected xml element. The parent (`">"`) and attribute (`"[]"`) selectors can also be used.

Example

```
${&fileAuthor} ! insert the text inside <fileAuthor> tag.
```

`${xmlName}`

Returns the name of the current xml file without extension.

`${exec command}`

Executes `command` as a shell command and returns the result. This is available only on linux.

Example

```
${exec date} ! insert the current date
```

ATOMFLOW

Fluxes of elements from species to species are calculated and written into file using the input format of the FluxViewer visualisation tool.
The flux of element A from species j to species k through reaction step i is defined by

$$A_{ijk} = \frac{n_{\mathrm{A},j} n_{\mathrm{A},k} r_i}{N_{\mathrm{A},i}} \tag{1}$$

where $n_{\mathrm{A},j}$ and $n_{\mathrm{A},k}$ are the number of A atoms in species j and k, respectively, $N_{\mathrm{A},i}$ is the number of A atoms on either side of reaction step i and r_i is the rate of reaction step i. The overall fluxes between two species at a given time or distance are the sum of the element fluxes calculated in each reaction step:

$$A_{jk}(t) = \sum_i A_{ijk}(t) \tag{2}$$

Example

```
ATOMFLOW
  MECHANISM mech
  THREAD_LIMIT      4

  ELEMENTS H C O

  NAME path_to_input/xml_file_1 POINTS ALL
  NAME path_to_input/xml_file_2 POINTS 1 2 3
  NAME path_to_input/xml_file_3 POINTS ALL EXCLUDE 3 4
END
```

The FluxViewer input file is saved to the *xml*_p_*n*_fluxviewer.txt file in the output folder, where *xml* is the name of the RKD xml file and n is the corresponding datapoint in the xml file.

List of ATOMFLOW keywords

The MECHANISM, SETTINGS_TAG, NODISK, TIME_LIMIT and THREAD_LIMIT keywords are to be used in the same way as in a MECHTEST or SENSITIVITY block.

ELEMENTS

Defines the list of elements for which the element fluxes are calculated.

Format	Whitespace separated list of strings
Default value	None
Type	Compulsory

Example

```
ELEMENTS C H O
```

Elements C, H and O are selected.

`T_INC`

Define the temperature increment of the `ATOMFLOW` output.

`Format`	Real number
`Default value`	0.0 (Each point is printed)
`Unit`	K
`Type`	Optional

Example

```
T_INC 30
```

The output is printed with 30 K temperature step.

`THRESHOLD`

Define the minimum value of element flux that will be printed.

`Format`	Real number
`Default value`	1e-30
`Unit`	mole/(cm^3 sec)
`Type`	Optional

Example

```
THRESHOLD       1e-20
```

Only $A_{jk}(t) > 10^{-20}$ will be printed.

Calling Optima++ from command line

In this section the keywords of the Optima++ command line inputs are described. You can read information on the usage, the syntax, and find examples.

The path for the output files will be a subdirectory of the outputs/commandLineJobs/ directory.
To call Optima++ from the command line with the command "COMMAND" navigate to the package directory and give the following command on unix-type systems

```
bin/Release/OptimaPP COMMAND
```

or on Windows

```
bin\Release\OptimaPP.exe COMMAND
```

The executable should be run from the root directory of the package, as certain files are needed from the res/ and settings/ directories included in the release package, and the paths are currently hardcoded.
The command "COMMAND" should include at least one Optima++ keyword and at least one input file.
There are different legal styles for "COMMAND", depending on the task.
The first is:

```
COMMAND = [optimapp_keyword] [input_file(s)] {-o [output folder]}
optimapp_keyword = TXT_TO_XML | CHECK_XML | XML_TO_TXT
                 | XML_TO_CKII | XML_TO_FM | XML_TO_OS | EXP_INFO
```

NOTE: The "{-o [output folder]}" part is optional and it cannot be used for the CHECK_XML keyword because it does not create output file.

The second is:

```
COMMAND = [optimapp_keyword] [targetMechanismFile]
optimapp_keyword = COMPILE_CKII | COMPILE_FM | COMPILE_OS
```

The last one is:

```
COMMAND = [PRINT_CKII_MECH | PRINT_FM_MECH] [targetMechanismFile] \
          [expression_ckii_mech | expression_fm_mech]
expression_ckii_mech = REMOVE_SPECIES [species]
                       | REMOVE_REACTION [reactions]
                       | KEEP_SUBMECH [species]
                       | SWITCH_A_UNIT [unit]
                       | SWITCH_E_UNIT [unit]
expression_fm_mech = REMOVE_SPECIES [species]
                     | REMOVE_REACTION [reactions]
                     | KEEP_SUBMECH [species]
```

TXT_TO_XML

Create XML datafiles according to the ReSpecTh Kinetics Data Format Specification version 2.3 from simple and easy-to-edit plaintext files.

Usage

In a `TXT_TO_XML` command the input line should contain two or more strings. The first should be the `TXT_TO_XML` keyword and the further are the full paths of the input plaintext files.

Each source file will be read individually, and if the source file defines a valid and complete RKD file then a new RKD file will be created in the directory specified after the `-o` option, if it is not specified the default `outputs/commandLineJobs/TxtToXml` output directory will be used. The file will contain the same information as the RKD input file and can be converted back using the `XML_TO_TXT` function of `Optima++`.

The name of the output files will be almost the same as the appropriate input files, only the file extension will be `.xml`.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP TXT_TO_XML path_to_input/text_file \
              -o path_to_output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP TXT_TO_XML path_to_input/text_file1 \
path_to_input/text_file2 path_to_input/text_file3 \
-o path_to_output_folder
```

The input text files to be converted into XML files must follow the format described in section Format of the input file.

CHECK_XML

Check the conformity of the XML datafiles to the appropriate ReSpecTh Kinetics Data Format Specification.

Usage

In a `CHECK_XML` command the input line should contain the keyword followed by one or more strings which should contain the full path of the input XML files.

`Optima++` checks species appearing in the input XML files and their identifiers based on the `speciesDatabase.xml` file. The program warns if a given species is missing from the database or if there are conflicting identifiers.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP CHECK_XML path_to_input/xml_file_1
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP CHECK_XML path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3
```

XML_TO_TXT

Create plain text files from ReSpecTh format files that can be used by `TXT_TO_XML`. The general purpose of this is to avoid directly editing XML files when data is to be modified, and rather edit more easily readable text files or convert the data in a non-tagged format which can be useful for people who are not familar with XML files. The conformity of the file to the appropriate ReSpecTh Kinetics Data Format Specification is strictly checked during the conversion (to be more precise, only the structure of the XML will be verified).

Usage

In an `XML_TO_TXT` command the input line should contain the keyword followed by one or more strings which should contain the full path of the input XML files.

Each source file will be read individually. If the input file is a valid RKD file, then a plaintext file will be created in the directory specified after the `-o` option, if it is not specified the default `outputs/commandLineJobs/XmlToTxt` output path will be used. The file will contain the same information as the RKD input file and can be converted back using the `TXT_TO_XML` function of `Optima++`.

NOTE: When converting an RKD file using `XML_TO_TXT` and back with `TXT_TO_XML` the files might not be exactly identical, but semantically they will be the same. If `chemName`, `CAS`, `InChI` or `SMILES` for a species is missing `Optima++` automatically adds this information based on the species' `preferredKey` if it can be found in the `speciesDatabase.xml`.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP XML_TO_TXT path_to_input/xml_file_1 \
-o output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP XML_TO_TXT path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3 \
-o output_folder
```

XML_TO_CKII

Create Chemkin-II input files from ReSpecTh format files. Inputs can be created for SENKIN (ignition delay, outlet concentration and concentration-time profile measurements), JSR and PREMIX (only burning velocity measurements currently).
The conformity of the file to the appropriate ReSpecTh Kinetics Data Format Specification is strictly checked during the conversion (to be more precise, only the structure of the XML will be verified).

NOTE: Direct rate coefficient determination type XMLs cannot be used to create Chemkin-II input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_CKII command the input line should contain at least one string after the keyword, which should be the full path of the input plaintext file(s). The path of the directory where the output files will be printed is the directory specified after the -o option, if it is not specified the default outputs/commandLineJobs/TxtToCKII output directory will be used.
The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP XML_TO_CKII path_to_input/xml_file_1 \
-o output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP XML_TO_CKII path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3 \
-o output_folder
```

XML_TO_FM

Create FlameMaster input files from ReSpecTh format files. Inputs can be created for 0D, PSR and freely propagating flame configurations.

NOTE: Direct rate coefficient determination type XMLs cannot be used to create FlameMaster input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_FM command the input line should contain at least one string after the keyword, which should be the full path of the input plaintext file(s). The path of the directory where the output files will be printed is the directory specified after the -o option, if it is not specified the default outputs/commandLineJobs/TxtToFM output directory will be used.
The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP XML_TO_FM path_to_input/xml_file_1 \
-o output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP XML_TO_FM path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3 \
-o output_folder
```

XML_TO_OS

Create OpenSmoke++ input files from ReSpecTh format files. Inputs can be created for 0D, PSR and freely propagating flame configurations.

NOTE: Direct rate coefficient determination type XMLs cannot be used to create OpenSmoke++ input files, as evaluating rate coefficients does not require a solver.

Usage

In an XML_TO_OS command the input line should contain at least one string after the keyword, which should be the full path of the input plaintext file(s). The path of the directory where the output files will be printed is the directory specified after the -o option, if it is not specified the default outputs/commandLineJobs/TxtToOS output directory will be used.
The names of the output files will be assembled from the name of the input RKD file (without extension) and a _p tag followed by the index of the experiment in the file and .inp as an extension. E.g. myExperiment_p3.inp for the third experiment from myExperiment.xml.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP XML_TO_OS path_to_input/xml_file_1 \
-o output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP XML_TO_OS path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3 \
-o output_folder
```

EXP_INFO

Print the experimental conditions of measurements from ReSpecTh Kinetics Data Format Specification XML files.

Usage

When using `EXP_INFO`, a list of input XML files must be defined. The XML files must be given with their full paths. Two output files will be created in in the directory specified after the `-o` option, if it is not specified the default `outputs/commandLineJobs/ExpInfo` path will be used. The `ExpInfo_info` file will contain a summary of the experimental condition ranges, one line per XML file (see `INFO_OUTPUT` keyword of the `EXP_INFO` block), and `ExpInfo_raw` will containd detailed print of conditions, one line per experimental point (see `RAW_OUTPUT` keyword of the `EXP_INFO` block).

Processing a single file on unix-type systems

```
bin/Release/OptimaPP EXP_INFO path_to_input/xml_file_1 \
-o output_folder
```

Processing multiple files on unix-type systems

```
bin/Release/OptimaPP EXP_INFO path_to_input/xml_file_1 \
path_to_input/xml_file_2 path_to_input/xml_file_3 \
-o output_folder
```

MECHMOD-type keywords

For details see MECHMOD.

Usage

When using MECHMOD-type keywords, a Chemkin-II format mechanism file must be defined as a command line argument preceded by the appropriate keyword. The file will be read and checked for errors.

COMPILE_CKII

Compile the mechanism into a Chemkin-II format binary file. This does not include the transport properties, as Chemkin-II uses a separate binary file for transport properties.

NOTE: This functionality uses ckinterp to carry out the compilation, which is part of the Chemkin-II package. As Chemkin-II is not freely distributable, ckinterp is not provided in the Optima++ package. To be able to use Chemkin-II, please place the ckinterp executable (or ckinterp.exe on Windows) into the Optima++ package root.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP COMPILE_CKII path/targetMechanismFile.inp
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/CompileCKII.
The name of the output mechanism file will be assembled from the name of the input file (without extension) followed by .bin as an extension. E.g. targetMechanismFile.bin from targetMechanismFile.inp.

COMPILE_FM

Compile the mechanism into a FlameMaster format binary file. This includes transport properties if they were provided as an input too. It should be noted that the transport properties are not necessary for compiling a FlameMaster mechanism, but if they are not provided, the mechanism will only be usable for 0D simulations.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Processing a single file on unix-type systems without transport properties

```
bin/Release/OptimaPP COMPILE_FM path/targetMechanismFile.inp
```

Processing a single file on unix-type systems with transport properties

```
bin/Release/OptimaPP COMPILE_FM path/targetMechanismFile.inp \
path/transportFile.tran
```

The path of the directory where the compiled mechanism file will be printed is `outputs/commandLineJobs/CompileFM`.
The name of the output mechanism file will be assembled from the name of the input file (without extension) followed by `.pre` as an extension. E.g. `targetMechanismFile.pre` from `targetMechanismFile.inp`.

`COMPILE_OS`

Compile the mechanism into a `OpenSmoke++` format kinetics folder. This includes transport properties if they were provided as an input too. It should be noted that the transport properties are not necessary for compiling a `OpenSmoke++` mechanism, but if they are not provided, the mechanism will only be usable for 0D simulations.

WARNING: Currently it is not explicitly checked if the mechanism compilation was successful.

Processing a single file on unix-type systems without transport properties

```
bin/Release/OptimaPP COMPILE_OS path/targetMechanismFile.inp
```

Processing a single file on unix-type systems with transport properties

```
bin/Release/OptimaPP COMPILE_OS path/targetMechanismFile.inp \
path/transportFile.tran
```

The path of the directory where the compiled mechanism file will be printed is `outputs/commandLineJobs/CompileOS`.
The name of the output mechanism folder will be the name of the input file without extension.

`PRINT_CKII_MECH` with `REMOVE_SPECIES`

Remove one or more species from the mechanism, including all thermodynamic and transport properties and all reactions in which it is a reactant or product, and write the mechanism into a file in `Chemkin-II` format, including all thermodynamic properties.

Species to be removed must be preceded by the `REMOVE_SPECIES` keyword and be separated by whitespace characters.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_CKII_MECH path/targetMechanismFile \
REMOVE_SPECIES C3H8 C3H7 C3H6
```

The path of the directory where the compiled mechanism file will be printed is `outputs/commandLineJobs/PrintCKIIMech`.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag `_reduced_CKII` followed by `.inp` as an extension. E.g. `targetMechanismFile_reduced_CKII.inp`.

PRINT_CKII_MECH with REMOVE_REACTION

Remove one or more reactions from the mechanism, and write the mechanism into a file in Chemkin-II format, including all thermodynamic properties.

Reactions to be removed must be preceded by the REMOVE_REACTION keyword and be separated by whitespace characters. The reaction string must not contain whitespace characters, and each species name must be written out separated by "+" signs (i.e. "H+H" should be used instead of "2H"). The reactants and products must be separated by "=", "=>" or "<=>". The different separators between reactants and products are treated as identical (i.e. "H+O2=>OH+O" is equivalent to "H+O2=OH+O").

The reactants and products can be given in an arbitrary order, and parantheses will be ignored. In case of duplicate reactions all instances will be removed.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_CKII_MECH path/targetMechanismFile \
REMOVE_REACTION H+H+M=H2+M HO2+OH=H2O+O2
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/PrintCKIIMech.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag _reactions_removed_CKII followed by .inp as an extension. E.g. targetMechanismFile_reactions_removed_CKII.inp.

PRINT_CKII_MECH with KEEP_SUBMECH

Specify one or more species to keep in the mechanism, and write the mechanism into a file in Chemkin-II format, including all thermodynamic properties.

Species to be kept must be preceded by the KEEP_SUBMECH keyword and be separated by whitespace characters. All reactions will be removed from the mechanism that contain none of the specified species.

Therefore, it is generally expected that some reactions will remain in the mechanism that have not been specified after KEEP_SUBMECH.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_CKII_MECH path/targetMechanismFile \
KEEP_SUBMECH H O OH
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/PrintCKIIMech.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag _submech_CKII followed by .inp as an extension. E.g. targetMechanismFile_submech_CKII.inp.

PRINT_CKII_MECH with SWITCH_A_UNIT

Change the unit of the A Arrhenius parameters in the mechanism, convert all parameter values appropriately, and write the mechanism into a file in `Chemkin-II` format, including all thermodynamic properties. The valid units are those that are available in `Chemkin-II`. In all cases the volume units are cm^3 and the time units are seconds.

NOTE: The exact units of the A Arrhenius parameters always depend on the order of the respective reaction.

The new unit must be preceded by the `SWITCH_A_UNIT` keyword.

Valid units are: `MOLES` ($\mathrm{cm}^3\,\mathrm{mol}\,\mathrm{s}$); `MOLECULES` ($\mathrm{cm}^3\,\mathrm{molecule}\,\mathrm{s}$).

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_CKII_MECH path/targetMechanismFile \
SWITCH_A_UNIT MOLECULES
```

The path of the directory where the compiled mechanism file will be printed is `outputs/commandLineJobs/PrintCKIIMech`.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag `_A_unit_` followed by the new unit and the tag `_CKII` and `.inp` as an extension. E.g. `targetMechanismFile_A_unit_MOLECULES_CKII.inp`.

PRINT_CKII_MECH with SWITCH_E_UNIT

Change the unit of the E Arrhenius parameters in the mechanism, convert all parameter values appropriately, and write the mechanism into a file in `Chemkin-II` format, including all thermodynamic properties.

The new unit must be preceded by the `SWITCH_E_UNIT` keyword.

Valid units are: `KELVINS`; `CAL/MOLE`; `KCAL/MOLE`; `JOULES/MOLE`; `KJOULES/MOLE`.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_CKII_MECH path/targetMechanismFile \
SWITCH_E_UNIT KJOULES/MOLE
```

The path of the directory where the compiled mechanism file will be printed is `outputs/commandLineJobs/PrintCKIIMech`.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag `_E_unit_` followed by the new unit and the tag `_CKII` and `.inp` as an extension. E.g. `targetMechanismFile_E_unit_KJOULESperMOLE_CKII.inp`.

PRINT_FM_MECH with REMOVE_SPECIES

Remove one or more species from the mechanism, including all reactions in which it is a reactant or product and write the mechanism into a file in `FlameMaster` format.

Note that the FlameMaster mechanism file contains only the kinetic information, but no thermodynamic or transport data.

Species to be removed must be preceded by the REMOVE_SPECIES keyword and be separated by whitespace characters.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_FM_MECH path/targetMechanismFile \
REMOVE_SPECIES C3H8 C3H7 C3H6
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/PrintFMMech.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag _reduced_FM followed by .inp as an extension. E.g. targetMechanismFile_reduced_FM.inp.

PRINT_FM_MECH with REMOVE_REACTION

Remove one or more reactions from the mechanism, and write the mechanism into a file in FlameMaster format. Note that the FlameMaster mechanism file contains only the kinetic information, but no thermodynamic or transport data.

Reactions to be removed must be preceded by the REMOVE_REACTION keyword and be separated by whitespace characters. The reaction string must not contain whitespace characters, and each species name must be written out separated by "+" signs (i.e. "H+H" should be used instead of "2H"). The reactants and products must be separated by "=", "=>" or "<=>". The different separators between reactants and products are treated as identical (i.e. "H+O2=>OH+O" is equivalent to "H+O2=OH+O").

The reactants and products can be given in an arbitrary order, and parantheses will be ignored. In case of duplicate reactions all instances will be removed.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_FM_MECH path/targetMechanismFile \
REMOVE_REACTION H+H+M=H2+M HO2+OH=H2O+O2
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/PrintFMMech.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag _reactions_removed_FM followed by .inp as an extension. E.g. targetMechanismFile_reactions_removed_FM.inp.

PRINT_FM_MECH with KEEP_SUBMECH

Specify one or more species to keep in the mechanism, and write the mechanism into a file in FlameMaster format. Note that the FlameMaster mechanism file contains only the kinetic information, but no thermodynamic or transport data.

Species to be kept must be preceded by the KEEP_SUBMECH keyword and be separated by whitespace characters. All reactions will be removed from the mechanism that contain none of the specified species.

Therefore, it is generally expected that some reactions will remain in the mechanism that have not been specified after KEEP_SUBMECH.

Processing a single file on unix-type systems

```
bin/Release/OptimaPP PRINT_FM_MECH path/targetMechanismFile \
KEEP_SUBMECH H O OH
```

The path of the directory where the compiled mechanism file will be printed is outputs/commandLineJobs/PrintFMMech.
The name of the output mechanism file will be assembled from the name of the input file (without extension) and a tag _submech_FM followed by .inp as an extension. E.g. targetMechanismFile_submech_FM.inp.

References

[1] T. Turányi, T. Nagy, I. G. Zsély, M. Cserháti, T. Varga, B. T. Szabó, I. Sedyó, P. T. Kiss, A. Zempléni, H. J. Curran, Int. J. Chem. Kinet., **44**, pp. 284–302 (2012).

[2] T. Varga, C. Olm, T. Nagy, I. G. Zsély, É. Valkó, R. Pálvölgyi, H. J. Curran, T. Turányi, Int. J. Chem. Kinet. (2016), in press.

[3] T. Nagy, T. Turányi, Int. J. Chem. Kinet., **43**, pp. 359–378 (2011).

[4] T. Nagy, É. Valkó, I. Sedyó, I. G. Zsély, M. J. Pilling, T. Turányi, Combust. Flame, **162**, pp. 2059–2076 (2015).

Appendices

Copyright notices

`Optima++` contains third-party software. They are redistributed under their own copyright.

`Optima++` is distributed under the following FreeBSD licence.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

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

`tinyxml2` is governed with the following copyright notice.
Original code by Lee Thomason (www.grinninglizard.com).

This software is provided "as-is", without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

`dos2unix` is governed with the following copyright notice.
The `dos2unix` package is distributed under FreeBSD style license.
See also http://www.freebsd.org/copyright/freebsd-license.html.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice in the documentation and/or other materials provided with the distribution.

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

`Eigen` is primarily MPL2 licensed. See `COPYING.MPL2` and these links:
http://www.mozilla.org/MPL/2.0/
http://www.mozilla.org/MPL/2.0/FAQ.html
Some files contain third-party code under BSD or LGPL licenses. For details see the `COPYING.*` files in the `Eigen` release packaged with `Optima++`.

`TXT_TO_XML` for ReSpecTh version 1.0 files

Format of the input file

`Author:` *string*

The string following the keyword will be used to identify the author of the XML file (which does not imply that the experimental data were obtained by this author).

`Specification version:` *string*

The number following the keyword will be used to specify the ReSpecTh Kinetics Data Format Specification version the created file adheres to. The latest released specification version is 2.3. Two integers separated by a dot will be understood as `major_version.minor_version` and will be inserted into the XML file accordingly.

`Source reference:` *string*

The string following the keyword will be stored as the bibliographic reference of the measurements stored in the XML file.

`Experiment type:` *string*

The string following the keyword is used to define the experiment type.
The allowed types are:

- `Ignition delay measurement`
- `Laminar flame speed measurement`
- `Outlet concentration measurement`
- `Concentration time profile measurement`
- `Jet stirred reactor measurement`
- `Burner stabilized flame speciation measurement`
- `Direct rate coefficient measurement`

`Apparatus:` *string*

The string following the keyword is used to provide the apparatus type in which the experiment was carried out (e.g. shock tube). Providing this information is optional, and will not influence the interpretation of the XML files.

`Operation mode:` *string*

The string following the keyword is used to provide the operating mode of the experimental apparatus in which the experiment was carried out (e.g. reflected shock). Providing this information is optional, and will not influence the interpretation of the XML files.

`Common experimental conditions:` *varied*

The keyword can be followed by multiple lines, each specifying an experimental condition that is common across all individual experiments that are described within the file.
The lines should follow the following format (the order of the sub-keywords is arbitrary):
`Type:`*string* `Species:`*string* `Value:`*number* `Unit:`*string*

The string after "`Type:`" defines what type of physical property is given (e.g. `temperature`, `pressure`).

The string after "`Species:`" defines which species a composition or concentration type of property refers to. For other types of properties it should not be used.

The number after " `Value:`" provides the numeric value of the property, and the string after "`Unit:`" provides the corresponding unit.
Valid property types and corresponding units are summarized in the table at the end of this section.

`Varied experimental conditions and measured results:` *varied*

The keyword can be followed by multiple lines, each specifying an experimental condition that was varied between the individual experiments, or an experimental result.
The lines should follow the following format (the order of the sub-keywords is arbitrary):
`Type:`*string* `Species:`*string* `Unit:`*string* `ID:`*string* `Label:`*string*

The string after "`Type:`" defines what type of physical property is given (e.g. `temperature`, `pressure`).

The string after "`Species:`" defines which species a composition or concentration type of property refers to. For other types of properties it should not be used.

The string after "`Unit:`" defines the unit for the numeric values that will be given later in the file.

The string after "`ID:`" provides an identifier for the property which will be used to identify which values correspond to which property.

The string after "`Label:`" defines a label for the species which is the suggested plot label for the given property. As no plotting is carried out by `Optima++` providing a label is not necessary and the given label is not used (only stored in the XML files).

`Varied values:`

A header line must follow the keyword, which must contain the `ID`s that were defined for the varied properties. The order of the `ID` strings defines the order in which the numerical values are provided for the varied properties. Common `ID` names are e.g. "`x1`, `x2`,...", but any name can be chosen.
The header line must be followed by lines with as many numerical values as many `ID`s were provided in the header, and in the corresponding order. Each line defines an experimental data point.

An example is provided below:

```
Varied values:
x1       x2
25.64    0.0466
37.34    0.0529
45.11    0.0591
49.89    0.0653
51.28    0.0713
50.00    0.0773
42.45    0.0832
33.30    0.0891
22.77    0.0948
```

`Ignition definition:`

The definition of the ignition delay can be specified with this keyword.
The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

`MeasuredQuantity:`*`string`* `Type:`*`string`* `Value:`*`number`* `Unit:`*`string`*

The string following "`MeasuredQuantity:`" defines what physical property was used to define the ignition delay. This can be pressure, temperature or the concentration of a species. These are to be denoted by "`p`", "`T`", and the name of the species (e.g. "`OH`"), respectively.

The string following type "`Type:`" defines which feature of the measured physical property is considered for the ignition delay. Valid values are summarized below:

- `max`
- `d/dt max`
- `baseline max intercept from d/dt`
- `baseline min intercept from d/dt`
- `concentration`
- `relative concentration`

See the RKD Format Specification for a detailed description of the types.

The number following "`Value:`" defines the absolute or relative concentration value the target species has to reach for an ignition to occur. This attribute can only be used when the value of type is "`concentration`" or "`relative concentration`". The string following "`Unit:`" defines the corresponding unit.

`Time shift definition:`

The time shifting procedure can be specified with this keyword for a concentration time profile measurement.

The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

`MeasuredQuantity:`*string* `Type:`*string* `Value:`*number*

The string following "`MeasuredQuantity:`" defines what physical property was used to define the time shift. This must be the name of a single measured species.

The string following type "`Type:`" defines which part of the specified species profile is matched with the experiments. The valid types are summarized below

- `half`
- `inflexion`
- `relative`

See the RKDFS manual for a detailed description of the types.

The number following "`Value:`" defines the absolute or relative concentration value the target species has to reach for ignition to occur. This attribute can only be used when the value of type is "`concentration`" or "`relative concentration`". The string following "`Unit:`" defines the corresponding unit.

`Volume-time profile:`

A volume-time history can be defined with this keyword for an ignition delay experiment (usually an RCM experiment).

The keyword must be followed by lines defining a time and a volume property, in an identical way to property lines after "`Varied experimental conditions and measured results:`".

These lines must be followed by "`Profile:`", after which a header line must appear with the IDs defined here. This must be followed by the numeric values for the time–volume pairs.

An example can be seen below:

```
Volume-time profile:
Type: time      Unit: s          ID: x4
Type: volume    Unit: cm3        ID: x5
Profile:
x4              x5
0.000000e+000   1.000000e+000
1.000000e-006   9.998782e-001
7.564500e-004   9.886456e-001
1.004150e-003   9.831007e-001
```

`Reaction string:`

For direct rate coefficient determinations this keyword must be used to specify the reaction of which the rate coefficient is described.

The keyword must be followed by a line with the following format (the order of the sub-keywords is arbitrary):

`Reaction string:`*string* `Order:`*integer* `Bulkgas:`*string*

The string following "`Reaction string:`" is the reaction string. The names of species on the same side of the reaction must be separated by "`+`" characters, and the reactant and product sides must be separated by an "`=`" sign, which can be bordered by "`>`" or "`<`" characters.
The reaction string must contain "`LP`" or "`HP`" before the first species separated by a space, if the rate coefficients measured are at the low pressure or high pressure limit respectively, for pressure dependent reaction rate coefficients (e.g. "`LP H+O2+M=HO2+M`").

The integer following "`Order:`" defines the order of the reaction. This should match with the reaction string, including high/low pressure limit specification. For fall-off reactions use the lower order (i.e. that which corresponds to the high-pressure limit).

The string following "`Bulkgas:`" specifies the major diluent gas for the experiment/theoretical determination. This only has significance for pressure-dependent rate coefficients and low-pressure limit rate coefficients. When calculating the rate coefficient the bulkgas will be taken into account through third-body collision efficiency effects, as if the whole gas composition was made up by the bulkgas.

NOTE: It is also possible to define a detailed composition through the common or varied conditions, and in this case the bulkgas will be completely ignored.

Summary of valid units

When printing XML files from text files, `Optima++` does not check explicitly is the specified units are valid within the RKDFS. The given units are printed into the XML files and if incorrect or unhandled units are given, then any solver input files printed from XML will not be correct or complete (a warning will be printed in such a case). Therefore it is important to use only those units that are handled within the RKDFS.

In the following table a summary of the unit strings that are currently handled within the RKDFS is given. Here all strings are given in the *exact way* as it should appear in the file. This means that exponents are not typed as superscript, and the micro (μ) prefix should be typed as "`u`" to guarantee that these can be typed in plain text files.

Property type	Valid units
`temperature`	`K`
`pressure`	`Pa, kPa, MPa, Torr, torr, bar, mbar, atm`
`volume`	`m3, dm3, cm3, mm3, L`
`time`	`s, ms, us, ns, min`
`residence time`	`s, ms, us, ns, min`
`distance`	`m, dm, cm, mm`
`ignition delay`	`s, ms, us, ns, min`
`length`	`m, dm, cm, mm`
`density`	`g m-3, g dm-3, g cm-3, g mm-3,` `kg m-3, kg dm-3, kg cm-3, kg mm-3`
`flow rate`	`g m-2 s-1, g dm-2 s-1, g cm-2 s-1, g mm-2 s-1,` `kg m-2 s-1, kg dm-2 s-1, kg cm-2 s-1, kg mm-2 s-1`
`flame speed`	`m/s, dm/s, cm/s, mm/s, m s-1, dm s-1, cm s-1, mm s-1`
`composition`	`mole fraction, percent, ppm, ppb`

`concentration`	`mol/m3, mol/dm3, mol/cm3, mol m-3, mol dm-3, mol cm-3,`
	`molecule/m3, molecule/dm3, molecule/cm3, molecule m-3,`
	`molecule dm-3, molecule cm-3`

`rate coefficient`	`s-1,`
	`m3 mol-1 s-1, dm3 mol-1 s-1, cm3 mol-1 s-1,`
	`m3 molecule-1 s-1, dm3 molecule-1 s-1, cm3 molecule-1 s-1,`
	`m6 mol-3 s-1, dm6 mol-2 s-1, cm6 mol-2 s-1,`
	`m6 molecule-2 s-1, dm6 molecule-2 s-1, cm6 molecule-2 s-1`