1 Introduction

Welcome to the documentation on the Python binding of ORCHESTRA! In this document, we will discuss the approach on how to use the chemical equilibrium software ORCHESTRA within Python using Pybind11. The document is part of the Master Thesis by Guido de Zeeuw for the Applied Earth Sciences master's programme at the TU Delft. The work discussed here is fully made with the collaboration of the creator of ORCHESTRA: Hans Meeussen. For information on the ORCHESTRA software, I invite you to check out the ORCHESTRA website: http://orchestra.meeussen.nl.

1.1 Introduction to ORCHESTRA

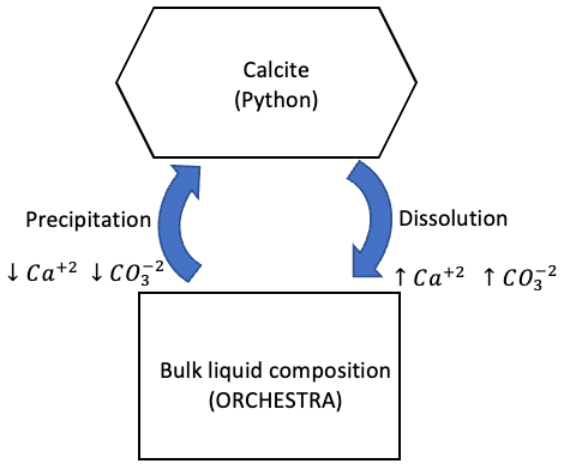
ORCHESTRA (**O**bject **R**epresenting **CHE**mical **S**peciation and **TRA**nsport) is a chemical equilibrium model developed in the early 2000's to be a valid replacement for other available software (Meeussen, 2003). It must be noted, however, that the more conventional chemical equilibrium software (e.g. MINTEQA2, PHREEQC, MINEQL and ECOSAT) are already very powerful. These programs work by selecting a set of chemical reactions and their reaction constants, followed by a correct calculation of the chemical speciation. However, the problem arises with the general structure regarding such software. While the user can select a set of chemical components, the reactions related to those components are already pre-defined within the source code. Although this allows for a user-friendly experience, it also poses little transparency and flexibility. For example, if one would desire to change or add model definitions (e.g. exclude chemical equilibrium reactions from the calculation), this would require changing and recompiling the source code. For this, the source code must available in the first place and even if this is the case, recompiling may take up a significant amount of time as the source code files may be very large.

This is the main reason why ORCHESTRA was created. With ORCHESTRA, the model definitions (e.g. chemical reactions, kinetics, general conditions) are defined in an input file. The calculation is done in a separate kernel that uses this input file. This means that the calculation kernel (or source code) does not contain any chemical information in itself and only acts as a calculator object. By structuring the software this way, changing model definitions does not require changing and recompile the source code. This allows the user to quickly select the reactions that the solver should include for calculating chemical equilibrium. The user can thereby exclude reactions that are described kinetically from those partaking in chemical equilibrium. Separating the model definitions from the source code also means that the calculation kernel is extremely small and, because of that, very efficient.

Although the calculator in itself is less complex and more flexible compared to the other equilibrium models, the difficulty with ORCHESTRA lies in correctly defining the chemical input file. To counteract the loss of a user-friendly experience, a graphical user interface (GUI) is developed to interactively set up the input file. The GUI can be downloaded from http://orchestra.meeussen.nl. The user interface comes standard with the ORCHESTRA-ready database for PHREEQC and various adsorption models. If needed, the user can create its own database with user-defined reactions.

1.2 Using ORCHESTRA in Python: why?

As ORCHESTRA allows the user to exclude reactions from the chemical equilibrium solver, ORCHESTRA is very useful when combined with kinetic solvers. Kinetic solvers are used to monitor the state of the system by keeping track of the reactants and products for a given reaction. A simple example is the dissolution of (e.g.) calcite, where the dissolution rate is described by a maximum rate and various inhibiting factors for different conditions (e.g. pH, liquid composition and available minerals). While the dissolution of calcite is described kinetically, other reactions (e.g. equilibrium between $H^+ + CO_3^{-2} \leftrightarrow H_2CO_3$) can be treated as instantaneous. In other words: they are not described kinetically but their relation can be calculated using ORCHESTRA. Below is an illustrative scheme on how this relation between a kinetic solver and ORCHESTRA works.



The illustration describes the effects of calcite dissolution/precipitation. Upon dissolution, the bulk composition of the liquid phase increases with CO_3^{-2} and Ca^{+}. On the other hand, precipitation 'extracts' CO_3^{-2} and Ca^{+2} from the liquid system to form calcite. The kinetic solver (**in Python**) therefore monitors the *changes* in bulk composition of the liquid phase used to calculate chemical equilibrium (**in ORCHESTRA**).

This simple example directly shows the suitability of ORCHESTRA for this system. As the calcite dissolution/precipitation reaction is described kinetically, it must be excluded from the chemical equilibrium solution. As ORCHESTRA is input-based, the user only has to exclude the calcite formation from the chemical input file after which calcite dissolution/precipitation is no longer considered in the equilibrium calculation.

Note!!
Python is also a highly suitable tool for statistical analysis and managing large datasets. Apart from the option to separate reaction kinetics from equilibrium calculations, the wrapper presents additional benefits of integrating the wide variety of functions and tools presented by Python (e.g. the plotting package matplotlib or data manipulation with numpy & pandas.

2 Installing the package

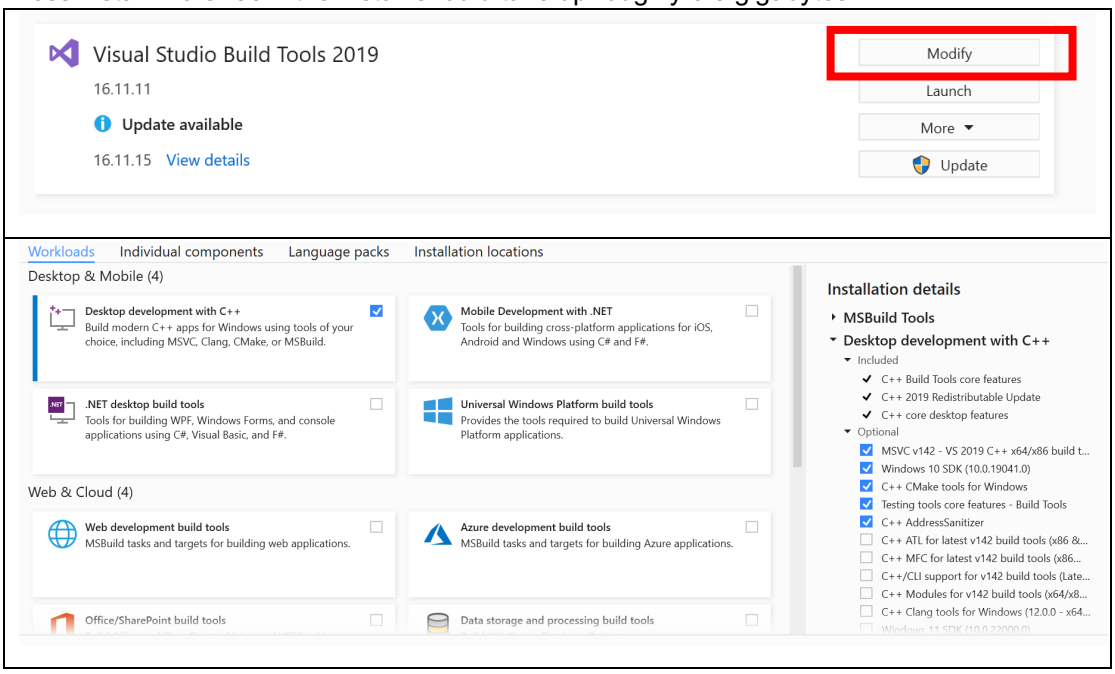
The ORCHESTRA package was written in both Java (original) and C++ (recently); the C++ version is based on Java, following the same coding structure. The binding package will be based on the C++ version, as Python is originally written in C++.

2.1 Setting up requirements

Windows:

1. Install the latest Python version (from https://www.python.org/downloads/ or using a virtual environment, e.g. Anaconda).
2. Download and install the Build Tools for Visual Studio 2022 (to allow the computer to run C++ code): https://visualstudio.microsoft.com/downloads/?q=build+tools (at the bottom of the page). When the installer is finished, open the visual studio code installer. Navigate to 'modify' and check the C++ build tools box. Make sure the following boxes are checked:
    - MSVC v142 – VS 2019 C++ x64/x86 build tools (v14…
    - Windows 10 SDK (10.0.18362.0)
    - C++ CMake tools for Windows
    - Testing tools core features – Build Tools
    - C++ AddressSanitizer (Experimental)

    Press install. **To check:** the install should take up roughly 6.8 gigabytes.



3. To run the ORCHESTRA GUI, Java must be installed. To install Java, go to your preferred webbrowser and navigate to https://www.java.com/en/download/manual.jsp. Navigate to the offline downloader ('Windows Offline (64-bit)') and follow the instructions given by the installer.
4. Lastly, install the Pybind11 package that allows binding C++ code to Python. Open the terminal (e.g. 'command prompt') and type 'pip install pybind11'. This will install the latest version of pybind11

2.2 Cloning the gitlab repository to your local directory folder

Once the prerequisites are installed, the ORCHESTRA binding/wrapper can be copied from gitlab. To do so, open a command prompt/terminal and navigate to your local directory folder where you want the binding folder to be copied. Here, for example, we navigate to a local folder called 'gitlabs'.

```
cd Documents/Guido/gitlabs
```

When you are inside the correct directory, type:

```
git clone https://gitlab.tudelft.nl/guidozeeuw/orchestra_python.git
```

This command clones the gitlab repository to your local directory. If done correctly, a folder called ‘orchestra_python’ should appear. To quickly check the presence of such a folder, type ‘dir’ (Windows) or ‘ls’ (MacOS & Linux). The folder name should be listed on your command prompt/terminal window.

The ‘orchestra_python’ folder contains one folder ‘v1.3.1’ (or a different name depending on the version). In turn, the folder ‘v1.3.1’ contains three folders:

- Install: folder containing all files to install the PyORCHESTRA package on a local device.
- Examples: various examples from this tutorial.
- ORCHESTRA: folder containing the ORCHESTRA GUI to create chemical input files.

2.3 Installing the ORCHESTRA binding package

To install the Python binding to ORCHESTRA, open terminal (‘conda prompt’) and navigate to the ‘setup_package’ directory:

```
cd v1.0.0/multiple_nodes/setup_package
```

This folder contains the ‘setup.py’ file that is used to install the package. Type the following and press ‘enter’:

```
pip install .
```

Installing the package will take roughly 1 minute, depending on the computer specifications. the command prompt/terminal should show a print result similar to: ‘successfully installed PyORCHESTRA-1.3.1’.

Congrats! The package is correctly installed to your Python environment!

3 Using PyORCHESTRA

The PyORCHESTRA submodule can be used in a Python script similar to any other Python package. First, the PyORCHESTRA submodule must be imported.

```
import numpy as np
import matplotlib.pyplot as plt
import PyORCHESTRA # here, the ORCHESTRA submodule is imported
```

To use the different functions that come with PyORCHESTRA, we assign the function class 'ORCHESTRA' to a user defined variable (here 'p').

```
p = PyORCHESTRA.ORCHESTRA() # Here, we call the function class and call it object 'p'
```

The ORCHESTRA class (now called 'p') contains 4 different functions:

- `p.initialise`(InputFile.inp, NoCells, InVars, OutVars)
- `p.updatevalues`(IN)
- `p.calculate`()
- `p.set_and_calculate`(IN)

3.1 Detailed description of functions

`p.initialise``(InputFile, NoCells, InVars, OutVars)`

INPUT

- InputFile.inp: text file; containing chemical information, obtained with ORCHESTRA GUI.
- NoCells: int; number specifying the number of cells of the given system.
- InVars: matrix; array listing the variable names that are used as input (e.g. 'pH' or 'CO3-2.tot'). Must have the same length as defined NoCells.
- OutVars: matrix; array listing the variable names that are defined as output (e.g. 'pH' or 'CO3-2.con').

OUTPUT

None

FUNCTION DESCRIPTION

Function that takes the user defined chemical system boundaries (chemical system, number of cells) as input and constructs a so-called 'calculator'-object around this information. This information is now stored in this 'p' variable and can be used in further calculations.

`p.updatevalues``(IN)`

INPUT

- IN: matrix; containing an array for every cell which is used to update the bulk-composition of the system for which equilibrium is calculated.

OUTPUT

None

FUNCTION DESCRIPTION

Function takes an array (or matrix for multidimensional systems) as input containing values for the variables listed in InVars (see p.initialise).

p.calculate()

INPUT
None

OUTPUT
- Matrix containing an array for every cell with the same length as Outvars containing the output values specified in Outvars.

FUNCTION DESCRIPTION
This function calculates the equilibrium for a system with given bulk composition (initialised with p.updatevalues) and returns an array containing the results for variables specified in OutVars (see p.initialise)

p.set_and_calcultate(IN)

INPUT
- IN: matrix; array for every cell which is used to update the bulk-composition of the system for which equilibrium is calculated.

OUTPUT
- Array with the same length Outvars with output values.

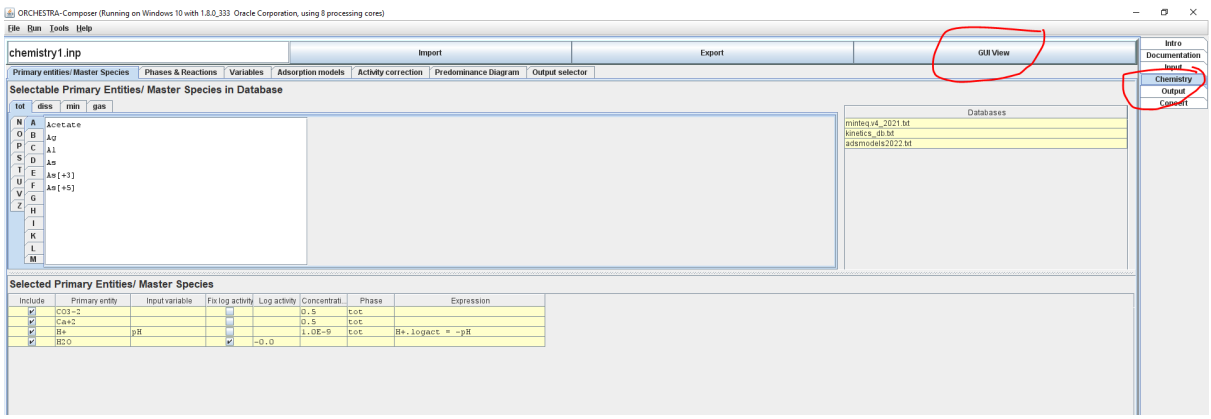
FUNCTION DESCRIPTION
A combination of the functions p.updatevalues and p.calculate. Takes an array with values as input containing a given 'bulk composition' for the variables specified in InVars (see p.initialise). The function directly calculates the equilibrium and returns an array containing the result values for the variables specified in OutVars (see p.initialise).

4 ORCHESTRA tutorial: Chemistry input file

To calculate chemical equilibrium with ORCHESTRA, a chemical input file must be defined. This input file contains the following information:

- Variable definitions
- Phase hierarchy
- Master species
- Chemical reactions (including adsorption models)

For most cases, the chemistry file can be fully constructed with the ORCHESTRA GUI. However, in some cases, this is not the case and manually adding information to the text file is sometimes required. Therefore, a walkthrough of the input file is presented here. This walkthrough discusses the input file from top to bottom!



4.1 Setting up ORCHESTRA chemical input file with the GUI

Manually writing out the full chemistry input file can be a time-consuming task. The GUI can therefore be used to interactively create the input file. This includes:

- Specifying primary entities (section 4.6) excluding the special case discussed in section 4.9
- Setting up variables (section 4.4)
- Choosing the reaction network (section 4.7) excluding the special case discussed in section 4.8. The reaction network used is based on the used database. ORCHESTRA comes standard with the minteq database. The reader is invited to check this file and see how the individual reactions are defined.

An example of how the ORCHESTRA GUI is used is available in the example in section 5.1

4.2 Version and general commands

At the top of the chemistry input file, the user finds the version date. Here, the user may also alter some internal settings:

```
//******** Version: dd/mm/yyyy hh:mm *****
@NrDigits: 12 // number of decimal places in reaction coefficients
//@rewriteReactions: //indicates that reactions are rewritten in terms of primary
entities
```

- The '//' syntax is used to add comments.

4.3 Databases

In this part, the user specifies the databases that will be included in the chemistry calculation. These can either be local files or files from the internet. Here, we include the minteq database:

```
//******** The database file(s) *****
//Syntax for local files:       @database: minteq.txt
//Syntax for files on internet: @database: www.meeussen.nl/orchestra/minteqv4.txt
```

```
//******** end of thedatabase file(s) *****
```

Below this section, the user can specify additional parts of code by using the '@class:' command. These extra pieces of code will not be changed by the GUI.

```
//******** extra text 0 *****
@class: extra_text_0(){
  @stop:
  @scan: objects2022.txt
}
@extra_text_0()
//******** end of extra text 0 *****
```

Adding '@stop:' commands ORCHESTRA to create an 'iteration.dat' file when the program fails while calculating equilibrium. This file can be used as a rough results log file to examine why the calculation fails.

The '@include:' command is used to include local files. These local files may contain user-defined ORCHESTRA expressions. In this case, we include the 'objects2022.txt' file. This is a file that is standardly available in the ORCHESTRA download folder. This file contains fundamental calculation definitions used by ORCHESTRA. It is therefore advised to always include this file in the chemistry input file.

4.4 Variables

The user may specify a custom variable. A default value must be given (otherwise ORCHESTRA will raise an error). **Note**: the PyORCHESTRA wrapper allows the user to overwrite these variables from within Python! By doing so, the default value can be overwritten.

```
//******** The variables *****
//general structure: @Var: [VARIABLE_NAME] [DEFAULT_VALUE]
@Var: porosity .9 //porosity of the system
@Var: saturation 1.0 //saturation of voids
@Var: totvolume 1.0 //dm3
@Var: my_own_variable_1 382.24
@Var: my_own_variable_2 -3.0
//******** End of the variables *****
```

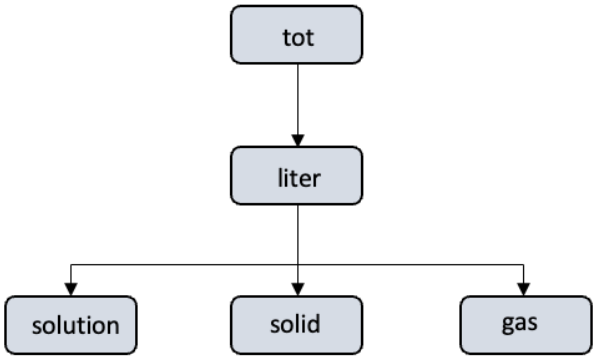
Below the variables, there is the possibility to add extra text. This extra text can be used to relate variables to each other. For example, here we give an expression for 'my_own_variable_1' and 'my_own_variable_2'. For this, use the '@Calc:' syntax.

```
//******** Extra text 1 *****
@class: extra_text_1(){
  @Calc: (1, "my_own_variable_1 = porosity*totvolume*saturation") //total water volume
  @Calc: (1, "my_own_variable_2 = porosity*(1-saturation)") //total gas volume
}
@extra_text_1()
//******** End of Extra text 1 *****
```

Mathematical expressions must be between the " "-syntax.

4.5 Phase Hierarchy

The standard ORCHESTRA file comes with a pre-defined phase hierarchy:



The table below indicates how the different phases relate to the given species CO_3^{-2}.

$\boldsymbol{CO_3^{-2}.tot}$	The total amount of CO_3^{-2} in the system (in $\frac{mol}{L}$). Also known as the system *bulk composition*.
$\boldsymbol{CO_3^{-2}.liter}$	The share of CO_3^{-2} in the liter phase (in $\frac{mol}{L}$).
$\boldsymbol{CO_3^{-2}.solution}$	The share of CO_3^{-2} in solution (in $\frac{mol}{L}$).
$\boldsymbol{CO_3^{-2}.solid}$	The share of CO_3^{-2} in solids (in $\frac{mol}{L}$).
$\boldsymbol{CO_3^{-2}.gas}$	The share of CO_3^{-2} in gas (in $\frac{mol}{L}$).

Note, that the *liter* phase contains all of a certain species in the combined *solution*, *solid* and *gas* phase. This has no physical meaning other than to make sure that all three phases are linked together. In turn, the solution, *solid* and *gas* phase could be further subdivided into smaller phases. This could be useful for systems with multiple liquids (subdivide *solution* in liquid 1 and liquid 2) or with multiple solid phases (e.g. different minerals).

The hierarchy is listed in the chemical input file. For example, here we describe how the *liter* phase is linked to the *tot* phase:

1. Start by defining a *tot* phase.
2. Then define the *liter* phase
3. Link *liter* to *tot* using the @link_phase command.
   - The first argument is the phase that needs to be linked.
   - The second argument is the parent phase to which the new phase is linked.
   - The last argument determines the unit of conversion. Here the *liter* phase is the watervolume (=my_variable_1 in section 4.4). For example, let's say we have a species X with a total of 1 mol/L (X.tot = 1 mol/L) of which 20% is present in the liquid. In addition to this, the porosity and saturation of the system are 0.4 and 0.9 respectively. The share of X.tot in the liquid is then 1*20% = 0.2 mol/L. **But** the volume of the liquid is itself 1*0.4*0.9 = 0.36 mol/L. Therefore the concentration in the liter phase (X.liter) is calculated as 0.2/(0.36) = 0.556 mol/L.

```
@phase(tot)
@phase(liter)
@link_phase(liter, tot, “my_variable_1”)
```

In a similar matter, we link the *solution* phase to the *liter* phase:

1. Define the *solution* phase. (Write also the definition of solid and gas)
2. Link *solution, solid*, and *gas* to *liter*. A conversion unit of 1 indicates that solution has the same unit. In this case, this implies that all different phases are part of the water volume phase.

```
@phase(solution)
@link_phase(solution, liter, “1”)
@phase(solid)
```

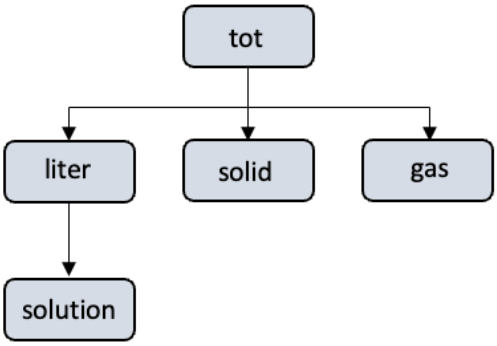
```
@link_phase(solid, liter, “1”)
@phase(gas)
@link_phase(gas, liter, “1”)
```

4.5.1 Custom hierarchy

In the previous hierarchy all *gas, solid* and *solution* phases are part of the *liter* (or water volume) phase, indicated by the shared conversion unit “1”. However, this introduces limitations for systems where one wants to differentiate between phases not part of the *liter* phase. Luckily, the user is free to choose any type of hierarchy. For example, here we define a phase hierarchy where the *solution, gas* and *solid* phase do not share the same conversion unit.

```
@phase(tot)
@phase(liter)
@link_phase(liter, tot, “porosity*saturation”)
@phase(solution)
@link_phase(solution, liter, “1”)
@phase(solid)
@link_phase(solid, tot, “totvolume*(1-porosity)”)
@phase(gas)
@link_phase(gas, tot, “porosity*(1-saturation)”)
```

This notation describes the following hierarchy:



4.5.2 .con & .logact

The different phases described above list the bulk composition of a given species in the different phases. However, it is important to note that $CO_3^{-2}.solution$ **is not** the concentration of $[CO_3^{-2}]$ in the solution. It is rather the sum of CO_3^{-2} in the solution shared over all the different species (= the bulk composition):

$$CO_3^{-2}.solution = [CO_3^{-2}]_{in\ CO_3^{-2}} + [CO_3^{-2}]_{in\ H_2CO_3} + [CO_3^{-2}]_{in\ HCO_3^-} + [CO_3^{-2}]_{in\ CaCO_3} + \ldots$$

Or, mathematically expressed:

$$CO_3^{-2}.solution = \sum_{i=0}^{j} [CO_3^{-2}]_i$$

Where j is the total amount of species and $[CO_3^{-2}]_i$ is the total amount of CO_3^{-2} in the i^{th} species. The concentration and molar activities of a given species is obtained with by using the syntax ‘.con’ (for concentration) or ‘.logact’ (for molar activity) next to the species name. **Note:** The molar activity is given on a $\log_{10}$ scale!

4.6 Primary entities

Now the user must specify the primary entities (or master species) that ORCHESTRA will use to calculate chemical equilibrium. The primary entities can be seen as the building blocks needed to build all the components (or ‘normal’ entities) of interest. As an example, the dissolution of calcite in water requires 4 different building blocks:

- Calcium (Ca^{+2})
- Carbonate (CO_3^{-2})
- Water (H_2O)
- Protons (H^+)

With these 4 primary entities, all the other entities are made:

- Ca^{+2}
- CO_3^{-2}
- $CaCO_3$ (calcite)
- HCO_3^-
- H_2CO_3
- H^+
- OH^-
- O_2
- CO_2
- $CaCO_3$
- $CaHCO_3^+$
- $CaOH^+$
- $Aragonite$
- $Lime$
- $Portlandite$

The primary entities are defined after the phase hierarchy. To include a species as primary entity, the user must include 2 lines.

- One line that adds the species/entity to ORCHESTRA. **But note** that ORCHESTRA does not know whether this is a primary entity or not based on this line. Therefore;
- A separate line must be included that defines this entity as 'primary_entity'.

The following sections describe how different species (elements, compounds, minerals and gases) are included as primary entity.

4.6.1 Set element as primary entity

The following lines describe how elements (H, N, O, S, Ca) are included as primary entity:

```
@entity(O, tot, 0) //add oxygen (O) as entity
@primary_entity(O, -9.0, tot, 1.0e-9) //set oxygen as primary entity
```

The @entity command adds an element as entity. This requires 3 input variables:

- The element name
- The phase to which it belongs (here: 'tot')
- A value (Hans? What is this value)

The @primary_entity command sets a species as 'primary'. This requires 4 input variables:

- The element name which will be set to 'primary'
- A value that ORCHESTRA uses as initial value to calculate equilibrium (always set to -9.0)
- The phase in which their quantity unit is expressed as input (here: 'tot'). This can also be the case if the entity itself is part of a different phase (for example: @entity(O, solution, 0)). Now, the input quantity is expressed in 'tot' (total in whole system). ORCHESTRA automatically converts this to the unit of its actual phase ('solution').
- Information on how much mol/L of the element is available (here 1.0e-9 mol/L).

4.6.2 Set chemical compound and ions as primary entity

Apart from elements, building blocks may also consist of chemical compounds (CO_3^-, NH_4^+) or ions (Ca^{+2}, H^+). For this, the syntax @species is used:

```
@species(CO3-2, -2) //add carbonate (CO3-2) as entity
@primary_entity(CO3-2, -9.0, tot, 1.0e-9) //set carbonate as primary entity
```

The @species command holds two arguments:

- The compound/ion name.
- The charge of the compound/ion

4.6.3 Set minerals as primary entity

Minerals are added using the @mineral command:

```
@mineral(Gypsum[s]) //add gypsum as entity
@primary_entity(Gypsum[s], -9.0, tot, 1.0e-9) //set gypsum as primary entity
```

Note, that ORCHESTRA itself does not contain information on the elemental composition of (e.g.) Gypsum. For this, either a database is required (see Chapter 4.3) or the method described in Section 4.9 must be applied.

4.6.4 Set gas as primary entity

Similar to minerals, gases can be added as primary_entity with the @gas syntax:

```
@gas(CH4[g]) //add gypsum as entity
@primary_entity(CH4[g], -9.0, tot, 1.0e-9) //set gypsum as primary entity
```

4.6.5 Fixed logactivity for a primary entity

Let us consider the primary entity H^+. Based on the previous chapters, this can be added with:

```
@species(H^+, 1) //add proton as entity
@primary_entity(H+, -9.0, tot, 1.0e-9) //set proton as primary entity
```

According to Chapter 4.6, the input quantity of H^+ (here: $H^+.tot = 1.0e^{-9}\frac{mol}{L}$) is not the actual amount of free H^+ in the system. In fact, it is the total amount of H^+ available in the system for all compounds that contain H^+. This might introduce some problems for cases where the user wants to base equilibrium on a fixed pH; here, this requires the system to user the amount of free H^+ in the system as: $[H^+] = 10^{-pH}$. The question now is: *how do we set the amount of free H^+ as input, rather than the total amount of H^+ in the system?*

To do this, the user can relate the H^+ entity to a fixed log activity. In simple words: by doing this the user 'tells' ORCHESTRA what the amount of free H^+ in solution should be at equilibrium. Based on this log activity the concentrations of other components are determined.

For a given species "my_own_species+2", this is included as follows:

```
@species(my_own_species+2, 2) //add proton as entity
@primary_entity(my_own_species+2, 7.0) //set proton as primary entity
```

Here the @primary_entity syntax contains 2 arguments:

- The name of the entity/species
- The "fixed" log activity of the species

4.7 Equilibrium reactions

The chemical input file concludes with specifying the equilibrium reactions that ORCHESTRA should take into account. In ORCHESTRA, a "reaction" object is used to form new entities from existing entities. For example:

```
@species(HCO3-1, -1) //add bicarbonate as entity
@reaction(HCO3-1, 3.455, 1, H+, 1, CO3-2) //set proton as primary entity
```

Here, 3.455 is the K-constant of the reaction
Alternatively, you can use the log K version of the reaction:

```
@species(HCO3-1, -1) //add bicarbonate as entity
@logKreaction(HCO3-1, 3.7, 1, H+, 1, CO3-2) //set proton as primary entity
```

Both logKreactions and reactions can be mixed in a single input file.

The example above presents a very simple reaction, where:

$$CO_3^{-2} + H^+ \leftrightarrow HCO_3^-$$

But what if we have a more complex reaction, for example:

$$CaOH^+ + H^+ \leftrightarrow Ca^{+2} + H_2O$$

Here, the new species ($CaOH^+$) is formed, along with an already defined entity (H^+). The following syntax is used:

```
@species(CaOH+, 1) //add calcium hydroxide as entity
@logKreaction(CaOH+, -12.679, 1, Ca+2, 1, H2O, -1, H+) //set proton as primary entity
```

Where:

- -12.679 is the log K constant of the reaction (this is listed in the minteq database)
- "1", "1", "-1" are the stoichiometry coefficients of Ca^{+2}, H_2O and H^+, respectively.

4.8 Special case: mass-action and mass-balance

In the examples above, the coefficients indicate their share in the mass balance for the given reaction: "1, Ca+2" implies that 1 CaOH+ contains 1 Ca^{+2}. This is called the *mass-balance* coefficient. ORCHESTRA also uses information on the *mass-action* coefficient for a given species. By separating mass-action and mass-balance coefficients, we can use different coefficients for the reaction balance calculation and the calculation of the masses of the species involved.

This is further illustrated with a the syntax of a logKreaction for the adsorption of NH_4^+ to a body surface following the Freundlich equation:

$$\log\left(q_{NH_4^+}\right) = K_d + n \cdot \log\left(a_{NH_4^+}\right)$$

Where $q_{NH_4^+}$ is the concentration of NH_4^+ adsorbed to the body surface, K_d is the Freundlich constant, n is a coefficient and $a_{NH_4^+}$ is the molar activity of NH_4^+ in solution.

n describes the shape of the curve, where a linear relationship is found for $n = 1$. For $n = 1$, the reaction is defined in ORCHESTRA below. Here it is assumed that the 'ads' phase and the NH_4^+ entity are defined earlier in the text file (see Chapter 4.5 & 4.6):

```
@species(NH4_ads, ads, 1) //add species to file and relate the species to phase 'ads'
@logKreaction(NH4_ads, [Kd], 1, NH4+) //relate new species to NH4+ in solution
```

As $n = 1$, both the mass-action and mass-balance coefficients are 1; $NH_{4,ads}^+$ counts for 1 in the mass-balance of NH_4^+. However, if a non-linear relation is found between NH_4^+ and $NH_{4,ads}^+$, a different mass-action coefficient can be used. For example, adopting $n = 0.8$ leads to the following input.

```
@species(NH4_ads, ads, 1) //add species to file and relate the species to phase 'ads'
@logKreaction(NH4_ads, [Kd], "0.8, 1", NH4+) //relate new species to NH4+ in solution
```

Here, the "1" is replaced with "0.8, 1", where 0.8 is the mass-action and 1 is the mass-balance coefficient.

4.9 Special case: custom primary entity

In the previous chapter 'NH4_ads' is a user defined entity where its composition is based on the logKreaction. In other words, 'NH4_ads' is an entity that is based on primary entities. However, the same method cannot be applied to defining custom primary entities; primary entities cannot be based on other primary entities following a logKreaction. Let us consider a system where our main building blocks (primary entities) are CO_3, Ca^{+2} and NH_4^+. Following section 4.6, these are included following:

```
@species(CO3-2, -2)
@primary_entity(CO3-2, -9.0, tot, 1e-9)
@species(Ca+2, 2)
@primary_entity(Ca+2, -9.0, tot, 1e-9)
@species(NH4+, 1)
@primary_entity(NH4+, -9.0, tot, 1e-9)
```

We also want to include a custom compound called 'tutorial_compound' with a composition $Ca(CO_3)_2NH_4$ as a primary entity:

```
@entity(tutorial_compound, tot, 1) //add new mineral as entity
@primary_entity(tutorial_compound, -9.0, tot, 1e-9) // set as primary entity
```

While we have successfully added the compound as a primary entity, ORCHESTRA does not know the composition of the compound (as we have not specified this anywhere!). The following setup, however, allows us to specify its composition in a separate line:

1. Open a new text file.
2. Add the following lines:

```
@class: composition(name, c1, n1){
   @link(<name>, <n1>, 0, <c1>)
}

@class: composition(name, c1, n1, c2, n2){
   @link(<name>, <n1>, 0, <c1>)
   @link(<name>, <n2>, 0, <c2>)
}

@class: composition(name, c1, n1, c2, n2, c3, n3){
   @link(<name>, <n1>, 0, <c1>)
   @link(<name>, <n2>, 0, <c2>)
   @link(<name>, <n3>, 0, <c3>)
}

@class: composition(name, c1, n1, c2, n2, c3, n3, c4, n4){
  @link(<name>, <n1>, 0, <c1>)
   @link(<name>, <n2>, 0, <c2>)
   @link(<name>, <n3>, 0, <c3>)
   @link(<name>, <n4>, 0, <c4>)
}
```

3. Save the file in the ORCHESTRA directory and give it an appropriate name (here: 'kinetics_db.txt').
4. Include the file with the '@scan command as described in Section 4.3.

```
//******** extra text 0 *****
@class: extra_text_0(){
      @stop:
      @scan: objects2022.txt
      @scan: kinetics_db.txt
}
@extra_text_0()
//******** end of extra text 0 *****
```

With the previous steps, a function called '@composition' is formulated that allows the user to define a composition for a given entity. The '@composition' command is used in the extra text section below the primary entity section:

```
//******** Extra text 2a *****
@class: extra_text_2a(){
  @composition(tutorial_compound, 1.0, Ca+2, 2.0, CO3-2, 1.0, NH4+)
}
@extra_text_2a()
```

As we have provided ‘tutorial_compound’ with a composition, we can use it in logKreactions similarly to other primary entities.

5 Examples

5.1 The $Ca^{+2} - CO_3^{-2}$ system

The first example is a simple case of the $Ca^{+2} - CO_3^{-2}$ system. The example calculates chemical speciation at equilibrium at different pH levels, given a fixed bulk composition of Ca^{+2} and CO_3^{-}. The example represents a batch experiment of a closed system where the pH is fully controlled. **Important:** We assume that the system has infinite time to equilibrate. By doing so, no rate dependencies on mineral growth are introduced.

5.1.1 Setting up the chemistry input file

Before calculating chemical equilibrium, the user is required to construct a chemistry input file. This input file contains all the chemical information needed for ORCHESTRA to build a calculator object. The chemistry input file is constructed using the ORCHESTRA GUI. The GUI comes with the gitlab repository. Opening the GUI leads to the following screen.

On the right (or top when working on a windows machine), there are various tabs available. In general, only the 'chemistry' tab is needed.

Step 1: specify primary components

The ORCHESTRA chemical input file should contain information on what the primary species/master species for the given calculation are. Primary species can be described as the elements describing the *bulk composition*. These can either be elements (H, N, O, C, etc.) or chemical components and ions (CO_3^-, NH_3, SO_4^{-2}, H^+). In this example, we will calculate chemical equilibrium using chemical components and ions. In this case, the main components (or *building blocks*) are Ca^{+2}, CO_3^{-2} and H^+. Do the following:

1. Navigate to 'chemistry' → 'Primary entities/Master species'.
2. Add Ca^{+2}, CO_3^{-2} & H^+ for the 'diss' tab.
3. Lastly, we will add water to the system. Go to the 'liter' tab and select H_2O.

Step 2: fix log activities if necessary

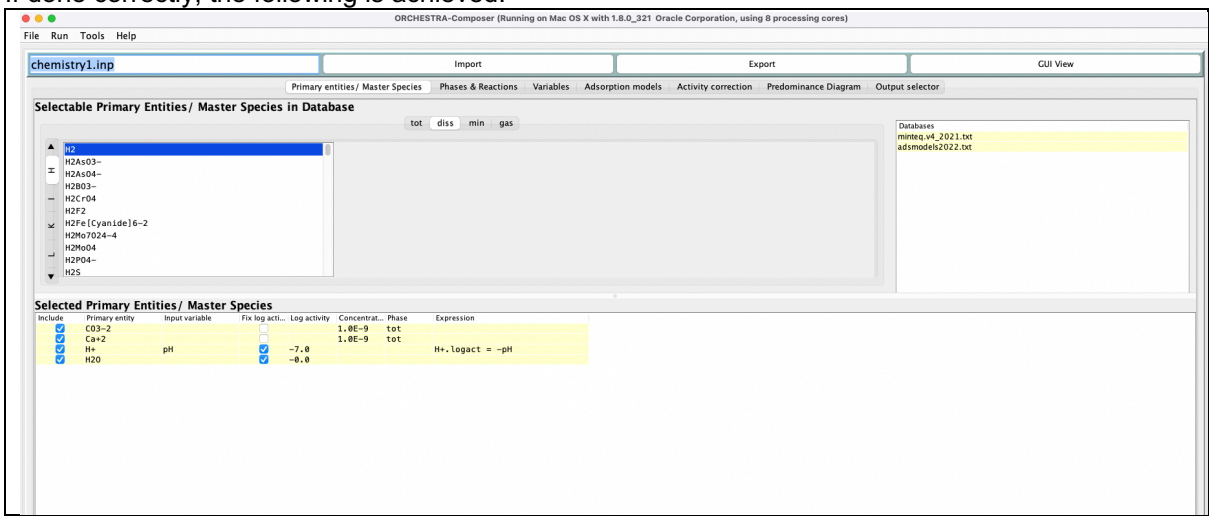
In most-simple words, ORCHESTRA works by iteratively converging to a solution where equilibrium is met. To do so, the program searches for the log activities at equilibrium of the different species. Now, the user has 2 options: (1) either **fix** the log activity of a master species, or (2) allow the log activity to **change** during the search for chemical equilibrium. For example, in this example we want to compare chemical speciation at different pH levels. Therefore, the pH must stay at a fixed level within every individual calculation; if we want to calculate chemical equilibrium at a pH of (e.g.) 7, we don't want

the system to allow a change in log activity of the H^+. Therefore, the user must specify that the pH is fixed.

In this example the log activities are fixed for both H^+ and H_2O. To do so:

1. For the H_2O species, select 'fix log activity' and set the value to -0.0.
2. Do the same for H^+: select 'fix log activity'.

If done correctly, the following is achieved:



Don't change the values in the column 'concentrations'. You can, but this does not matter as this will be done later in Python as well.

Step 3: select reaction network

The next step is the selection of the reaction network (in the Phases & Reactions tab). This is an important step that requires the user to critically think about the problem at hand. If a reaction is selected, ORCHESTRA takes this reaction into account when calculating chemical equilibrium. In this example, we assumed that the system calculates equilibrium at $t = \infty$. Therefore, all reactions need to be considered.

1. Navigate to the Phases & Reactions tab.
2. Check all reactions.

Include	Name	Log K (25C)	Phase	Coefficient	Reactant	Coefficient	Reactant	Coefficient	Reactant	Coefficient	Reac
☑	Alkalinity	0.0	tot	1.0	CO3-2						
☑	Aragonite[s]	8.300000	min	1.0	CO3-2	1.0	Ca+2				
☑	C	0.0	tot	1.0	CO3-2						
☑	CO2[g]	18.14700	gas	1.0	CO3-2	2.0	H+	-1.0	H2O		
☑	C[+4]	0.0	tot	1.0	CO3-2						
☑	Ca	0.0	tot	1.0	Ca+2						
☑	CaCO3	3.200000	diss	1.0	CO3-2	1.0	Ca+2				
☑	CaHCO3+	11.59900	diss	1.0	CO3-2	1.0	Ca+2	1.0	H+		
☑	CaOH+	-12.69700	diss	1.0	Ca+2	-1.0	H+	1.0	H2O		
☑	Calcite[s]	8.480000	min	1.0	CO3-2	1.0	Ca+2				
☑	H	0.0	tot	1.0	H[+1]						
☑	H2CO3	16.68100	diss	1.0	CO3-2	2.0	H+				
☑	HCO3-	10.32900	diss	1.0	CO3-2	1.0	H+				
☑	H[+1]	0.0	tot	1.0	H+						
☑	Lime[s]	-32.69930	min	1.0	Ca+2	-2.0	H+	1.0	H2O		
☑	O	0.0	tot	1.0	O[-2]						
☑	OH-	-13.99700	diss	-1.0	H+	1.0	H2O				
☑	O[-2]	0.0	tot	1.0	H2O						
☑	Portlandite[s]	-22.80400	min	1.0	Ca+2	-2.0	H+	2.0	H2O		

Step 4: export chemistry text file

1. The chemistry setup is finalized! Export the chemistry file by selecting 'Export' at the top.
2. A text file called 'chemistry1.inp' is created in the ORCHESTRA folder.

5.1.2 Calculating chemistry with Python.

For this example, we will examine what the effects of pH are on the system composition at equilibrium. In Python, the pH is changed iteratively. For every pH, a new equilibrium composition is calculated using ORCHESTRA. In turn, the results are plotted using Python.

Step 1: create .py file and import modules.

1. Open a new python file and call it 'carbonate_species.py'

2. Start the file by importing the required submodules:

```
import numpy as np
import matplotlib.pyplot as plt
import PyORCHESTRA # here, the ORCHESTRA submodule is imported
```

Step 2: get the ORCHESTRA function class 'p'.

```
p = PyORCHESTRA.ORCHESTRA()
```

Step 3: Initialise the ORCHESTRA calculator.

Now, the user must define the boundary conditions of the system for which chemical equilibrium will be calculated.

1. Name the chemistry text file. InputFile = 'chemistry1.inp'
2. Specify the number of cells. Here we will consider a 0-D system.
3. Specify the variables that will be used as input. Here, this includes the bulk composition (total Ca^{+2} and CO_3^{-2} in the system) and pH. In addition to this, ORCHESTRA comes with pre-available parameters such as porosity and saturation.
4. Specify the variables that the user wants as output. Here, this includes the concentration of several important species (e.g. $CaCO_3$ and CO_3^{-2}).
5. Call the function 'initialise' to initialize the ORCHESTRA calculator object.

```
InputFile = 'chemistry1.inp'
NoCells = 1 #only 1 cell to have a 0-D system
InVars = np.array(['Ca+2.tot', 'CO3-2.tot', 'pH', 'porosity', 'saturation'])
OutVars = np.array(['CaCO3.con', 'CO3-2.con', 'HCO3-.con', 'H2CO3.con', 'Ca+2.con',
                    'Calcite[s].min'])

p.initialise(InputFile, NoCells, InVars, OutVars)
```

Step 4: Set the initial conditions.

The 'p'-object now contains an initialized system that is ready to be used. By specifying values for the given input parameters (in InVar), an equilibrium will be calculated.

1. First, we create the variable matrix 'IN' containing arrays of the same length as InVars (1 value for every parameter must be given).
2. Using the `np.where` function, we can change the value in IN at the position corresponding to that parameter.
3. We will assume that the system is fully saturated, with a porosity of 1. The bulk composition is set to 0.5 $\frac{mol}{L}$ Ca^{+2} and 0.5 $\frac{mol}{L}$ CO_3^{-2}.

```
IN = np.array([np.ones_like(InVars)]).astype(float)

IN[0][np.where(InVars == 'porosity')] = 1
IN[0][np.where(InVars == 'saturation')] = 1
IN[0][np.where(InVars == 'Ca+2.tot')] = 0.5
IN[0][np.where(InVars == 'CO3-2.tot')] = 0.5
```

Step 5: change pH and get equilibrium

The pH is increased from 2 to 12 with a stepsize of 0.001. For every new 'configuration', system equilibrium is calculated using 'set_and_calculate'. When ORCHESTRA calculates equilibrium, it returns a matrix with values for the parameters listed in OutVars.

1. Define range for pH (here, using `np.arange`).
2. Do a simple for loop, iterating over the different pH's.
3. Assign the new pH-value to the correct position in InVars.
4. Calculate equilibrium with p.set_and_calculate(IN)
5. Assign output values to the correct lists.

```
pH_list = np.arange(2,12,0.001) # range of pH

# define output lists
CaCO3, CO3, HCO3, H2CO3, Ca, calcite = [], [], [], [], [], []
```

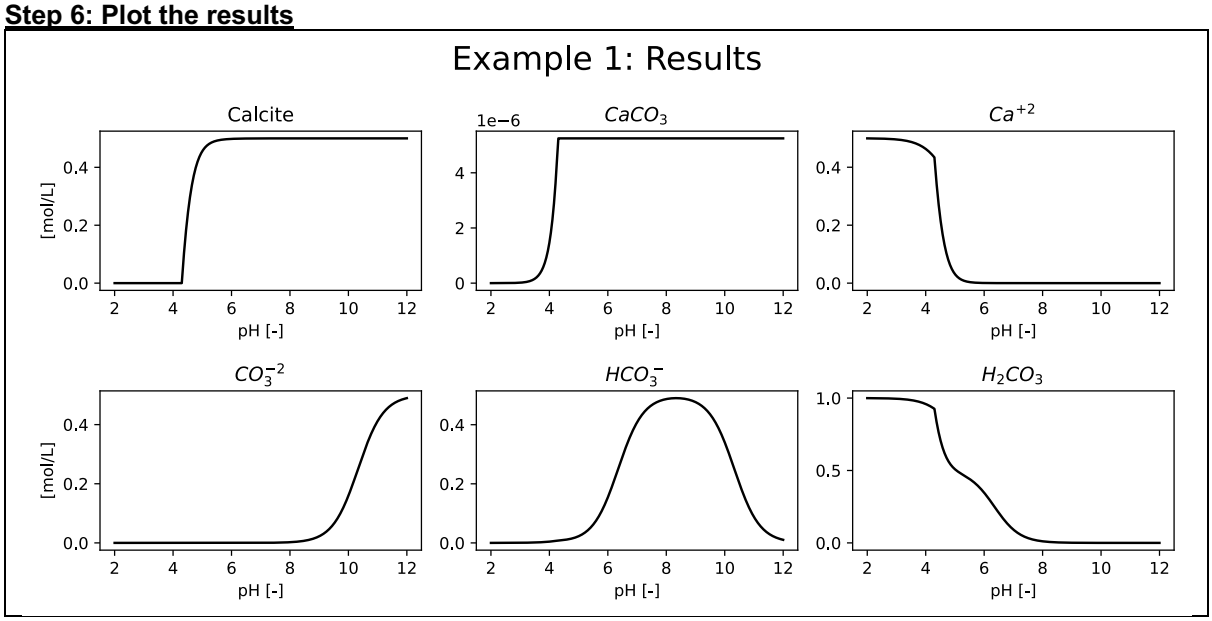
```
# loop over every pH
for pH in pH_list:

   # change pH
   IN[0][np.where(InVars == 'pH')] = pH

   # run ORCHESTRA
   OUT = p.set_and_calculate(IN)

   # get output
   CaCO3.append(OUT[0][np.where(OutVars == 'CaCO3.con')])
   CO3.append(OUT[0][np.where(OutVars == 'CO3-2.con')])
   HCO3.append(OUT[0][np.where(OutVars == 'HCO3-.con')])
   H2CO3.append(OUT[0][np.where(OutVars == 'H2CO3.con')])
   Ca.append(OUT[0][np.where(OutVars == 'Ca+2.con')])
   calcite. append(OUT[0][np.where(OutVars == 'Calcite[s].min')])
```

Step 6: Plot the results



5.2 Calcite dissolution

In the previous example, ORCHESTRA calculated the calcite concentration based on a given composition of Ca^{+2}, CO_3^{-2} and H^+. Now, we will do the opposite: "what would happen to the pH and carbonate concentrations if we dissolve calcite in the solution?". For this, we will start with an initial concentration of calcite (0.5 mol/L). We dissolve calcite by iteratively decreasing the calcite mass (in Python), which will release Ca^{+2} and CO_3^{-2} in the solution, thus affecting the pH and concentrations. **It is assumed that no CO_3^{-2} and Ca^{+2} is initially present in the liquid!** To calculate the pH, ORCHESTRA is used. The following primary species ('*building blocks*') are used:

- $CO_3^{-2}.tot$
- $Ca^{+2}.tot$
- $H^+.tot$
- $calcite\_mineral.tot$

$calcite\_mineral.tot$ is the total amount of calcite ($CaCO_3[s]$) in the system.

To calculate chemical equilibrium with ORCHESTRA, a bookkeeping "trick" is used that allows ORCHESTRA to know how much CO_3^{-2} and Ca^{+2} is available in the liquid based on the amount of calcite present. This trick is based around understanding the following equations:

$$CO_3^{-2}.tot_{liquid} = CO_3^{-2}.tot - CO_3^{-2}.tot_{calcite}$$
$$Ca^{+2}.tot_{liquid} = Ca^{+2}.tot - Ca^{+2}.tot_{calcite}$$

Let us, for example consider the case where (1) no calcite has been dissolved yet. As we assume that no free CO_3^{-2} and Ca^{+2} is initially present in the liquid, the following totals are applicable:

- $calcite\_mineral.tot = 0.5\ mol/L$
- $CO_3^{-2}.tot = 0.5\ mol/L$
- $Ca^{+2}.tot = 0.5\ mol/L$

By defining the system this way, ORCHESTRA sees that all CO_3^{-2} and Ca^{+2} is present in calcite and therefore none is available for the other species in the liquid ($[CO_3^{-2}]$, $[HCO_3^-]$, $[H_2CO_3]$ & $[Ca^{+2}]$ are all 0).

Now, we take the case where 0.1 mol of calcite has been dissolved:

- $calcite\_mineral.tot = 0.4\ mol/L$
- $CO_3^{-2}.tot = 0.5\ mol/L$
- $Ca^{+2}.tot = 0.5\ mol/L$

As you can see, the total amount of CO_3^{-2} and Ca^{+2} remain the same, but the total amount of calcite in the system changes. As the total amount of CO_3^{-2} and Ca^{+2} exceed those of $calcite\_mineral$, ORCHESTRA knows that there must be 0.1 mol/L CO_3^{-2} and Ca^{+2} available to calculate equilibrium in the liquid (as $0.4\ mol/L$ is part of calcite_mineral).

Now follows a step-by-step walkthrough on how to define this system using ORCHESTRA.

5.2.1 Setting up the chemistry input file

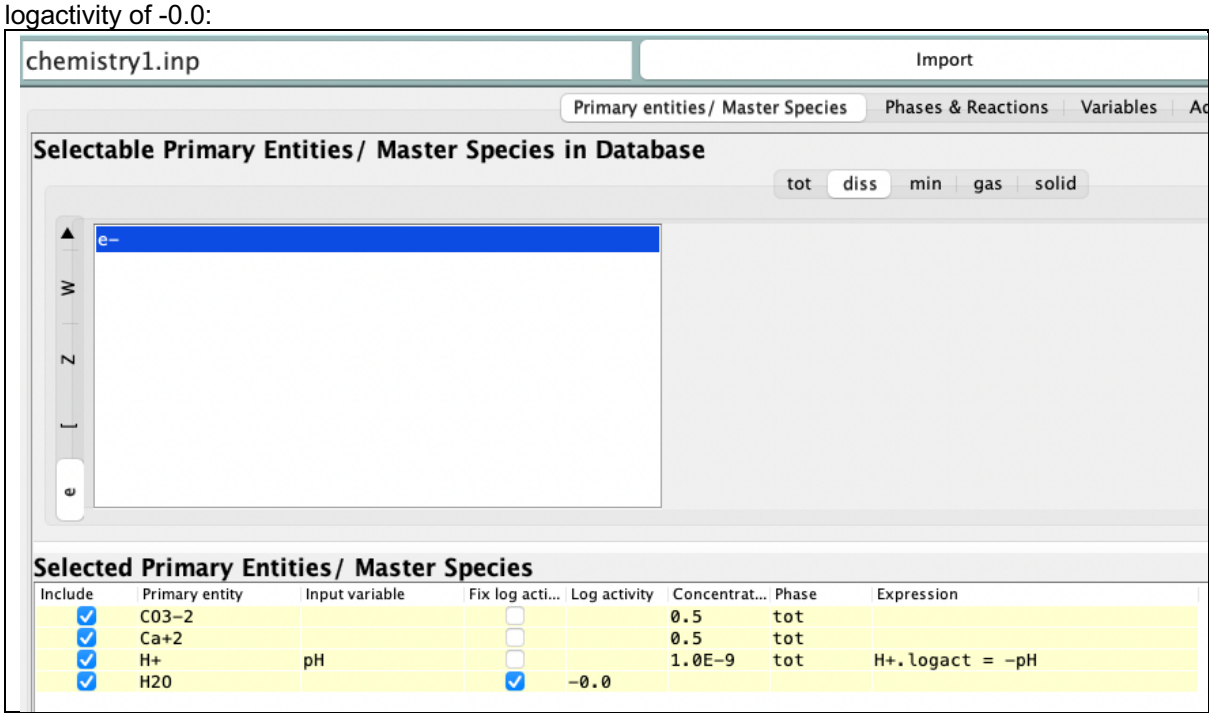
Step 1: specify primary components

The ORCHESTRA chemical input file should contain information on what the primary species/master species for the given calculation are. Primary species can be described as the elements describing the *bulk composition*. These can either be elements (H, N, O, C, etc.) or chemical components and ions (CO_3^-, NH_3, SO_4^{-2}, H^+). In this example, we will calculate chemical equilibrium using chemical components and ions. In this case, the main components (or *building blocks*) are Ca^{+2}, CO_3^{-2} and H^+. Do the following:

1. Navigate to 'chemistry' → 'Primary entities/Master species'.
2. Add Ca^{+2}, CO_3^{-2} & H^+ for the 'diss' tab.
3. Lastly, we will add water to the system. Go to the 'liter' tab and select H_2O.

Step 2: set logactivities

In the previous example, the pH was an **input** parameter where we needed to set the logactivity to a fixed value (constant in equilibrium calculation). However, for this case, we want the relate the pH to the dissolution of calcite. For this, we need the pH to be the **output**. Therefore, set only H_2O to a fixed logactivity of -0.0:

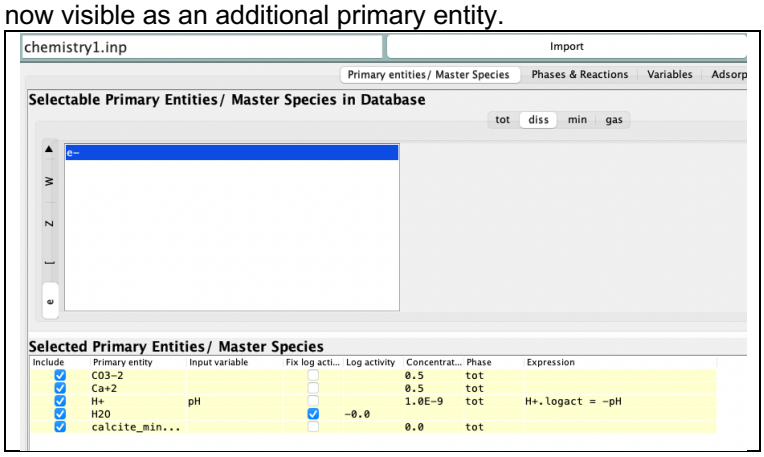


Step 3: add calcite_mineral as a primary entity

To do so, navigate to 'GUI view' and to the section "the primary entities". Add here calcite_mineral as a primary entity following the steps described in section 4.9. Here, we put calcite_mineral in the **solid** phase:

```
//******** The primary entities *******
@species(CO3-2, -2)
@primary_entity(CO3-2, -9.0, tot, 0.5)
@species(Ca+2, 2)
@primary_entity(Ca+2, -9.0, tot, 0.5)
@Global: pH
@Calc:(1, "H+.logact = -pH")
@species(H+, 1)
@primary_entity(H+, pH, 7.0, lin, 0.1, tot, 1.0E-9)
@entity(H2O, liter, 55.6 )
@primary_entity(H2O, -0.0)
@entity(calcite_mineral, solid, 1) //add here calcite_mineral as species
@primary_entity(calcite_mineral, -9.0, tot, 0) //set species to primary entity
//*************************************
//******************************
```

To check, navigate back to 'text view' → 'Primary entities/Master Species'. Here, calcite_mineral is now visible as an additional primary entity.



Step 4: give composition to calcite_mineral

As described in section 4.9, the composition of calcite_mineral is given in the extra_text segment below the definition of the primary entities. **Be sure to add the kinetics_db.txt** file as indicated in section 4.9). Add here the composition of $CaCO_3$. This leads to the following:

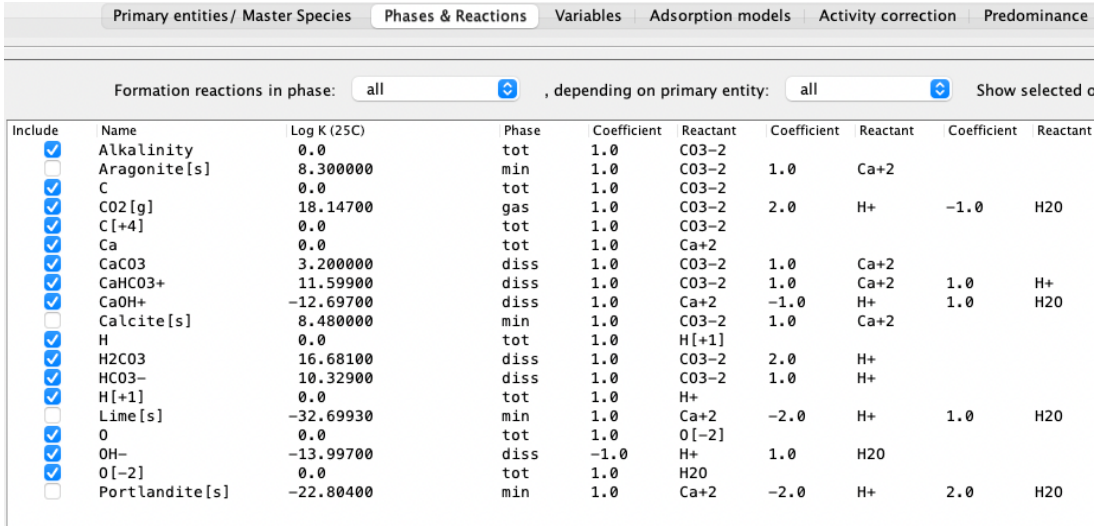
```
//********* The primary entities *******
@species(CO3-2, -2)
@primary_entity(CO3-2, -9.0, tot, 0.5)
@species(Ca+2, 2)
@primary_entity(Ca+2, -9.0, tot, 0.5)
@Global: pH
@Calc:(1, "H+.logact = -pH")
@species(H+, 1)
@primary_entity(H+, pH, 7.0, lin, 0.1, tot, 1.0E-9)
@entity(H2O, liter, 55.6 )
@primary_entity(H2O, -0.0)
@entity(calcite_mineral, solid, 1 )
@primary_entity(calcite_mineral, -9.0, tot, 0.0)
//*****************************************
//*******************************

//********* Extra text 2a **************

@class: extra_text_2a(){
@composition(calcite_mineral, 1, CO3-2, 1, Ca+2)
 // Here is some space for extra code that will be used in the calculations but will not be changed by the GUI.
}
@extra_text_2a()
```

Step 5: set reaction network

As we consider dissolution of minerals to be described kinetically, we must exclude those from the equilibrium calculation. Navigate to “phases & reactions” and uncheck all minerals:



Include	Name	Log K (25C)	Phase	Coefficient	Reactant	Coefficient	Reactant	Coefficient	Reactant
☑	Alkalinity	0.0	tot	1.0	CO3-2				
☐	Aragonite[s]	8.300000	min	1.0	CO3-2	1.0	Ca+2		
☑	C	0.0	tot	1.0	CO3-2				
☑	CO2[g]	18.14700	gas	1.0	CO3-2	2.0	H+	-1.0	H2O
☑	C[+4]	0.0	tot	1.0	CO3-2				
☑	Ca	0.0	tot	1.0	Ca+2				
☑	CaCO3	3.200000	diss	1.0	CO3-2	1.0	Ca+2		
☑	CaHCO3+	11.59900	diss	1.0	CO3-2	1.0	Ca+2	1.0	H+
☑	CaOH+	-12.69700	diss	1.0	Ca+2	-1.0	H+	1.0	H2O
☐	Calcite[s]	8.480000	min	1.0	CO3-2	1.0	Ca+2		
☑	H	0.0	tot	1.0	H[+1]				
☑	H2CO3	16.68100	diss	1.0	CO3-2	2.0	H+		
☑	HCO3-	10.32900	diss	1.0	CO3-2	1.0	H+		
☑	H[+1]	0.0	tot	1.0	H+				
☐	Lime[s]	-32.69930	min	1.0	Ca+2	-2.0	H+	1.0	H2O
☑	O	0.0	tot	1.0	O[-2]				
☑	OH-	-13.99700	diss	-1.0	H+	1.0	H2O		
☑	O[-2]	0.0	tot	1.0	H2O				
☐	Portlandite[s]	-22.80400	min	1.0	Ca+2	-2.0	H+	2.0	H2O

Note that Calcite[s] is still present in the list. This is because we have defined our own mineral called “calcite_mineral” with our own defined composition. Therefore, Calcite[s] has no actual relation to calcite_mineral and can still be treated as a separate species.

Step 6: export file

Press “Export”.

5.2.2 Calculating chemistry with Python.

For this example, we will examine what the effects of calcite dissolution are on the system composition at equilibrium. In Python, the amount of calcite is changed iteratively. For every calcite total, a new equilibrium composition is calculated using ORCHESTRA. In turn, the results are plotted using Python.

Step 1: create .py file and import modules.

3. Open a new python file and call it ‘calcite_dissolution.py’
4. Start the file by importing the required submodules:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import PyORCHESTRA # here, the ORCHESTRA submodule is imported
```

Step 2: get the ORCHESTRA function class 'p'.

```
p = PyORCHESTRA.ORCHESTRA()
```

Step 3: Initialise the ORCHESTRA calculator.

Now, the user must define the boundary conditions of the system for which chemical equilibrium will be calculated.

1. Name the chemistry text file. InputFile = 'chemistry1.inp'
2. Specify the number of cells. Here we will consider a 0-D system.
3. Specify the variables that will be used as input. Here, this includes the bulk composition (total $calcite\_mineral, Ca^{+2}$, H^+ and CO_3^{-2} in the system). **Do not include the pH here as this is an output** parameter. In addition to this, ORCHESTRA comes with pre-available parameters such as porosity and saturation.
4. Specify the variables that the user wants as output. Here, this includes the pH.
5. Call the function 'initialise' to initialize the ORCHESTRA calculator object.

```
InputFile = 'chemistry1.inp'
NoCells = 1 #only 1 cell to have a 0-D system
InVars = np.array(['calcite_mineral.tot', 'Ca+2.tot', 'CO3-2.tot', 'H+.tot', 'porosity',
'saturation'])
OutVars = np.array(['pH'])

p.initialise(InputFile, NoCells, InVars, OutVars)
```

Step 4: Set the initial conditions.

The 'p'-object now contains an initialized system that is ready to be used. By specifying values for the given input parameters (in InVar), an equilibrium will be calculated.

1. First, we create the variable matrix 'IN' containing arrays of the same length as InVars (1 value for every parameter must be given).
2. Using the `np.where` function, we can change the value in IN at the position corresponding to that parameter.
3. We will assume that the system is fully saturated, with a porosity of 1. The bulk composition is set to 0.5 $\frac{mol}{L}$ Ca^{+2} and 0.5 $\frac{mol}{L}$ CO_3^{-2}. Also include H^+, give it a total of 0.1 $\frac{mol}{L}$.

```
IN = np.array([np.ones_like(InVars)]).astype(float)

IN[0][np.where(InVars == 'porosity')] = 1
IN[0][np.where(InVars == 'saturation')] = 1
IN[0][np.where(InVars == 'Ca+2.tot')] = 0.5
IN[0][np.where(InVars == 'CO3-2.tot')] = 0.5
IN[0][np.where(InVars == 'H+.tot')] = 0.9
```

Step 5: change calcite_mineral and get equilibrium

The total amount of calcite_mineral is iteratively lowered from 0.5 to 0.0 mol/L with a stepsize of 0.001. For every new 'configuration', system equilibrium is calculated using 'set_and_calculate'. When ORCHESTRA calculates equilibrium, it returns a matrix with values for the parameters listed in OutVars.

1. Define range for calcite_mineral (here, using `np.arange`).
2. Do a simple for loop, iterating over the different amounts of calcite_mineral.
3. Assign the new calcite_mineral-value to the correct position in InVars.
4. Calculate equilibrium with p.set_and_calculate(IN)
5. Assign output values to the correct lists.

```
#define output lists
pH = []

#loop over different amounts of calcite
for calcite in calcite_list:
          #change amount of calcite
```

```
        IN[0][np.where(InVars == 'calcite_mineral.tot')] =calcite

        #run ORCHESTRA
        OUT = p.set_and_calculate(IN)

        #get output
        pH.append(OUT[0])
```

Step 6: plot the results

As the figure indicates, the pH increases as calcite dissolves!