MEDUSA

—

Reference Guide to the Configuration and Code Generation Tool MEDUSACOCOGEN

(for SVN revision 388*ff* of MEDUSA)

Guy Munhoven
Université de Liège, Belgium
http://www.astro.ulg.ac.be/~munhoven

30th August 2018

Contents

1 General Overview

The MEDUSA COde COnfiguration and GENeration tool MEDUSACOCOGEN is used to produce critical parts of the Fortran 95 source code of MEDUSA. The standard MEDUSA source code contains only the common framework, for mixing and transporting the constituents. The actual composition of the solid and solute phases, together with the chemical and physical properties of the species (solubility, diffusion coefficients, etc.), the processes that affect them (chemical reactions, kinetic rate laws) and the thermodynamic equilibria that may affect subsets needs to be integrated into MEDUSA's differential-algebraic equations framework used to solve the underlying system of advection-diffusion-reaction equations.

MEDUSACOCOGEN itself is also written in Fortran 95. The selection of constituents, processes and equilibria is based upon XML definition files. To process the XML code of the definition files, MEDUSACOCOGEN calls upon the Fortran 95 μXML library, developed especially for this purpose, but independent and usable for more general usage. The XML-processing capacities of the μXML library are still very limited (hence the μ in the name). Standard conforming comments are correctly recognized and filtered out; syntax and nesting errors are detected. Attribute content can be delimited by simple or by double quotes. `CDATA` sections are recognized and processed (this is required, e.g., for code snippets). Entity processing is, however, not yet supported (as of SVN revision 32 of μXML), not even for the five predefined entities `&quot;` ("), `&apos;` ('), `&` (&), `<` (<) and `>` (>). These five should, however, be introduced in the future. For the common usage of XML, however, not even these are indispensable. Entity processing in XML is unfortunately a recursive process requiring a lot of overhead. Furthermore only the most basic character encoding `us-ascii` is supported. `<!DOCTYPE>` definitions are detected, but ignored. They may nevertheless be included for checking consistency of the prepared XML files with external utilities.

MEDUSACOCOGEN calls upon an extensible library of modules implementing various rate law expressions and chemical equilibrium relationships.

This document describes the formats and element trees of the different XML files required to produce a compilable source file tree of MEDUSA. The code generation is not 100% automatic. For one purpose or the other users have, e.g., to provide relatively low-level support (verbatim code, etc.) that requires some knowledge of naming conventions inside MEDUSA. These internals of MEDUSA are described as well.

In the appendix, we also outline the structure of the module files for the rate laws and the laws of mass action for typical chemical equilibria, and the underlying API (application programming interface) that must be

respected to make the modules usable by MEDUSA and how to make available the information required by MEDUSACOCOGEN to produce code that can be reliably integrated into the MEDUSA source file tree.

The source code of MEDUSACOCGEN is located in the `src-mcg` directory of the tree retrieved from the MEDUSA SVN repository. Please refer to the companion "*Installation and Compilation Guide*" for detailed informations about system requirements and about the procedure for downloading and installing the required source files.

2 Main Building List

MEDUSACOCOGEN reads the list of definition files for the components to include, and the processes and equilibria to consider in an XML file called `list.xml`. The root element of that list is `<MedusaCoCoGen>`. Its recognized child elements are as follows.

- `<Composition>` lists the components (species) and systems to include: each component to include is referred to by one of three empty elements
  - `<Solid file="`*`solid.xml`*`" order="`*`nn`*`" />`
  - `<Solute file="`*`solute.xml`*`" order="`*`nn`*`" />`
  - `<SoluteSystem file="`*`solsys.xml`*`" order="`*`nn`*`" />`

  The `file` attribute is mandatory for all of these and gives the pathname of the respective definition file, relative to the execution directory of MEDUSACOCOGEN. The `order` attribute is optional and its value ***nn*** must be integer. It allows to set the order by which the species are assembled in MEDUSA. The order and interdependency of the species influences on the compactness of the Jacobian of the equation system, and may thus impinge on the computational efficiency of the model. Order numbers need not to be consecutive. For solids and solutes, the order number is taken as is; for solute systems, which group solutes, the given order number is used for the system itself, and as an offset for the actual components, to be resolved according to the definition in the given file (see section 3.3 below). If the `order` attribute is omitted, the species and systems are ordered by order of appearance in `list.xml`; if one order number appears more than once, they are treated on a *first-read–first-processed* basis.

- `<Processes>` lists the process definition files for the processes to consider, each one referred to by an empty element of the form

```
<Process file="process.xml" />
```

where the mandatory `file` attribute gives the pathname of the definition file for that process, relative to the directory where MEDUSACOCOGEN executes from.

- `<Equilibria>` lists the definition files for the chemical equilibria to consider, each one referred to by an empty element

```
<Equilibrium file="process.xml" />
```

where the mandatory `file` attribute gives the pathname of the definition file for that equilibrium, similarly to `<Process>` above.

Each one of the `<Composition>`, `<Processes>` and `<Equilibria>` elements may appear once at most; they may be omitted or left empty if adequate. Duplicate file names are detected and processed only once.

Below follows the `list.xml` that is used to configure MEDUSA for use with MBM. In this version, MEDUSA considers a sediment with four different solid components, one solute system (which includes three solute species—its definition file is reproduced and detailed as an example in section 3.3 below) plus two individual solute species, three processes and one chemical equilibrium. In this example, taken from the standard distribution of MEDUSA, all of the XML files are located in a `xml` subdirectory in the directory where the MEDUSACOCOGEN executable `medusa_cocogen_xml` is run from (normally `src-mcg`).

```
1  <?xml version="1.0"?>
2
3  <!-- List of XML files for components required for the MEDMBM
       configuration -->
4
5  <MedusaCoCoGen>
6
7     <Composition>
8        <Solid        file="xml/clay.xml" order="1" />
9        <Solid        file="xml/calc.xml" order="2" />
10       <Solid        file="xml/arag.xml" order="3" />
11       <SoluteSystem file="xml/dic.xml"  order="4" />
12       <Solid        file="xml/orgm.xml" order="8" />
13       <Solute       file="xml/o2.xml"   order="9" />
14       <Solute       file="xml/ca.xml"   order="10" />
15    </Composition>
```

```
16
17    <Processes>
18       <Process      file="xml/proc_arag_diss.xml" />
19       <Process      file="xml/proc_calc_diss.xml" />
20       <Process      file="xml/proc_orgm_oxic.xml" />
21    </Processes>
22
23    <Equilibria>
24       <Equilibrium  file="xml/equi_dic.xml" />
25    </Equilibria>
26
27 </MedusaCoCoGen>
```

3 Fundamental Component and Species Definition File Formats

The formats of the different definition files will now be presented. For each kind, we will outline the main features in a descriptive text, and then consider one or two examples to precise and illustrate the details.

3.1 Solutes

Definition files for solutes must provide all the information that MEDUSA requires about a solute: names, alkalinity content, code snippets to calculate diffusion coefficients or to evaluate properties from parametrizations, and possibly conservation properties for setting up mass balances. "Solutes" are, however, not limited to dissolved substances in this context, but encompass sediment properties in the *solute phase*, including isotopic signatures colour tracers.

The root element of a solute's definition file is `<Solute>`. It has generally only one attribute, `type`, which defines how the evolution of the solute property should be dealt with in MEDUSA:

- with `type="normal"`, the usual partial differential equations are used to control the evolution of the solute's concentration; `<CodeBits>` (see below) must include a `<DiffCoeff>` element with the code snippet for calculating the diffusion coefficient
- with `type="parameterized"`, the solute's concentration is not meant to be controlled by a partial differential equation, but calculated directly from a parametrization; in this case, `<CodeBits>` must include a

`<TotalConcentration>` element with the code snippet to perform that calculation.

- with `type="ignored"` (default), the component is simply disregarded by MEDUSACOCOGEN.

For more advanced usage, the `<Solute>` root element may also include a `class` attribute (see section 3.4.2 below) which can, however, most often be omitted. If omitted, ts value defaults to `BasicSolute`.

Further characteristics of the general layout of a solute's XML definition file will be exposed with a concrete definition file as a support. Here we have chosen the definition file of bicarbonate, `hco3.xml`, which can be found in the `src-mcg/xml` directory:

```
1  <?xml version="1.0"?>
2
3  <Solute type="normal">
4
5     <Names>
6        <Generic>HCO3</Generic>
7        <Long>Bicarbonate Ion</Long>
8        <ShortID>hco3</ShortID>
9     </Names>
10
```

The `<Names>` element must be present exactly once and it must include three child elements:

- `<Generic>` gives the *generic name* of the component. The generic name is used to refer to the component in rate laws and laws of mass-action in other XML definition files and also as a *variable name* in the NETCDF files generated by MEDUSA. The generic name must not contain any spaces; it may contain underscore characters though. Its length is limited to 20 characters (set by the `PARAMETER n_lmaxnamesgen` in `MOD_MEDUSA_COCOGEN`).

- `<Long>` gives the *long name* of the component used, e.g., as the long name attribute for the variables in the NETCDF files. Its length is limited to 30 characters (set by the `PARAMETER n_lmaxnameslong` in `MOD_MEDUSA_COCOGEN`).

- `<ShortID>` gives a short identificator that is used to construct the variable names for indices or special properties of the solute in the MEDUSA source code. It must obviously not contain any spaces. Its length

is limited to 10 characters (set by the `PARAMETER n_lmaxshortid` in `MOD_MEDUSA_COCOGEN`).

```
11
12    <CodeBits>
13       <DiffCoeff>
14          <Fortran requires="wtmpdc">
15 <![CDATA[
16       ! D_HCO3 : from Boudreau (1997, Table 4.8, in cm2/s)
17       {varname} =
18      &    (5.06D-6 + 0.275D-6*wtmpdc)*dp_cm2_p_sec
19 ]]>
20          </Fortran>
21       </DiffCoeff>
22    </CodeBits>
23
```

The `<CodeBits>` element includes code snippets for various purposes. Normal solutes must include a `<CodeBits>` element with a `<DiffCoeff>` child-element, providing the code basis for calculating the solute component's diffusion coefficient. One may of course envisage future developments and transpositions of MEDUSACOCOGEN to produce MEDUSA instances in different programming languages. The code snippets may therefore theoretically have to be included for any programming language. Relevant code snippets are thus encapsulated in a child element of `<DiffCoeff>` referring to the name of the programming language. As MEDUSA currently only exists in Fortran, MEDUSACOCOGEN also only takes into account the `<Fortran>` tag. The actual code is quoted as character data (non-parsable) within `<![CDATA[ ... ]]>`. MEDUSACOCOGEN receives such code "as is" from the XML parser and inserts it into the final Fortran 95 source code file `mdiffc.F`, after having replaced the `{varname}` placeholder will be replaced by the actual name or array element reference for the diffusion coefficient of the component. White space, blank lines and comments are preserved. Other components may include, as illustrated in the "parameterized" example component below. Please notice that the pre-requisites that can be listed via the `requires` attribute are currently not utilized.

```
24
25    <Alkalinity units="eq/mol">1</Alkalinity>
26
```

The `<Alkalinity>` tag can be used to indicate the alkalinity carried by each mol of the component. The content can be given either numerically or in

terms of internal MEDUSA variables (see below). This alkalinity content is used for the mass balance diagnostics (see below), if required.

The `<ConservationProperties>` (the `units` attribute is only given for documentation purposes, but is otherwise ignored) is used to set up the mass balance equations

```
27
28    <ConservationProperties units="mol/mol">
29       <C>  1</C>
30    </ConservationProperties>
31
32 </Solute>
```

Calcium is typically treated as a parametrized component in seawater: its concentration is conservative and can be expressed as a function of salinity, which is one of the boundary conditions that a host model needs to provide. For simplicity, we also consider it to be conservative in the surface sediments.

```
1  <?xml version="1.0"?>
2
3  <Solute type="parameterized">
4
5     <Names>
6        <Generic>Ca</Generic>
7        <Long>Calcium</Long>
8        <ShortID>ca</ShortID>
9     </Names>
10
11    <CodeBits>
12       <TotalConcentration units="mol␣m-3␣seawater">
13          <Fortran requireslibrary="libthdyct"
14                   requires="wtmpk,␣wsalin,␣wdbsl,␣rho">
15 <![CDATA[
16       {varname} = TOTCA(wtmpk, wsalin, wdbsl) * rho
17 ]]>
18          </Fortran>
19       </TotalConcentration>
20    </CodeBits>
21
22 </Solute>
```

3.2 Solids

Definition files for solids must provide all the information that MEDUSA requires about a solid: names, alkalinity content, physical properties such as its density or its molar mass, chemical composition, code snippets to calculate properties such as saturation concentrations or solubility products, and possibly conservation properties for setting up mass balances. Similarly to solutes, "Solids" are, once again, not limited to actual substances in this context, but encompass all properties in the *solid phase* of the sediment, including isotopic signatures, colour tracers and immaterial properties, such as age or time of formation.

The root element of a solid's definition file is `<Solid>`. It has generally only one attribute, `type`, which defines how the evolution of the solid is to be controlled in MEDUSA:

- with `type="normal"`, the usual partial differential equations are used to control the evolution of the solid's concentration;
- with `type="ignored"` (default), the component will be disregarded by MEDUSACOCOGEN.

For more advanced usage the `<Solid>` may further include a `class` or a `master` attribute (see sections 3.4.2 and 5.1 below). The `class` attribute can nevertheless generally be omitted, in which case its content defaults to `BasicSolid`; the `master` attribute is intended for special purposes. At this stage, there is *no* `parameterized` type of solids.

Further characteristics of the general layout of a solid's XML definition file will again be exposed with a concrete definition file as a support. Here we chose the definition file of opal, which can be found in `opal.xml` in the `src-mcg/xml` directory:

```
1  <?xml version="1.0"?>
2
3  <Solid type="normal">
4
5     <Names>
6        <Generic>Opal</Generic>
7        <Long>Opal</Long>
8        <ShortID>opal</ShortID>
9     </Names>
10
```

The `<Names>` element must fulfil the same requirements for solids as for solutes (see p. 7).

```
11    <PhysicalProperties>
12       <Density units="kg/m3">2100</Density>
13       <MolWeight units="kg/mol">0.0600843</MolWeight>
14    </PhysicalProperties>
15
```

The <PhysicalProperties> element may have two child elements.

- <Density> provides the density [kg/m^3] of the component;
- <MolWeight> provides a first estimate of the molar mass of the component. This is re-calculated for consistency reasons from the chemical composition, when available from <ChemicalComposition> (next).

```
16    <ChemicalComposition>
17       <Si>1</Si>
18       <O >2</O>
19    </ChemicalComposition>
20
```

<ChemicalComposition> is optional. If present, it should have as many child elements as there are chemical elements that constitute the component being defined. Each child element tag of <ChemicalComposition> is equal to the chemical symbol of a chemical element included and its content is equal to the number of atoms of that element per component molecule (here SiO_2 requires thus two child elements: <Si> with the content 1 and <O> with the content 2).

```
21    <CodeBits>
22       <SaturationConc units="mol/m^3">
23          <Fortran>
24 <![CDATA[
25 !      ! Faure (1991) Inorganic Geochemistry,
26 !      ! Eqn (11.93), page 201:
27 !      ! [H4SiO4] at equilibrium with
28 !      ! amorphous silica = 10^{-2.4} mol/litre
29 !      {varname} = 10.0D+00**(-2.4D+00)*1.0D+03
30
31       ! Hurd (1973) GCA37:2257-2282,
32       ! cited by Archer et al (1993):
33       ! solubility of opal at 4 degC = 1000 uM
34       {varname} =  1.0D+00
35 ]]>
```

```
36          </Fortran>
37       </SaturationConc>
38    </CodeBits>
39
40
41    <Alkalinity units="eq/mol">0</Alkalinity>
42
43
44    <ConservationProperties units="mol/mol">
45       <Si>  1</Si>
46    </ConservationProperties>
47
48 </Solid>
```

For numerical stability reasons, it is mandatory to include one solid component that is inert and has the attribute `type="normal"`. Its root element must have the attribute `extra="mud"`. If several components happen to have this `extra` attribute, the first of these that MEDUSACOCOGEN registers takes this special role.

3.3 Solute Systems

The root element for a solute system is `<SoluteSystem>`. Similarly to the `<Solute>` root element in the definition files of normal solutes, it can have a `type` attribute. The `class` attribute may be used to further consider the kind of system we have (e.g., acid-base system), but it is currently only included for information purposes and not taken into account by MEDUSACOCOGEN during code generation. `<SoluteSystem>` has two mandatory child elements:

- `<Names>` must provide the three different names for the solute system (as for solutes and solids);
- `<Composition>` lists the files of the individual solutes that make up the system, similarly to the `<Composition>` in the main `list.xml`.

The definition file for Dissolved Inorganic Carbon (`dic.xml`) is reproduced below:

```
1 <?xml version="1.0"?>
2
3 <SoluteSystem type="normal" class="acid-base">
4
5    <Names>
```

```
6         <Generic>DIC</Generic>
7         <Long>Dissolved Inorganic Carbon</Long>
8         <ShortID>dic</ShortID>
9      </Names>
10
11     <Composition>
12        <Solute file="xml/co2.xml"  disso="0" order="3" />
13        <Solute file="xml/hco3.xml" disso="1" order="2" />
14        <Solute file="xml/co3.xml"  disso="2" order="1"/>
15     </Composition>
16
17  </SoluteSystem>
```

Please notice that the attribute `disso`, which gives the dissociation level with respect to the fundamental acid in the acid-base system, is currently only reserved, but not processed. The final order of the individual species that will be used for MEDUSA's internal arrays is obtained by adding the value of the `order` attribute in the solute system definition file to that of the solute system itself given in the main composition list `list.xml`.

3.4 Advanced Topics for Components and Species

3.4.1 Material characteristics names

Some inside knowledge about how the names of variables holding quantitative material properties are built in MEDUSA is indispensable for an efficient usage of MEDUSACOCOGEN. The most commonly required such variables are the molar compositions of components, which are required, e.g., in the stoichiometric definition of reactions and also to set up mass balance conservation equations.

Whenever a component has a `<ChemicalComposition>` tag in its definition file, one variable is declared in `MOD_MATERIALCHARAS` for each entry in that `<ChemicalComposition>`, to hold the molar content in the given chemical element. The name of each such molar composition variable is obtained by appending the chemical symbol of the element to the short ID of the component, as given by the `<ShortID>` child of the `<Names>` element, separated by an underscore (_). The variables are initialized to the content of the elements, which must be numeric and will be converted to `DOUBLE PRECISION`. The chemical composition data included in the `opal.xml` file above thus lead to the declarations

```
DOUBLE PRECISION, SAVE :: opal_si = 1.0D+00
DOUBLE PRECISION, SAVE :: opal_o = 2.0D+00
```

Please notice that tag names in the XML files are case-sensitive while Fortran 95 variable names are not.

These variables are provided for possible use in MEDUSA. Whether they are actually used cannot be predicted in all generality. They *are* used for organic matter components, e. g., for which it is mandatory to include the `<ChemicalComposition>` in the definition file (see section 3.4.2 below).

Whenever the chemical composition is provided in the definition file, code is inserted into the subroutine `SOLIDS_STOECHIOMETRY` in `MOD_MATERIALCHARAS` to recalculate the molar mass of the component, for optimal mass conservation. If you plan to include components with exotic chemical compositions, please check in `mod_basicdata_medusa.F` whether the required atomic masses are already provided and if not, please complete the missing information therein by following the given obvious naming systematics. `mod_basicdata_medusa.F` currently only includes data for the most common chemical elements.

The variable name for the molar mass of a component is obtained from its short ID prefixed by '`mol_`'. It is generated for every solid, and declared in `MOD_MATERIALCHARAS`. Again, for the `opal.xml`, the following declaration is included in `MOD_MATERIALCHARAS`:

```
 DOUBLE PRECISION, SAVE
&  :: mol_opal = 6.00843000000000002E-002
```

The initial value at declaration is taken from the `<MolWeight>` entry of the `<PhysicalProperties>` element (the difference with the given value of 0.0600843 stems from the conversion to `DOUBLE PRECISION`).

The name of the variable carrying a solid's density is obtained by prefixing the short ID of the solid with '`rho_`'. In the `opal.xml` example above, this leads to

```
 DOUBLE PRECISION, PARAMETER , PRIVATE
&  :: rho_opal = 2100
```

3.4.2 Organic matter class

Organic matter, be it in solid or solute phase, requires additional information regarding their composition. Organic matter solids and solute components must therefore be marked with a special `class` attribute in the root element of their definition file (`<Solid>` or `<Solute>`) which must be set to `OrgMatter_CNP`. In addition, a `<ChemicalComposition>` child element is mandatory, even for solutes of this class, that either defines

- the simplified stoichiometric composition in terms of C, N and P only, by including only the three tags `<C>`, `<N>` and `<P>` – the additional characteristics are then derived from these three by assuming that organic matter has the chemical "formula" $(CH_2O)_c(NH_3)_n(H_3PO_4)_p$, c, n and p being the values to be given by `<C>`, `<N>` and `<P>`, respectively (106, 16 and 1 resp. for the classical Redfield composition);

- or the general stoichiometric composition in terms of C, N, P, O and H, by including the five tags `<C>`, `<N>`, `<P>`, `<O>` and `<H>`, in which case organic matter is assumed to have the chemical "formula" $C_cN_nP_pO_oH_h$, c, n, p, o and h being the values of the respective tag analogues.

In the first case, the variables holding the chemical composition in O and H are also declared and initialized according to the simplified composition.

In both cases an additional variable which holds the number of moles of O_2 released for each mole of organic matter produced, is declared in `MOD_MATERIAL_CHARAS` and initialized on the basis of the available data in the chemical constituents. Its name is obtained by appending the suffix '`_remin_o2`' to the short ID of the organic matter component. The same amount of O_2 will of course be consumed during oxic remineralization of the organic component.

Particulate Organic Matter Below, we reproduce `orgm.xml`, the definition file for a particulate organic matter component (`orgm.xml` can be found in `src-mcg/xml`):

```
<?xml version="1.0"?>

<Solid type="normal" class="OrgMatter_CNP">

   <Names>
      <Generic>OrgMatter</Generic>
      <Long>Organic Matter</Long>
      <ShortID>om</ShortID>
   </Names>

   <PhysicalProperties>
      <Density units="kg/m3">1450</Density>
      <MolWeight units="kg/mol">3.55326</MolWeight>
   </PhysicalProperties>

   <ChemicalComposition>
```

```
17        <C>106</C>
18        <N> 16</N>
19        <P>  1</P>
20     </ChemicalComposition>
21
22
23     <Alkalinity units="eq/mol">(-om_n-om_p)</Alkalinity>
24
```

This example illustrates how the MEDUSA specific names of the variables that carry the material characteristic data for organic matter are used: each mole of organic matter carries as much alkalinity as they contain N and P, and it carries it in negative form. As explained in section 3.4.1 the N content of an organic matter component, in moles of N per mole of organic matter, is given by the variable `om_n` for an organic matter component with a short ID `om` (from line 8 above). Similarly, for P the respective variable is `om_p`.

```
25
26     <ConservationProperties units="mol/mol">
27        <C>  om_c</C>
28        <O2> -om_remin_o2</O2>
29     </ConservationProperties>
30
31 </Solid>
```

Dissolved Organic Matter Organic matter can also be considered in the fluid phase, as dissolved organic matter (DOM). DOM components are special cases of solutes. Their definition files must include all the elements required for solutes in general (`<Names>`, `<CodeBits>`, `<Alkalinity>`, ...) plus the extra ones for `class="OrgMatter_CNP"` (e.g., `<ChemicalComposition>`):

```
1  <?xml version="1.0"?>
2
3  <Solute type="normal" class="OrgMatter_CNP">
4
5     <Names>
6        <Generic>DissOrgMatter</Generic>
7        <Long>Dissolved Organic Matter</Long>
8        <ShortID>dom</ShortID>
9     </Names>
10
11
```

```
12    <ChemicalComposition>
13       <C>106</C>
14       <N> 16</N>
15       <P>  1</P>
16    </ChemicalComposition>
17
18
19    <CodeBits>
20       <DiffCoeff>
21          <Fortran>
22 <![CDATA[
23       ! D_DOM : approximate average value for estuarine
24       ! natural DOM samples from Balch and Gueguen (2015)
25       ! Environmental Chemistry 12(2):253-260,
26       ! DOI:10.1071/EN14182
27       ! Reported range: 2.42E-6 to 10.7E-6 cm^2 s^-1
28       {varname} = 6.56D-6*dp_cm2_p_sec
29 ]]>
30          </Fortran>
31       </DiffCoeff>
32    </CodeBits>
33
34
35    <Alkalinity units="eq/mol">(-dom_n-dom_p)</Alkalinity>
36
37
38    <ConservationProperties units="mol/mol">
39       <C>  dom_c</C>
40       <O2> -dom_remin_o2</O2>
41    </ConservationProperties>
42
43 </Solute>
```

3.4.3 Data inheritance

Some components may be intimately linked to each other and share some common characteristics; the same may be true for some processes (see section 4.2.2 below). To call upon of such data inheritance into account, the root element (`<Solute>` or `<Solid>`) may include a `master="`*`OtherCompo`*`"` attribute, where *`OtherCompo`* is the generic name of the component from which any missing data should be taken. The description file of *`OtherCompo`*

must have been read in before the current component's data chain is being initialised. This is a limitation to bear in mind: in order to avoid having to implement complex circular referencing detection schemes in MEDUSACOCOGEN, it is expected that the user itself controls the order by which the list the definition files in the lists.

A typical example of a component that uses data inheritance is C^{13}-OrgMatter , which is a solid component of `class SolidColour`. For consistency, it must have the same C:N:P:O:H composition as bulk organic matter: the chemical composition is therefore inherited from the XML-tree of the component with the generic name `OrgMatter` by including the attribute `master` in the `<Solid>` root element and setting it to `"OrgMatter"`, and by providing only the specific data for C^{13}-OrgMatter, as illustrated in the description file `orgm_c13.xml` from `src-mcg/xml`:

```
<?xml version="1.0"?>

<Solid type="normal" class="SolidColour" master="OrgMatter">

   <Names>
      <Generic>OrgMatter_C13</Generic>
      <Long>Organic Matter C13</Long>
      <ShortID>om_c13</ShortID>
   </Names>


   <PhysicalProperties/>


   <ConservationProperties/>

</Solid>
```

The omitted `<ChemicalComposition>` data will be taken from `OrgMatter`.

4 Processes and Equilibria

4.1 The `<ChemicalReaction>` XML Element

We call upon the same `<ChemicalReaction>` element to describe chemical reactions such as

$$n_{\mathrm{A}}\mathrm{A} + n_{\mathrm{B}}\mathrm{B} + n_{\mathrm{C}}\mathrm{C} \longrightarrow n_{\mathrm{D}}\mathrm{D} + n_{\mathrm{E}}\mathrm{E}$$

in process and in equilibrium definition files alike. We therefore present this element first.

The chemical reagents on the left-hand side will be called the *reactants*, those on the right-hand side the *products*. The `<ChemicalReaction>` element has been designed to represent the stoichiometry of such reactions in an XML format. A `<ChemicalReaction>` element has as many child elements as there are reagents. The child elements come as two types: `<Reactant>` and `<Product>`. Both have the following common structure:

```
<Reactant id="r1">
   <Name>GenericNameofA</Name>
   <StoechCoeff>n_A</StoechCoeff>
</Reactant>

<Product id="p1">
   <Name>GenericNameofD</Name>
   <StoechCoeff>n_D</StoechCoeff>
</Product>
```

Each reagent, reactant or product, is identified in `<Name>` by its generic name. The stoichiometric coefficient of each reagent is given in the `<StoechCoeff>`, either numerically, or by calling upon declared variables from MEDUSA, more particularly those from the `MOD_MATERIALCHARAS` module.

When MEDUSACOCOGEN parses a `<ChemicalReaction>` element, it scans the list of components already registered. The `<Composition>` element of `list.xml` is processed before the `<Processes>` and `<Equilibria>` elements so that all the components modelled or parametrized are already known. Components that are not yet listed are added with the `type="ignored"`.

The attribute `id` that can be used with both element types provides a means to refer to the different reagents by a freely chosen ID.[1] One of these IDs will be used, e.g., in the rate laws to specify the reagent with respect to which the reaction rate is expressed. Normally these IDs should be unique within one single `<ChemicalReaction>` element. This is however currently not enforced by MEDUSACOCOGEN, and left to the responsibility of the users. Since each process or chemical equilibrium definition file must have one and only one `<ChemicalReaction>`, the IDs are also unique within each definition file and referring to one of the reagents by their ID does not lead to any ambiguities.

In addition to `id`, any of the elements can have a `wildcard` attribute, whose value will be a freely chosen character string that will be searched for

[1] I usually name the reactants `r1`, `r2`, ... and the products `p1`, `p2`, ...

in the contents of all the `<StoechCoeff>` children of all the `<Reactant>` and `<Product>` elements and all of its occurrences will be replaced by the short ID of the component referenced in the `<Name>` child of the respective reagent. An application of this feature is illustrated below.

The complete symbolic reaction above is encoded as

```
<ChemicalReaction>

   <Reactant id="r1">
      <Name>GenericNameofA</Name>
      <StoechCoeff>n_A</StoechCoeff>
   </Reactant>

   <Reactant id="r2">
      <Name>GenericNameofB</Name>
      <StoechCoeff>n_B</StoechCoeff>
   </Reactant>

   <Reactant id="r3">
      <Name>GenericNameofC</Name>
      <StoechCoeff>n_C</StoechCoeff>
   </Reactant>


   <Product id="p1">
      <Name>GenericNameofD</Name>
      <StoechCoeff>n_D</StoechCoeff>
   </Product>

   <Product id="p2">
      <Name>GenericNameofE</Name>
      <StoechCoeff>n_E</StoechCoeff>
   </Product>

<ChemicalReaction>
```

4.2 Processes

4.2.1 Process Definition File Format

The root element for a *process definition file* is `<Process>`. It generally has a `realms` attribute to delimit the realms where the process must be

considered: `dbl` – in the diffusive boundary layer; `reaclay` – in the reactive layer.[2] The `realms` attribute takes a comma separated list of the realms where the process should be considered as its value. Processes involving solids must not be considered in the diffusive boundary layer.

The rest of the elements of a process definition file will be exposed below on the basis of the process description file for the oxic degradation of $\mathrm{OrgMatter} = (CH_2O)_c(NH_3)_n(H_3PO_4)_p$

$$\begin{aligned}&\mathrm{OrgMatter} + r\mathrm{O_2}\\ &\quad\longrightarrow\quad c\mathrm{CO_2} + p\mathrm{H_2PO_4^-} + n\mathrm{NO_3^-} + (n+p)\mathrm{H^+} + (c+n)\mathrm{H_2O}\end{aligned}$$

which we rewrite by adding $(n+p)\mathrm{HCO_3^-}$ on both sides in order to eliminate $\mathrm{H^+}$ via the equilibrium $\mathrm{CO_2} + \mathrm{H_2O} \rightleftharpoons \mathrm{HCO_3^-} + \mathrm{H^-}$,

$$\begin{aligned}&\mathrm{OrgMatter} + r\mathrm{O_2} + (n+p)\mathrm{HCO_3^-}\\ &\quad\longrightarrow\quad (c+n+p)\mathrm{CO_2} + p\mathrm{H_2PO_4^-} + n\mathrm{NO_3^-} + (c+2n+p)\mathrm{H_2O}.\end{aligned}$$

This process definition can be found in `proc_orgm_oxic.xml` in the directory `src-mcg/xml`).

```
<?xml version="1.0"?>
<Process realms="reaclay">

   <Names>
      <Generic>OrgMatterOxicDegrad</Generic>
      <Long>Orgm oxic degradation</Long>
      <ShortID>oxicd</ShortID>
   </Names>

```

`<Process>` has a `<Names>` child element just like `<Solute>`, `<Solid>` and `<SoluteSystem>` presented before.

```

   <ChemicalReaction>

      <Reactant id="r1" wildcard="{om}">
         <Name>OrgMatter</Name>
         <StoechCoeff>1</StoechCoeff>
      </Reactant>

```

[2] Additionally, `tranlay` and `corelay` are reserved realm names for future developments, in case reactions could also be allowed to take place in the transition layer and the core layers.

The preceding `<Reactant>` has a `wildcard` attribute: the contents of all the `<StoechCoeff>` sub-children of the current `<ChemicalReaction>` will be searched for the character string held in that attribute (i.e., `{om}` here) and replace that character string by the short ID of the component referenced in the same element (i.e., `OrgMatter`) as specified in its definition file.

```
18
19        <Reactant id="r2">
20           <Name>O2</Name>
21           <StoechCoeff>{om}_remin_o2</StoechCoeff>
22        </Reactant>
23
24        <Reactant id="r3">
25           <Name>HCO3</Name>
26           <StoechCoeff>({om}_n+{om}_p)</StoechCoeff>
27        </Reactant>
28
29
30        <Product  id="p1">
31           <Name>CO2</Name>
32           <StoechCoeff>({om}_c+{om}_n+{om}_p)</StoechCoeff>
33        </Product>
34
35        <Product  id="p2">
36           <Name>H2PO4</Name>
37           <StoechCoeff>{om}_p</StoechCoeff>
38        </Product>
39
40        <Product  id="p3">
41           <Name>NO3</Name>
42           <StoechCoeff>{om}_n</StoechCoeff>
43        </Product>
44
45        <Product  id="p4">
46           <Name>H2O</Name>
47           <StoechCoeff>({om}_c+{om}_n+{om}_n+{om}_p)</
      StoechCoeff>
48        </Product>
49
50     </ChemicalReaction>
51
```

```
52
53     <RateLaw reference_id="r1" subr="MONOD1_C">
54        <RateConstant    type="globalconstant"/>
55        <HalfSatConstant type="globalconstant"/>
56        <MonodConc>O2</MonodConc>
57        <Proportional>OrgMatter</Proportional>
58     </RateLaw>
59
60
61  </Process>
```

The `reference_id` attribute of the `<RateLaw>` element indicates that the reaction rate to be calculated by the subroutine `MONOD1_C` is for the reactant that has `id="r1"` in the `<ChemicalReaction>` above. Since it is a reactant, the rate will get a negative sign.

The child-elements of `<RateLaw>` are specific for the rate law provided by the subroutine `MONOD1_C`.

A comprehensive list with all the rate laws currently available in the library can be found the section A.3 in the appendix.

4.2.2 Advanced Topics for Processes

Similarly to the `<Solid>` and `<Solute>` root elements in solid and solute description files, the special attribute `master="`*`SomeOtherProcess`*`"` can also be used with the `<Process>` root element. Here, *`SomeOtherProcess`* must be the name of a process previously registered, called the *master process*. If the master process has not yet been registered MEDUSACOCOGEN aborts with an error.

In addition, the `<RateLaw>` element in a process description file may get the attribute `xref="`*`SomeOtherProcess`*`"`, where *`SomeOtherProcess`* must again be the name of a process previously registered. Missing information is then taken from the `<RateLaw>` element in that process's description file. The rate law referred to can be of a different type, as long as it includes tag names that correspond to those that are missing.

Both techniques are used in the description file related to solid's production time concentrations (see section 5.1 below for details about production time concentration of solids).

4.3 Equilibria

4.3.1 Equilibrium Description File

Equilibrium and process definition files are very similar. The only difference is that the `<RateLaw>` XML element in a process definition file is replaced by a `<LawOfMassAction>` XML element in an equilibrium definition file. In addition, the `<LawOfMassAction>` cannot have a `reference_id` attribute, as it would not make sense. The chemical equilibrium is described by a `<ChemicalReaction>` element, similar to what we have seen for processes above. The usage of `<LawOfMassAction>` is analogue to the usage of `<RateLaw>` in process description files.

Below, the equilibrium description file for the carbonate equilibrium (file `equi_dic.xml` from `src-mcg/xml` is reproduced.

```
<?xml version="1.0"?>

<Equilibrium>

   <Names>
      <Generic>EquiDIC</Generic>
      <Long>Carbonate Equilibrium</Long>
      <ShortID>eqdic</ShortID>
   </Names>


   <ChemicalReaction>

      <Reactant id="r1">
         <Name>CO2</Name>
         <StoechCoeff>1</StoechCoeff>
      </Reactant>

      <Reactant id="r2">
         <Name>CO3</Name>
         <StoechCoeff>1</StoechCoeff>
      </Reactant>

      <Reactant id="r3">
         <Name>H2O</Name>
         <StoechCoeff>1</StoechCoeff>
      </Reactant>
```

```
28
29        <Product  id="p1">
30           <Name>HCO3</Name>
31           <StoechCoeff>2</StoechCoeff>
32        </Product>
33
34     </ChemicalReaction>
35
36
37     <LawOfMassAction subr="R1R2P1P1">
38        <Reactant1>CO2</Reactant1>
39        <Reactant2>CO3</Reactant2>
40        <Product1>HCO3</Product1>
41     </LawOfMassAction>
42
43
44  </Equilibrium>
```

A comprehensive list with all the laws of mass-action currently available in the library can be found the section B.3 in the appendix.

5 Immaterial and Volume-less Properties

5.1 Age or Production Time of Solids

The particle age or equivalently, the date or time of production of a solid can be used to construct an "age model" for the synthetic core produced during an simulation experiment.

MEDUSACOCOGEN can produce the required source and sink terms for the corresponding evolution equations for the solid's *production time concentration*. It is therefore necessary to include a component description file for the production time concentration of the solid for which such equations should be generated. This is essentially a solid's definition file, where the `<Solid>` root element has the `class="SolidProductionTime"` attribute, and also the

Below, we reproduce the description file for the production time of calcite (file `calc_pt.xml` from `src-mcg/xml`).

```
1  <?xml version="1.0"?>
2
3  <Solid type="normal" class="SolidProductionTime"
4         master="Calcite">
```

```
5
6     <Names>
7        <Generic>CalcitePT</Generic>
8        <Long>Calcite Produc. Time</Long>
9        <ShortID>calc_pt</ShortID>
10    </Names>
11
12    <PhysicalProperties/>
13
14    <ConservationProperties/>
15
16 </Solid>
```

In addition, for each process that affects the master solid, a corresponding process definition file must be provided to take the effect on Production Time into account. However, since it is rather the transformation of the Production Time during an already defined process that must be considered, the "process" that transforms the Production Time is intimately linked to the characteristics of the physical process itself. Therefore, we must be sure that the rate constants etc. are consistent. Rather than copying them manually into the description file for the Production Time, we use the inheritance feature offered by MEDUSACACOGEN: The root `<Process>` element gets a `master` attribute, to have the `realms` attribute contents inherited from the master process; the `<RateLaw>` element may use the `xref` attribute to link the rate law to that of a previously read in process definition, and take over any missing information from that rate-law.

The process definition file that describes the effect of calcite dissolution on calcite Production Time concentration is reproduced below. It can be found in `proc_calc_diss_pt.xml` in the directory `src-mcg/xml`.

```
1  <?xml version="1.0"?>
2
3  <!-- Provide the 'master' attribute to derive 'realms' content
       -->
4  <Process master="CalcDissolution">
5
6     <Names>
7        <Generic>CalcPTDissolution</Generic>
8        <Long>CalcitePT Dissolution</Long>
9        <ShortID>calcd_pt</ShortID>
10    </Names>
11
```

```
12
13    <ChemicalReaction>
14
15       <Reactant id="r1">
16          <Name>CalcitePT</Name>
17          <StoechCoeff>1</StoechCoeff>
18       </Reactant>
19
20    </ChemicalReaction>
21
22
23    <!-- Provide the 'xref' attribute to derive the missing
24         information - simply copy the pp_XYZ from the xref
25         process onto pp_RST from the process described here.
      -->
26    <RateLaw reference_id="r1" xref="CalcDissolution">
27
28       <Proportional>CalcitePT</Proportional>
29
30    </RateLaw>
31
32 </Process>
```

5.2 Isotopic Properties of Solids

The treatment of isotopic signatures of solids is very similar to that of the age tracer above, calling upon the inheritance of properties and rate law parameters. Isotopic signatures of solids are considered to belong to a general class of "colours", i. e., properties devoid of volume. Solids' colours are characterized by a `class` attribute set to `SolidColour`, as in the description file `calc_c13.xml` for ^{13}C in calcite ($Ca^{13}CO_3$):

```
1  <?xml version="1.0"?>
2
3  <Solid type="normal" class="SolidColour" master="Calcite">
4
5     <Names>
6        <Generic>Calcite_C13</Generic>
7        <Long>Calcite C13</Long>
8        <ShortID>calc_c13</ShortID>
9     </Names>
```

```
10
11
12     <PhysicalProperties/>
13
14
15     <ConservationProperties/>
16
17 </Solid>
```

The dissolution of $Ca^{13}CO_3$ is then linked to the general calcite dissolution with the process description file `proc_calc_c13_diss.xml` :

```
1  <?xml version="1.0" encoding="US-ASCII"?>
2
3  <Process realms="reaclay">
4
5     <Names>
6        <Generic>CalcC13Dissolution</Generic>
7        <Long>Calcite C13 Dissolution</Long>
8        <ShortID>calcd_c13</ShortID>
9     </Names>
10
11
12    <ChemicalReaction>
13
14       <Reactant id="r1">
15          <Name>Calcite_C13</Name>
16          <StoechCoeff>1</StoechCoeff>
17       </Reactant>
18
19
20       <Product id="p1">
21          <Name>DIC13</Name>
22          <StoechCoeff>1</StoechCoeff>
23       </Product>
24
25    </ChemicalReaction>
26
27
28    <RateLaw reference_id="r1" xref="CalcDissolution">
29       <Proportional>Calcite_C13</Proportional>
30    </RateLaw>
```

```
31
32 </Process>
```

6 Building `libmedusa.a`

The executable of MEDUSACOCGEN is called `medusa_cocogen_xml`, and is built in its source directory `src-mcg`. It is normally called from the `Makefile` in `src-med`, which in turn may be done from the `Makefile` of a particular application.

It can nevertheless also be executed from `src-mcg`. In this case, the code is only generated and left in `src-mcg/gen`, but not copied over into the source tree under `src-med`. When called by

```
cd [...]/src-med
make codegen
```

the generated code is furthermore copied into `src-med/gen`. Some parts offer a certain degree of customization, as indicated by the final message printed during the `make codegen`. MEDUSACOCOGEN, however, produces fully usable general purpose templates for those parts (subroutines essentially). These can be used as is with

```
make usetemplates
```

or the templates, that can be found in `src-med/gen/template`, can be adapted, stored in a safe place (e.g., inside the source file directory for your application) and copied or linked into `src-med/gen/include`, from where the compiler includes them. As soon as the required files are in place, the generation of `libmedusa.a` can then be completed by

```
make libmedusa.a
```

The resulting library archive `libmedusa.a`, together with all the `*.mod` files, must then be made accessible to the building procedure for your application.

A Rate Law Library

The rate law library already includes a series of modules to cover the most common rate law expressions. The source files of these so-called `MODLIB` modules are located in `src-mcg/lib`. Each module therein provides an implementation of one rate-law expression. The library of rate law functions can be extended. New rate law expressions can be made available by developing Fortran 95 modules similar to those in the `modlib_xyz.F` source files

in `src-mcg/lib`. The way to to this has been designed such that a maximum of flexibility can be combined with a minimum of processing overhead.

In this appendix section, we will first describe the format of these special source files, provide a commented example, illustrate how to request a particulate rate law in a process definition file and how to transmit the required information.

A.1 Source File Format for `MODLIB` Rate Law Files

The module source files must be structured as follows:

- First comes a preamble which is delimited by a pre-processor switch `#ifdef CFG_MEDUSACOCOGEN ... #endif`. It is made of a series of Fortran 95 namelists providing the meta-information required by the MEDUSACOCOGEN to integrate the rate-law module into MEDUSA. The first of these namelists provides the general characteristics of the rate-law implementation:
  - name of the subroutine that is used to evaluate the rate-law for a given set of parameter and component concentration values
  - name of the derived type that collects the process parameter information: parameters include here the references to modelled solid sediment and porewater components, parametrized components, parametrized solubilities, solubility products, etc., as well as global constants
  - number of parameters
  - a formal expression of the rate law implemented, using a simple substitution scheme for the rate law parameters.

  The next ones provide the informations about the parameters. There must be one such namelist per parameter.

- The actual module source code follows the preamble. It must
  - define a derived type definition to encapsulate the parameters of the rate law.
  - contain a subroutine to evaluate the rate law and its derivatives with respect to all the modelled components.

For MEDUSACOCOGEN, `MODLIB` files are data files with a one-line header and data organized in a series of namelists, where the first one specifies

how many there are; for MEDUSA, `MODLIB` files are source files, where the initial (meta-)data section is hidden, because `CFG_MEDUSACOCOGEN` will not be defined during compilation.

It is the developer's responsibility to ensure that the metadata in the preamble is consistent with the module source code.

A.1.1 Preamble: Metadata

As explained above, the preamble is composed of a series of Fortran namelists that hold meta-data about the subroutine that is used to evaluate the rate law expression. This part of the file is hidden by the pre-processor switch `#ifdef CFG_MEDUSACOCOGEN` ... `#endif` during compilation of the file as the source code of a module. The first namelist is

```
&ratelaw_config
c_name          = 'RATELAW_NAME'
c_pp_type       = 'PP_RATELAW_NAME'
n_param         = n
c_expression    = 'Rate-law fct({#1}, {#2}, ..., {#nn})'
/
```

where

- `c_name` holds the name under which the rate law is registered in MEDUSA; it must be unique in the current model instance.
- `c_pp_type` holds the name of the Fortran derived type that will be defined and that holds the parameters of the rate law. It should be equal to '`PP_`' followed by the rate law name as given by `c_name`.[3] By "parameters" we understand here all the concentration references and constants that enter the rate law expression.
- `n_param` holds the number n of parameters that are required describe the rate law adequately.
- `c_expression` gives a the format string for the mathematical representation of the ratelaw. Each parameter is referred to by a token '`{#`*i*`}`', where i refers to the order at which the namelist that defines it appears hereafter (from 1 for the first to n for the last).

[3]The `TYPE` name can theoretically be freely chosen. However, historically, `c_pp_type` was set to `'PP_' // UPCASE(c_name)`. Some parts of MEDUSACOCOGEN might still rely on this.

This first namelist is followed by n namelists, of which each has the following layout:

```
&ratelaw_data
c_kindofparam    = '...'
c_typecomponame  = '...'
c_xmltagname     = '...'
c_xmlattstocheck = '...'
c_dummylabel     = '...'
/
```

In this second type of namelists

- `c_kindofparam` determines the kind of parameter
  - `gk` – *global constant*, currently to be read in from `medusa.rrp` at runtime
  - `pf` – *parameter function*, such as a solubility product
  - `pc` – *parametrized concentration*
  - `io` – *model variable*, which MEDUSA refers to by its `io` index.
- `c_typecomponame` is the name, which can be freely chosen, by which the parameter is going to be referred to (e.g., `SolubilityConstant`, `RateConstant`, ...), and that will be used as a component name in the declaration of the `TYPE PP_RATELAW_NAME`. It should be meaningful and must follow the standard requirements for Fortran identifiers.
- `c_xmltagname` is optional. By default, the derived type component name given by `c_typecomponame` is also used as the XML tag name for referring to the parameter in the XML file. This can be overridden by setting `c_xmltagname` to a different name.
- `c_xmlattstocheck` is optional. It can be used to provide a comma-separated list of attributes that can be used in the parameter's tag name to point out possible pathways to MEDUSACOCOGEN to derive the required information. The two most common applications involve `type` and `code` attributes.

  A `type` attribute can be used for rate constants or rate orders, which are generally global constants, to specify that
  - the values for these constants should be read from a configuration file (`type="globalconstant"`);

– they should use one of MEDUSA's known parameter constants, e.g., from `MOD_BASIC_DATA` module (`type="basicconstant"` – not yet implemented, but see below for a workaround based upon parameter functions);

A `code` attribute can be used with elements related to parameter functions or parametrized concentrations to indicate which parametrization to chose, or how the evaluate it:

– typically, for a solubility product (exemplified here by an XML element `<OmegaSolubilityProduct>`), the `<RateLaw>` entry

```
<OmegaSolubilityProduct code="SolubilityProduct">
   Aragonite
</OmegaSolubilityProduct>
```

instructs MEDUSACACOCGEN to base the parameter function evaluation on the code provided in the `<SolubilityProduct>` entry in the `<CodeBits>` section of the component whose generic name is `Aragonite`.

– For a parameterized total concentration (exemplified here by an XML element `<ConcProductParamConc>`) the `<RateLaw>` entry

```
<ConcProductParamConc code="TotalConcentration">
   Ca
</ConcProductParamConc>
```

instructs MEDUSACACOCGEN to base the parameterization evaluation on the code provided in the `<TotalConcentration>` entry of the `<CodeBits>` section of the component whose generic name is `Ca`;

– by using the special value `code="verbatim"`, one may even provide Fortran one-line expressions for the parametrization: e.g., for a parametrized saturation concentration (exemplified here by an XML element `<SaturationConc>`), the `<RateLaw>` entry

```
<SaturationConc code="verbatim">
   cct_ksp_calc/cct_ttcc_ca
</SaturationConc>
```

instructs MEDUSACACOCGEN to insert the content of the element "as is" (verbatim) into the Fortran code.

This is also the workaround that can be used to overcome the not-yet-implemented processing of `type="basicconstant"` for global constants.

- `c_dummylabel` defines a (short) symbol for parameters that are not of kind `io` when substituting the tokens in `c_expression` from the first namelist by actual names – `io` parameters refer to modelled variables, whose tokens are going to be substituted by their generic names.

A.1.2 Code Part

The Fortran source code for the module itself directly follows the preamble, which is delimited by `#ifdef CFG_MEDUSACOCOGEN ... #endif`.

The module and its contents must strictly comply with the following naming scheme, which allows MEDUSA to use it correctly.

1. The module name must by equal to the name of the rate law as given by `c_name` in the `ratelaw_config` namelist, prefixed by '`MODLIB_`'.

2. the module must declare a new derived type, with the name given by `c_pp_type` (also from the first namelist). This derived type must include n elements (as given by `n_param`), whose names are given by the `c_typcomponame` entries of the `ratelaw_data` namelists.

3. The module must contain a subroutine, whose name is given by `c_name` (`RATELAW_NAME` in this example) and whose dummy argument list must be as follows:

```
 SUBROUTINE RATELAW_NAME(pp_param, ac,
&                        arate, darate_dac)

 TYPE(PP_RATELAW_NAME), INTENT(IN)
&  :: pp_param
 DOUBLE PRECISION, DIMENSION(:), INTENT(IN)
&  :: ac
 DOUBLE PRECISION, INTENT(OUT)
&  :: arate
 DOUBLE PRECISION, DIMENSION(SIZE(ac)), INTENT(OUT)
&  :: darate_dac
 OPTIONAL :: arate, darate_dac
```

   Notice that the `pp_param` dummy variable must be of the `TYPE` just declared in the module.

   This subroutine must be able to evaluate the rate law expression as a function of the component concentrations provided in `ac` and return the result in `arate`, and to evaluate the derivatives of the rate law with respect to the `ac` components and return the results in `darate_dac`.

A.2 Commented Example: DELTAPC_POWN_C

Below, we reproduce commented excerpts from `modlib_deltapc_pown_c.F` which provides a subroutine for the evaluation of rate laws of the form $R = kC_1(K_{\mathrm{sp}} - P_1C_2)^n$ that can be used to describe the kinetics of aragonite dissolution.

Notice that starting with Fortran 95, it is possible to include comments in namelist files.

A.2.1 Preamble

The preamble starts with the `#ifdef CFG_MEDUSACOCOGEN` pre-processor directive:

```
#ifdef CFG_MEDUSACOCOGEN
&ratelaw_config
c_name          = 'DELTAPC_POWN_C'
c_pp_type       = 'PP_DELTAPC_POWN_C'
n_param         = 6
c_expression    = '{#1}␣*␣[{#6}]␣*␣({#2}␣-␣[{#4}][{#5}])**{#3}'
/
```

This `MODLIB` file implements a rate law called '`DELTAPC_POWN_C`' with six parameters (variable concentrations, constants, etc.). Details regarding these six parameters are provided by the six `ratelaw_data` namelists that follow.

```
! Parameter 1
&ratelaw_data
c_typecomponame  = 'RateConstant'
c_xmlattstocheck = 'type'
c_kindofparam    = 'gk'
c_dummylabel     = 'k'
/
```

The first parameter (referred to by the placeholder `{#1}` in `c_expression`) is a global constant. Its XML element in the `<RateLaw>` element of a process description file will be `<RateConstant>` since no `c_xmltagname` is given. Furthermore, MEDUSACOCOGEN should check the `type` attribute of `<RateConstant>` for extra information.

```
! Parameter 2
&ratelaw_data
c_typecomponame  = 'SolubilityProduct'
c_xmlattstocheck = 'code'
c_kindofparam    = 'pf'
```

```
20 c_dummylabel     = 'ksp'
21 /
```

Parameter #2 is a parameter function (`c_kindofparam = 'pf'`) according to `c_xmlattstocheck` and MEDUSACOCOGEN should look for a `code` attribute for information about which parameter function to use and how to apply it.

```
22 ! Parameter 3
23 &ratelaw_data
24 c_typecomponame  = 'RateOrder'
25 c_xmlattstocheck = 'type'
26 c_kindofparam    = 'gk'
27 c_dummylabel     = 'n'
28 /
```

Parameter #3 is similar to Parameter #1.

```
29 ! Parameter 4
30 &ratelaw_data
31 c_typecomponame  = 'acConcProductParam'
32 c_xmltagname     = 'ConcProductParam'
33 c_xmlattstocheck = 'code'
34 c_kindofparam    = 'pc'
35 /
```

Parameter #4 is similar to parameter #2, except that the element name for the derived type declaration should not be used in the XML files, where the one given by `c_xmltagname` should be used instead. Variable names for parameterized concentrations normally start with '`ac...`', in MEDUSA and the variable names that refer to indices for modelled components or species start by '`io...`'.

```
36 ! Parameter 5
37 &ratelaw_data
38 c_typecomponame  = 'ioConcProductSpecies'
39 c_xmltagname     = 'ConcProductSpecies'
40 c_kindofparam    = 'io'
41 /
42 ! Parameter 6
43 &ratelaw_data
44 c_typecomponame  = 'ioProportional'
45 c_xmltagname     = 'Proportional'
46 c_kindofparam    = 'io'
47 /
```

Parameters #5 and #6 refer to modelled variables, according to the value of `c_kindofparam`. MEDUSACOCOGEN must derive their `io` indices.

```
48 #endif
```

The line with the `#endif` ends the preamble.

A.2.2 Module Code

```
49 !---+----1----+----2----+----3----+----4----+----5----+----6-
50 !============================================================
51       MODULE MODLIB_DELTAPC_POWN_C
52 !============================================================
```

The name of the module must be equal to the chosen name for the rate law, as given by the `c_name` from the `ratelaw_config` namelist above, prefixed by 'MODLIB_'.

```
53
54       ! For laws of the form k * c3 * (k_sp - p1*c2)**n
55       ! (as DELTACC_POWN_P, with c1 parameterized)
56
57
58       IMPLICIT NONE
59
60       TYPE PP_DELTAPC_POWN_C
61         DOUBLE PRECISION :: RateConstant          ! k value
62         DOUBLE PRECISION :: SolubilityProduct     ! k_sp value
63         DOUBLE PRECISION :: RateOrder             ! n value
64         DOUBLE PRECISION :: acConcProductParam    ! p1 value
65         INTEGER          :: ioConcProductSpecies  ! io_c2
66         INTEGER          :: ioProportional        ! io_c3
67       END TYPE
68
```

The module must provide a derived type collecting all the parameters of the rate law. It is recommended to name this derived type by the name of the rate law, prefixed by 'PP_'. The names of the elements of the derived type are given by the `c_compotypename` parts of the `ratelaw_data` namelist above.

```
69
70 !     Sample usage in the XML declarations:
71 !
72 !     <RateLaw reference_id="r1" subr="DELTAPC_POWN_C">
```

```
!
!      <RateConstant type="globalconstant"/>
!      <RateOrder    type="globalconstant"/>
!      <SolubilityProduct code="SolubilityProduct">Aragonite
   </SolubilityProduct>
!      <ConcProductParam  code="TotalConcentration">Ca</
   ConcProductParam>
!      <ConcProductSpecies>CO3</ConcProductSpecies>
!      <Proportional>Aragonite</Proportional>
!
!     </RateLaw>


      CONTAINS

!-----------------------------------------------------------------
      SUBROUTINE DELTAPC_POWN_C(pp_param, ac, azdn,
     &                          arate, darate_dac)
!-----------------------------------------------------------------

      IMPLICIT NONE


! Argument list variables
! -----------------------

      TYPE(PP_DELTAPC_POWN_C),               INTENT(IN)
     &  :: pp_param
      DOUBLE PRECISION, DIMENSION(:),        INTENT(IN)
     &  :: ac
      DOUBLE PRECISION,                      INTENT(IN)
     &  :: azdn
      DOUBLE PRECISION,                      INTENT(OUT)
     &  :: arate
      DOUBLE PRECISION, DIMENSION(SIZE(ac)), INTENT(OUT)
     &  :: darate_dac

      OPTIONAL :: arate, darate_dac

```

Finally, the module must contain a subroutine that has the name of the rate

law and the standardized dummy argument list.

```
110
111 ! Local variables
112 ! ---------------
113       ...
114
115
116 ! Instructions
117 ! ------------
118       ...
119
120       IF (PRESENT(arate)) THEN
121         arate = ...
122       ENDIF
123
124       IF (PRESENT(darate_dac)) THEN
125         darate_dac(:) = 0.0D+00
126         darate_dac(...) = ...
127       ENDIF
128
129       RETURN
130
131 !-------------------------------------------------------------
132       END SUBROUTINE DELTAPC_POWN_C
133 !-------------------------------------------------------------
134
135
136 !=============================================================
137       END MODULE MODLIB_DELTAPC_POWN_C
138 !=============================================================
```

A.2.3 Application in a Process Description File

The information provided in the preamble of this MODLIB file can now be used as follows in the <RateLaw> element of a process description file:

```
<RateLaw reference_id="r1" subr="DELTAPC_POWN_C">
   <RateConstant type="globalconstant"/>
   <RateOrder    type="globalconstant"/>
   <SolubilityProduct code="SolubilityProduct">
```

```
      Aragonite
   </SolubilityProduct>
   <ConcProductParam  code="TotalConcentration">
      Ca
   </ConcProductParam>
   <ConcProductSpecies>CO3</ConcProductSpecies>
   <Proportional>Aragonite</Proportional>
</RateLaw>
```

As indicated by the `subr` attribute of `<RateLaw>`, the subroutine that should be called to evaluate this rate law is `DELTAPC_POWN_C`. MEDUSACOCOGEN will look for that subroutine in `modlib_deltapc_pown_c.F` (name of the subroutine, prefixed by '`MODLIB_`', all set to lower case for the file name of the source code) to read in the information about the parameters required for that rate law. To set up the code for initializing the components of the process parameter type `PP_DELTAPC_POWN_C`, the child elements of `<RateLaw>` will be interpreted as follows:

- Both the `RateConstant` (parameter #1) and the `RateOrder` (parameter #2) are global constants to be read from an external configuration file (`medusa.rrp`) by MEDUSA at runtime, as indicated by `type="globalconstant"`. This also explains why both elements are left empty – all the information required is already present.

- The solubility product (parameter #3, child `<SolubilityProduct>`) should be that of the component with the generic name `Aragonite`. The code attached as `<SolubilityProduct>` to that component should be used for this purpose, as indicated by `code="SolubilityProduct"`.

- The parametrized concentration in the concentration product (parameter #4, child `<ConcProductParam>`) should be the component with the generic name `Ca`. Its concentration is evaluated with the code attached under `<TotalConcentration>` to that component, as indicated by `code="TotalConcentration"`. Alternatively, this could have been done by the following form of the element for parameter #4, `<ConcProductParam>`:

  ```
  <ConcProductParam  code="verbatim">
     cct_ttcc_ca
  </ConcProductParam>
  ```

  if the short ID of `Ca` is `ca`, and that the total concentration for a parametrized component, that has a `<TotalConcentration>` entry in

its <CodeBits> and that this total concentrations is stored in a variable whose name is obtained by prefixing the short ID with cct_ttcc_. The special value verbatim of the code attribute indicates that the contents of the <ConcProductParam> elements are to be inserted "as is" (verbatim) into the Fortran code.

- The modelled concentration (parameter #5) in the concentration product should be the component with the generic name CO3.
- The proportional concentration (parameter #6) in the rate law should be the component with the generic name Aragonite.

A.3 Rate Law Library Reference

The basic MEDUSACOCOGEN comes already with a rich library of rate law MODLIB fies for rate laws. Their interfaces are detailed below, together with application examples.

LINEAR1

Rate laws of the form

$$R = kC_1$$

Example:

```
<RateLaw reference_id="ref_id" subr="LINEAR1">
   <RateConstant    type="globalconstant"/>
   <Proportional>GenNameCompo1</Proportional>
</RateLaw>
```

PLINEAR1

Rate laws of the form

$$R = \lambda C_1$$

where the rate constant λ is obtained by a parameter function or a parameter constant from MOD_BASICDATA_MEDUSA.
Example: Radiocarbon decay in calcite

$$R = \lambda[\text{CalciteC14}]$$

```
<RateLaw reference_id="ref_id" subr="PLINEAR1">
   <RateConstant code="verbatim">
      dp_lambda_C14
   </RateConstant>
   <Proportional>CalciteC14</Proportional>
</RateLaw>
```

PRODUCT2

Rate laws of the form

$$R = kC_1C_2$$

Example: Fe^{2+} oxidation by O_2

$$R = k[\text{Fe2plus}][\text{O2}]$$

```
<RateLaw reference_id="r1" subr="PRODUCT2">
   <RateConstant    type="globalconstant"/>
   <Proportional1>Fe2plus</Proportional1>
   <Proportional2>O2</Proportional2>
</RateLaw>
```

PPRODUCT2

Rate laws of the form

$$R = \lambda C_1C_2$$

where the rate constant λ is obtained by a parameter function or a parameter constant from `MOD_BASICDATA_MEDUSA`.

PRODUCT2_SIDINH1

Rate laws of the form

$$R = k\ C_1\ C_2\ \frac{1}{1 + \exp\left((C_3 - K_{\text{ic}})/K_{\text{is}}\right)}$$

with a non normalized sigmoid inhibition function
Example: organic matter oxidation by MnO_2 reduction, with inhibition by oxygen

$$R = k\ [\text{OrgMatter}]\ [\text{MnO2}]\ \frac{1}{1 + \exp\left(([\text{O2}] - K_{\text{ic}})/K_{\text{is}}\right)}$$

```
<RateLaw reference_id="r1" subr="PRODUCT2_SIDINH1">

   <RateConstant    type="globalconstant"/>
   <Proportional1>OrgMatter</Proportional1>
   <Proportional2>MnO2</Proportional2>
   <InhibConstant   type="globalconstant"/>
   <InhibScale      type="globalconstant"/>
   <InhibitionConc>O2</InhibitionConc>
</RateLaw>
```

DELTA1_C

Rate laws of the form

$$R = kC_1(K_{\text{sat}} - C_2)$$

Example: opal dissolution

$$R = k[\text{Opal}](K_{\text{sat}} - [\text{H4SiO4}])$$

```
<RateLaw reference_id="refid" subr="DELTA1_C">
   <RateConstant type="globalconstant"/>
   <DeltaSaturationConc code="SaturationConc">
     Opal
   </DeltaSaturationConc>
   <DeltaConc>H4SiO4</DeltaConc>
   <Proportional>Opal</Proportional>
</RateLaw>
```

Alternatively

```
<RateLaw reference_id="refid" subr="DELTA1_C">
   <RateConstant type="globalconstant"/>
   <DeltaSaturationConc code="verbatim">
      cct_ksat_opal
   </DeltaSaturationConc>
   <DeltaConc>H4SiO4</DeltaConc>
   <Proportional>Opal</Proportional>
</RateLaw>
```

DELTACC_POWN_C

Rate laws of the form

$$R = kC_1(K_{\text{sp}} - C_2C_3)^n$$

Example: aragonite dissolution

$$R = k[\text{Aragonite}](K_{\text{sp}} - [\text{Ca}][\text{CO3}])^n$$

when Ca is also an explicitly modelled component (`type="normal"`):

```
<RateLaw reference_id="r1" subr="DELTACC_POWN_C">
   <RateConstant type="globalconstant"/>
   <RateOrder    type="globalconstant"/>
   <SolubilityProduct code="SolubilityProduct">
      Aragonite
   </SolubilityProduct>
   <ConcProductSpecies1>Ca</ConcProductSpecies1>
   <ConcProductSpecies2>CO3</ConcProductSpecies2>
   <Proportional>Aragonite</Proportional>
</RateLaw>
```

DELTAPC_POWN_C

Rate laws of the form

$$R = kC_1(K_{\text{sp}} - P_1C_2)^n$$

Example: aragonite dissolution

$$R = k[\text{Aragonite}](K_{\text{sp}} - [\text{Ca}][\text{CO3}])^n$$

when Ca is a parametrized component (with `type="parameterized"`):

```
<RateLaw reference_id="r1" subr="DELTAPC_POWN_C">
   <RateConstant type="globalconstant"/>
   <RateOrder    type="globalconstant"/>
   <SolubilityProduct code="SolubilityProduct">
      Aragonite
   </SolubilityProduct>
   <ConcProductParam  code="TotalConcentration">
      Ca
   </ConcProductParam>
   <ConcProductSpecies>CO3</ConcProductSpecies>
   <Proportional>Aragonite</Proportional>
</RateLaw>
```

OMEGA1_C

Rate laws of the form

$$R = kC_1(1 - \frac{C_2}{K_{\text{sat}}})$$

Example: calcite dissolution following linear kinetics

$$R = k[\text{Calcite}](1 - [\text{CO3}]/K_{\text{sat}})^n$$

when Ca is a parametrized component (with `type="parameterized"`):

```
<RateLaw reference_id="r1" subr="OMEGA1_C"
         requires="Calcite,␣Ca">
   <RateConstant type="globalconstant"/>
   <OmegaConc>CO3</OmegaConc>
   <OmegaSaturationConc code="verbatim">
      cct_ksp_calc/cct_ttcc_ca
   </OmegaSaturationConc>
   <Proportional>Calcite</Proportional>
</RateLaw>
```

OMEGA1_POWN_C / OMEGA1_POWN_A

Rate laws of the form

$$R = kC_1(1 - \frac{C_2}{K_{\text{sat}}})^n$$

Example: calcite dissolution following non-linear kinetics

$$R = k[\text{Calcite}](1 - [\text{CO3}]/K_{\text{sat}})^n$$

when Ca is a parametrized component (with `type="parameterized"`):

```
<RateLaw reference_id="r1" subr="OMEGA1_C"
         requires="Calcite,␣Ca">
   <RateConstant type="globalconstant"/>
   <RateOrder    type="globalconstant"/>
   <OmegaConc>CO3</OmegaConc>
   <OmegaSaturationConc code="verbatim">
      cct_ksp_calc/cct_ttcc_ca
   </OmegaSaturationConc>
   <Proportional>Calcite</Proportional>
</RateLaw>
```

OMEGA1_POWN_C and OMEGA1_POWN_A are identical except for one detail: while OMEGA1_POWN_C is safe-guarded against negative values of the proportional concentration (the rate is set to zero for negative values of that concentration), OMEGA1_POWN_A is not. Safeguarding against negative concentration values helps to stabilize the model, in case the numerical procedure leads to negative values for physical constituents. However, production time concentrations can be negative (if the average production time is negative). For production time equations, OMEGA1_POWN_A must therefore be used instead of OMEGA1_POWN_C. Parameter inheritance works without problems between the two versions, as they share the same parameter names.

OMEGACC_C

Rate laws of the form

$$R = k\ C_1\ \left(1 - \frac{C_2 C_3}{K_{\text{sp}}}\right)$$

Example: calcite dissolution following linear kinetics

$$R = k\ [\text{Calcite}]\ \left(1 - \frac{[\text{Ca}]\ [\text{CO3}]}{K_{\text{sp}}}\right)$$

when Ca is also an explicitly modeled component (type="normal"):

```
<RateLaw reference_id="r1" subr="OMEGACC_C">
   <RateConstant type="globalconstant"/>
   <ConcProductSpecies1>Ca</ConcProductSpecies1>
   <ConcProductSpecies2>CO3</ConcProductSpecies2>
   <SolubilityProduct code="SolubilityProduct">
      Calcite
   </SolubilityProduct>
   <Proportional>Calcite</Proportional>
</RateLaw>
```

MONOD1

Rate laws of the form

$$R = k\ \frac{C_1}{K_{\text{hsat}} + C_1}$$

Example:

```
<RateLaw reference_id="refid" subr="MONOD1">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
```

```
   <MonodConc>GenNameCompo1</MonodConc>
</RateLaw>
```

MONOD1_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}$$

Example: oxic degradation of organic matter

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{O2}]}{K_{\text{hsat}} + [\text{O2}]}$$

```
<RateLaw reference_id="r1" subr="MONOD1_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>O2</MonodConc>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_HYPINH1_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{K_{\text{inh}}}{K_{\text{inh}} + C_3}$$

Example: organic matter degradation by full denitrification, with inhibition by oxygen

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{NO3}]}{K_{\text{hsat}} + [\text{NO3}]}\ \frac{K_{\text{inh}}}{K_{\text{inh}} + [\text{O2}]}$$

```
<RateLaw reference_id="r1" subr="MONOD1_HYPINH1_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>NO3</MonodConc>
   <InhibConstant   type="globalconstant"/>
   <InhibitionConc>O2</InhibitionConc>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_HYPINH2_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{K_{\text{inh},1}}{K_{\text{inh},1} + C_3}\ \frac{K_{\text{inh},2}}{K_{\text{inh},2} + C_4}$$

Example: organic matter degradation by sulphate reduction, with inhibition by oxygen and nitrate

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{SO4}]}{K_{\text{hsat}} + [\text{SO4}]}\ \frac{K_{\text{inh},1}}{K_{\text{inh},1} + [\text{O2}]}\ \frac{K_{\text{inh},2}}{K_{\text{inh},2} + [\text{NO3}]}$$

```
<RateLaw reference_id="r1" subr="MONOD1_HYPINH2_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>SO4</MonodConc>
   <InhibConstant1  type="globalconstant"/>
   <InhibitionConc1>O2</InhibitionConc1>
   <InhibConstant2  type="globalconstant"/>
   <InhibitionConc2>NO3</InhibitionConc2>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_HYPINH3_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{K_{\text{inh},1}}{K_{\text{inh},1} + C_3}\ \frac{K_{\text{inh},2}}{K_{\text{inh},2} + C_4}\ \frac{K_{\text{inh},3}}{K_{\text{inh},3} + C_5}$$

Example: organic matter degradation by sulphate reduction, with inhibition by oxygen, nitrate and iron oxide

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{SO4}]}{K_{\text{hsat}} + [\text{SO4}]}\ \frac{K_{\text{inh},1}}{K_{\text{inh},1} + [\text{O2}]}\ \frac{K_{\text{inh},2}}{K_{\text{inh},2} + [\text{NO3}]} \times \frac{K_{\text{inh},3}}{K_{\text{inh},3} + [\text{FeOH3}]}$$

```
<RateLaw reference_id="r1" subr="MONOD1_HYPINH3_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>SO4</MonodConc>
```

```
   <InhibConstant1  type="globalconstant"/>
   <InhibitionConc1>O2</InhibitionConc1>
   <InhibConstant2  type="globalconstant"/>
   <InhibitionConc2>NO3</InhibitionConc2>
   <InhibConstant3  type="globalconstant"/>
   <InhibitionConc3>FeOH3</InhibitionConc3>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_SIDINH1_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{1}{1 + \exp\left((C_3 - K_{\text{ic}})/K_{\text{is}}\right)}$$

with a non normalized sigmoid inhibition function
Example: organic matter degradation by full denitrification, with inhibition by oxygen

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{NO3}]}{K_{\text{hsat}} + [\text{NO3}]}\ \frac{1}{1 + \exp\left(([\text{O2}] - K_{\text{ic}})/K_{\text{is}}\right)}$$

```
<RateLaw reference_id="r1" subr="MONOD1_HYPINH1_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>NO3</MonodConc>
   <InhibConstant   type="globalconstant"/>
   <InhibScale      type="globalconstant"/>
   <InhibitionConc>O2</InhibitionConc>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_SIDINH2_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{1}{1 + \exp\left((C_3 - K_{\text{ic},1})/K_{\text{is},1}\right)}$$

$$\times\ \frac{1}{1 + \exp\left((C_4 - K_{\text{ic},2})/K_{\text{is},2}\right)}$$

with two non normalized sigmoid inhibition functions
Example: organic matter degradation by sulphate reduction, with inhibition by oxygen and nitrate

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{SO4}]}{K_{\text{hsat}} + [\text{SO4}]}\ \frac{1}{1 + \exp\left(([\text{O2}] - K_{\text{ic},1})/K_{\text{is},1}\right)} \times \frac{1}{1 + \exp\left(([\text{NO3}] - K_{\text{ic},2})/K_{\text{is},2}\right)}$$

```
<RateLaw reference_id="r1" subr="MONOD1_SIGINH2_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>SO4</MonodConc>
   <InhibConstant1  type="globalconstant"/>
   <InhibScale1     type="globalconstant"/>
   <InhibitionConc1>O2</InhibitionConc1>
   <InhibConstant2  type="globalconstant"/>
   <InhibScale2     type="globalconstant"/>
   <InhibitionConc2>NO3</InhibitionConc2>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

MONOD1_SIDINH3_C

Rate laws of the form

$$R = k\ C_1\ \frac{C_2}{K_{\text{hsat}} + C_2}\ \frac{1}{1 + \exp\left((C_3 - K_{\text{ic},1})/K_{\text{is},1}\right)} \times \frac{1}{1 + \exp\left((C_4 - K_{\text{ic},2})/K_{\text{is},2}\right)} \times \frac{1}{1 + \exp\left((C_5 - K_{\text{ic},3})/K_{\text{is},3}\right)}$$

Example: organic matter degradation by sulphate reduction, with inhibition by oxygen, nitrate and iron oxide

$$R = k\ [\text{OrgMatter}]\ \frac{[\text{SO4}]}{K_{\text{hsat}} + [\text{SO4}]}\ \frac{1}{1 + \exp\left(([\text{O2}] - K_{\text{ic},1})/K_{\text{is},1}\right)} \times \frac{1}{1 + \exp\left(([\text{NO3}] - K_{\text{ic},2})/K_{\text{is},2}\right)} \times \frac{1}{1 + \exp\left(([\text{FeOH3}] - K_{\text{ic},3})/K_{\text{is},3}\right)}$$

```
<RateLaw reference_id="r1" subr="MONOD1_SIDINH3_C">
   <RateConstant    type="globalconstant"/>
   <HalfSatConstant type="globalconstant"/>
   <MonodConc>SO4</MonodConc>
   <InhibConstant1  type="globalconstant"/>
   <InhibScale1     type="globalconstant"/>
   <InhibitionConc1>O2</InhibitionConc1>
   <InhibConstant2  type="globalconstant"/>
   <InhibScale2     type="globalconstant"/>
   <InhibitionConc2>NO3</InhibitionConc2>
   <InhibConstant3  type="globalconstant"/>
   <InhibScale3     type="globalconstant"/>
   <InhibitionConc3>FeOH3</InhibitionConc3>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

RAMP1

Rate laws of the form

$$R = k \ \min(\frac{C_1}{K_1}, 1)$$

Example:

```
<RateLaw reference_id="refid" subr="RAMP1">
   <RateConstant    type="globalconstant"/>
   <RampMaxConstant type="globalconstant"/>
   <RampConc>GenNameCompo1</RampConc>
</RateLaw>
```

RAMP1_C

Rate laws of the form

$$R = k \ C_1 \min(\frac{C_2}{K_1}, 1)$$

Example: oxic degradation of organic matter

$$R = k \ [\text{OrgMatter}] \ \min(\frac{[\text{O2}]}{K_{\max}}, 1)$$

```
<RateLaw reference_id="r1" subr="RAMP1_C">
   <RateConstant    type="globalconstant"/>
```

```
   <RampMaxConstant type="globalconstant"/>
   <RampConc>O2</RampConc>
   <Proportional>OrgMatter</Proportional>
</RateLaw>
```

B Law of Mass-Action Library

B.1 Source File Format for Law of Mass-Action `MODLIB` Files

The structure of the `MODLIB` files for Mass-Action Laws is similar to that of the Rate Law files:

- The files start with a preamble with a series of namelists providing the general information of the implemented law, such as the name by which it can be called and the number of parameters it has, as well as the parameter definition
- After the preamble starts the actual source code of the `MODLIB` module, which provides
  - a derived type that collects the parameters of the law
  - a first subroutine to evaluate the equilibrium relationship related to the law of mass-action
  - another subroutine to set the equilibrium constant in the derived type from the boundary conditions.

B.1.1 Preamble

The preamble is again hidden from the core module source by the preprocessor switch `#ifdef CFG_MEDUSACOCOGEN ... #endif`. It contains a series of namelists: the first one is

```
&eqlbrel_config
c_name          = 'EQLBREL_NAME'
c_ep_type       = 'EP_EQLBREL_NAME'
n_param         = n
c_expression    = 'Equil. rel. fct({#1}, {#2}, ..., {#nn}) = 0'
/
```

In this first namelist

- `c_name` holds the name under which the equilibrium relationship is registered in MEDUSA; it must be unique in the current model instance.

- `c_ep_type` holds the name of the Fortran derived type that will be declared and that collects the parameters of the relationship. It should be equal to '`EP_`' (*Equilibrium Relationship*) followed by the relationship name as given by `c_name`.[4] By "parameters" we understand here the equilibrium constant and all the concentration references that enter the rate law expression.

- `n_param` holds the number n of parameters that are required to set up the relationship `c_expression` gives a the format string for the mathematical representation of the equilibrium relationship. As in the Rate Law section, each parameter is referred to by a token '`{#i}`', where i refers to the order at which the namelist that defines it appears hereafter (from 1 for the first to n for the last).

This first namelist is followed by n namelists (where n is the number of parameters required to describe the equilibrium relationship adequately). Each one of these n namelists has the following layout:

```
&eqlbrel_data
c_typecomponame  = '...'
c_xmltagname     = '...'
c_kindofparam    = '...'
c_dummylabel     = '...'
/
```

In these second type of namelists

- `c_typecomponame` is the name (to be freely chosen), by which the parameter is going to be referred to (e.g., `EquilibConstant`, `ioReactant1`, ...) and that will be used as a component name in the definition of the `TYPE PE_EQLBREL_NAME`.

- `c_xmltagname` is optional. By default, the derived type component name given by `c_typecomponame` is also used as the XML tag name for referring to the parameter in the XML file. This can be overridden by setting `c_xmltagname` to a different name.

[4] The `TYPE` name can theoretically be freely chosen. However, historically, `c_ep_type` was set to `'EP_' // UPCASE(c_name)`. Some parts of MEDUSACOCOGEN might still rely on this.

- `c_kindofparam` determines the kind of parameter; there are less possibilities covered here than with the rate laws
  - `bc` – to be derived from the *boundary conditions*
  - `io` – *model variable*, which MEDUSA refers to by its `io` index.
- `c_dummylabel` allows to define a (short) symbol for parameters that are not of kind `io` when substituting the tokens in `c_expression` from the first namelist by actual names – `io` parameters refer to modelled variables, whose tokens are going to be substituted by their generic names.

B.1.2 Code Part

The Fortran source code for the module it self follows the preamble, which is encapsulated by `#ifdef CFG_MEDUSACOCOGEN ... #endif`.

The source code of the actual `MODLIB` module follows the preamble. It must fulfil the following requirements (in the examples quoted below, we use `EQLBREL_NAME` as a rate-law name – this name can be freely chosen, but must be unique in the library):

- its name must be equal to the name of the rate law, prefixed by '`MODLIB_`'
- it must make available a derived type whose name is given by the law name, prefixed by '`EP_`' (*Equilibrium Parameters*) and which encapsulates the parameters of the law;
- it must contain a first subroutine to evaluate the equilibrium relationship related to the law of mass-action, and which has the name of the law and the following standard dummy argument list:

```
 SUBROUTINE EQLBREL_NAME(ep_param, ac,
&                        aeqrl, daeqrl_dac)

 TYPE(EP_EQLBREL_NAME), INTENT(IN)
&  :: ep_param
 DOUBLE PRECISION, DIMENSION(:), INTENT(IN)
&  :: ac
 DOUBLE PRECISION, INTENT(OUT)
&  :: aeqrl
 DOUBLE PRECISION, DIMENSION(SIZE(ac)), INTENT(OUT)
&  :: daeqrl_dac
 OPTIONAL :: aeqrl, daeqrl_dac
```

- it must contain a second subroutine to initialize the equilibrium constant in the law of mass-action, and which has the name of the law prefixed by 'SET_EQUILCT_' and the following standard dummy argument list:

```
 SUBROUTINE SET_EQUILCT_EQLBREL_NAME(ep_param, ac)

 TYPE(EP_EQLBREL_NAME), INTENT(INOUT)
&  :: ep_param
 DOUBLE PRECISION, DIMENSION(:), INTENT(IN)
&  :: ac
```

B.2 Commented Example: MODLIB_R1R2P1P1

B.2.1 Preamble

```
#ifdef CFG_MEDUSACOCOGEN
&eqlbrel_config
c_name          = 'R1R2P1P1'
c_ep_type       = 'EP_R1R2P1P1'
n_param         = 4
c_expression    = '[{#4}]**2␣-␣{#1}␣*␣[{#2}]*[{#3}]␣=␣0'
/
! Parameter 1
&eqlbrel_data
c_typecomponame  = 'EquilibConstant'
c_kindofparam    = 'bc'  ! equilibrium constant, to be set
                         ! from the boundary conditions
c_dummylabel     = 'K'
/
! Parameter 2
&eqlbrel_data
c_typecomponame  = 'ioReactant1'
c_xmltagname     = 'Reactant1'
c_kindofparam    = 'io'
/
! Parameter 3
&eqlbrel_data
c_typecomponame  = 'ioReactant2'
c_xmltagname     = 'Reactant2'
c_kindofparam    = 'io'
```

```
/
! Parameter 4
&eqlbrel_data
c_typecomponame  = 'ioProduct1'
c_xmltagname     = 'Product1'
c_kindofparam    = 'io'
/
#endif
```

B.2.2 Module Code

```
!---+----1----+----2----+----3----+----4----+----5----+----6-
!=================================================================
      MODULE MODLIB_R1R2P1P1
!=================================================================

```

The name

```
      ! For equilibrium relationships
      ! P1**2 - K * R1*R2 = P1*P1 - K * R1*R2 = 0

      IMPLICIT NONE

      TYPE EP_R1R2P1P1
        INTEGER          :: ioReactant1     ! io of R1
        INTEGER          :: ioReactant2     ! io of R2
        INTEGER          :: ioProduct1      ! io of P1
        DOUBLE PRECISION :: EquilibConstant ! K value
      END TYPE

```

The module must provide ...

```

!     Sample usage in the XML declarations:
!
!    <LawOfMassAction subr="R1R2P1P1">
!      <Reactant1>CO2</Reactant1>
!      <Reactant2>CO3</Reactant2>
!      <Product1>HCO3</Product1>
!   </LawOfMassAction>
```

```
59
60
61        CONTAINS
62
63  !-------------------------------------------------------------
64        SUBROUTINE R1R2P1P1(ep_param, ac, aeqrl, daeqrl_dac)
65  !-------------------------------------------------------------
66
67        IMPLICIT NONE
68
69
70  ! Argument list variables
71  ! ----------------------
72
73        TYPE(EP_R1R2P1P1),                      INTENT(IN)
74       &  :: ep_param
75        DOUBLE PRECISION, DIMENSION(:),         INTENT(IN)
76       &  :: ac
77        DOUBLE PRECISION,                       INTENT(OUT)
78       &  :: aeqrl
79        DOUBLE PRECISION, DIMENSION(SIZE(ac)), INTENT(OUT)
80       &  :: daeqrl_dac
81
82        OPTIONAL :: aeqrl, daeqrl_dac
83
```

The module must contain a subroutine that has the name of the equilibrium and the standardized dummy argument list.

```
139
140 ! Local variables
141 ! ---------------
142       ...
143
144
145 ! Instructions
146 ! ------------
147       ...
148
149       IF (PRESENT(aeqrl)) THEN
150         aeqrl = ...
151       ENDIF
```

```
152
153
154       IF (PRESENT(daeqrl_dac)) THEN
155         daeqrl_dac(:)   =  0.0D+00
156         daeqrl_dac(...) =  ...
157       ENDIF
158
159
160   !-------------------------------------------------------------
161       END SUBROUTINE R1R2P1P1
162   !-------------------------------------------------------------
163
164
165
166   !-------------------------------------------------------------
167       SUBROUTINE SET_EQUILCT_R1R2P1P1(ep_param, ac)
168   !-------------------------------------------------------------
169
170       IMPLICIT NONE
171
172
173   ! Argument list variables
174   ! -----------------------
175
176       TYPE(EP_R1R2P1P1), INTENT(INOUT)           :: ep_param
177       DOUBLE PRECISION, DIMENSION(:), INTENT(IN) :: ac
178
```

Finally, the module must contain a subroutine that has the name of the rate law prefixed by '`SET_EQUILCT_`' and the standardized dummy argument list.

```
179
180   ! Local variables
181   ! ---------------
182       ...
183
184
185   ! Instructions
186   ! ------------
187       ...
188
189       ep_param%EquilibConstant = ...
```

```
190
191         RETURN
192
193   !-------------------------------------------------------------
194         END SUBROUTINE SET_EQUILCT_R1R2P1P1
195   !-------------------------------------------------------------
196
197
198   !=============================================================
199         END MODULE MODLIB_R1R2P1P1
200   !=============================================================
```

B.2.3 Application

B.3 Law of Mass-Action Library Reference

R1P1P2

The R1P1P2 law is for equilibrium relationships of the form

$$\frac{[P_1]\,[P_2]}{[R_1]} = K$$

or

$$[P_1]\,[P_2] - K[R_1] = 0$$

which arises, e.g., for the bicarbonate dissociation

$$\mathrm{HCO_3^-} + \mathrm{H_2O} \leftrightarrow \mathrm{CO_3^-} + \mathrm{H_3O^+}$$

The reactant and the two products must be components with `type="normal"`.

R1R2P1P1

The R1R2P1P1 law is for equilibrium relationships of the form

$$\frac{[P_1]^2}{[R_1]\,[R_2]} = K$$

or

$$[P_1]^2 - K[R_1]\,[R_2] = 0$$

which arises, e.g., for the carbonate equilibrium

$$\mathrm{CO_2} + \mathrm{CO_3^{2-}} + \mathrm{H_2O} \leftrightarrow 2\,\mathrm{HCO_3^-}$$

All of the reactants R_1 and R_2 as well as the product P_1 must be components with `type="normal"`.

R1R2P1P2

The R1R2P1P2 law is for equilibrium relationships of the form

$$\frac{[P_1]\,[P_2]}{[R_1]\,[R_2]} = K$$

or

$$[P_1]\,[P_2] - K[R_1]\,[R_2] = 0$$

. All of the reactants and products must be components with `type="normal"`.

P1P2

The P1P2 law is for equilibrium relationships of the form

$$[P_1]\,[P_2] = K$$

or

$$[P_1]\,[P_2] - K = 0.$$

A typical example of a reaction where this equilibrium relationship is appropriate is the self-ionization of water

$$2\,\mathrm{H_2O} \leftrightarrow \mathrm{OH^-} + \mathrm{H_3O^+}.$$

Both products must be components with `type="normal"`.