Phase Identification Numbers in THAMES

Jeffrey W. Bullard

2021-03-08 17:45:09-06:00

Contents

This document describes the different kinds of identification numbers in THAMES and how the relate to one another.

1 Introduction to ID numbers: ICs and DCs

Every component in THAMES, whether it is an independent component (IC), a dependent component (DC) or a phase, has *two* ways to identify it:

- its name (a C++ string)
- its identification number (a C++ string)

The reason that each component needs an identification number is that properties of the components (number of moles, atomic mass, *etc*) are stored in different arrays (or C++ vectors), and the elements of a vector are distinguished by their order in the list; the first element is element 0, the second is 1, and so on. So for example, suppose we have nine independent components (ICs) and we are storing the number of moles of the independent components in a vector called `icmoles`. The vector might look like this:

```
icmoles[0] = 0.2 , icmoles[1] = 1.05, . . . , icmoles[8] = 0.47
```

Now let's suppose that I want to change the number of moles of Si in the system. I need to know which element in `icmoles` corresponds to Si. We do that by assigning

an integer to Si, called an 'id'. For example, if I assign the id number 2 to Si, then I will always know that `icmoles[2]` is the one I need to change.

So every IC has a unique id number and a unique name. Similarly, every DC has a unique id number and a unique name. Here is some basic information about them:

- The IC and DC names are read from the GEMS DCH input file.
- The order in which an IC name is read from the DCH files is equal to its id number. For example, if Al is the first IC name read from the DCH file, then the id number for Al is 0.
- The same is true for the DC names. That is, the order in which a DC name is read from the DCH file is equal to its id number.
- The IC names are stored in the `ChemicalSystem` object, using a C++ vector called `ICnames_`. The DC names are also stored in the `ChemicalSystem` object using a vector called `DCnames_`.
- The numbers of moles of each IC are stored in the `ChemicalSystem` object using an array called `ICmoles_`, and similarly the numbers of moles of each DC are stored using an array called `DCmoles_`.
- The total number of ICs is remembered in the `ChemicalSystem` object with the integer called `ICnum_`, and the total number of DCs is remembered with the integer called `DCnum_`.

Notice that all of this information is stored in the `ChemicalSystem` object. In fact, the `ChemicalSystem` object stores almost *all* of the chemical information in THAMES, and the other parts of THAMES gain access to that information using functions that communicate with the `ChemicalSystem` object. Here is some other date about the ICs and DCs that are stored there:

- The molar masses of the ICs are stored in a vector called `ICmolarmass_`, and the molar masses of the DCs are stored in a vector called `DCmolarmass_`.
- For DCs, we need to know how many moles of each IC it contains in one formula unit. This is stored in a two-dimensional vector (the same thing as a matrix) called `DCstoich_`. Each row of the matrix is a DC, and each column in that row is the number of units of an IC in that DC.

Example 1. Suppose I have a system with three only three ICs in this order: Al, H, and O. Furthermore, suppose that there are three DCs in this system in this order: Al^{3+}, $Al(OH)_4^+$, and H_2O. Based on this information, we can write down what will be stored in all of the data structures we have described so far. See if you can understand each one.

Structure	Contents
ICname_	ICname_[0] = "Al" ICname_[1] = "H" ICname_[2] = "O"
DCname_	DCname_[0] = "Al3+" DCname_[1] = "Al(OH)4+" DCname_[2] = "H2O@"
ICmolarmass_	ICmolarmass_[0] = 26.981 ICmolarmass_[1] = 1.001 ICmolarmass_[2] = 15.999
DCmolarmass_	DCmolarmass_[0] = 26.981 DCmolarmass_[1] = 95.01 DCmolarmass_[2] = 18.020
DCstoich_	1 0 0 1 4 4 0 2 1

1.1 Accessing the data of an IC or DC

C++ is designed in such a way that the stored data in an object are *private*, which just means that they can only be accessed directly from within the object itself. If I want to use the data from within the object I can look directly at the vector element that holds it. For example, if I want to assign the molar mass of the first IC to a variable called `mmass` in the `ChemicalSystem.cc` file, I can just write this:

```
// Only works from within ChemicalSystem.cc
mmass = ICmolarmass_[0];
```

However, if I want to do the same thing from within a different object, you have to communicate with `ChemicalSystem` with a "getter" function:

```
// Works in other files
mmass = chemsys_->getICmolarmass(0);
```

Notice that now the id number, 0, is an argument of the function, enclosed in parentheses, instead of a vector index enclosed in square brackets. THAMES has been written to include both getter and setter functions for nearly all of the private data in each class, so you will use them most of the time

Next, suppose I want to change the number of moles of the third IC to the value 3.14. To do that, I can type this:

```
// Only works from within ChemicalSystem.cc
ICmoles_[2] = 3.14;
```

Alternatively, I can use the setter function,

```
// Works in other files
chemsys_->setICmoles(2,3.14);
```

1.2 Working with IC and DC id numbers

Usually, one will not memorize the id number of a particular IC or DC, but it is likely that you may know its name. THAMES has functions that will tell you the id number of a component with a given name. For example, the molar mass of the IC named `Al` can be found this way:

```
// Works in other files
int idAl = chemsys_->getICid("Al");
mmass = chemsys_->getICmolarmass(idAl);
```

Or you can even combine the two lines into one:

```
// Works in other files
mmass = chemsys_->getICmolarmass(chemsys_->getICid("Al"));
```

In addition, there is also a function that will do this directly by name:

```
// Works in other files
mmass = chemsys_->getICmolarmass("Al");
```

All three of these alternatives will accomplish the same task. And there are similar functions for other IC and DC data as well. (Almost) all the getter functions start with `get` and the setter functions start with `set`, and the next letter after that is always upper case. You can scan through all the function names and what they do in the `ChemicalSystem.h` file.

2 Phases

2.1 GEM phases work the same way ...

The `ChemicalSystem` object holds a master list of all the phases that GEMS recognizes in a particular system (I will call these phases "GEM phases" to distinguish them from microstructure phases below). Again, those GEM phase names are stored in the DCH file and they are read, in order, into a vector called `phasename_`. The same object also holds the molar mass in `phasemolarmass_`, number of moles of each GEM phase in `phasemoles_`, and mass of each GEM phase in `phasemass_`. And you can access all of these data using the same kinds of functions as for IC and DC components. Here are just a few examples.

```
// Get the phase name of the second phase
string pname2 = chemsys_->getPhasename(1);

// Get the molar mass of the fourth phase
double mm04 = chemsys_->getPhasemolarmass(3);

// Get the molar mass of the phase named Alite
double mmAlite = chemsys_->getPhasemolarmass("Alite");

// Set the moles of the sixth phase to 3.1
chemsys_->setPhasemoles(5,3.1);

// Get the mass of the phase named Gp
double gypsummass = chemsys_->getPhasemass("Gp");
```

In the same way that DCs can contain multiple ICs, so also GEM phases can contain multiple DCs. For example, the aqueous phase called "aq-gen" contains dozens of DCs, one for each type of dissolved ion or molecule that can exist in the solution. Another example is the GEM phase called "Alite", which has only one DC called "C3S".

There is a vector of vectors in the `ChemicalSystem` object called `phaseDCmembers_` that stores a list of every DC that is associated with each phase. In addition, there are several functions that access the information in `phaseDCmembers_`:

- `chemsys_->getPhaseDCmembers(i)` will return a vector of all the DC id numbers that are associated with the GEM phase id number `=i`.
- `chemsys_->getPhaseDCmembers(pname)` will return a vector of all the DC id numbers that are associated with the GEM phase name `pname`.
- `chemsys_->getPhaseDCmembers(i,j)` will return the id number of the `j`-th DC associated with the GEM phase id number `i`.

2.2 ...but phases are more complicated

The microstructure aspect of THAMES makes the accounting of phases more complicated because we map each microstructure phase to one or more GEM phases. For example, in THAMES we have a microstructure phase called `C3S` that is mapped to the GEM phase called `Alite`. And we usually have a microstructure phase called `GYPSUM` that is mapped to the GEM phase called `Gp`.

In fact, we could do all of the microstructure phases this way, so that every microstructure phase maps to exactly one GEM phase. Some aspects of THAMES would be easier if we did that. But in most cement systems there will be nearly 100 different phases, and some of them are very closely related. For example, GEMS defines at least seven different carbonated AFm type phases that differ in small ways from each other.[1] But for representing the microstructure we will find it convenient to combine all seven of them together into a single microstructure phase called `AFMC`. For this reason, we keep a separate list of microstructure phase names and microstructure id numbers. Both of these are defined by the user in the `chemistry.xml` file, and they are stored in the `ChemicalSystem` object with the vectors `micphasename_` and `micid_`. As before, there are getter and setter functions for these as well:

```
// Get the name of the third microstructure phase in chemistry.xml
string pname = chemsys_->getPhasename(2);

// Get the microstructure id number of the ninth microstructure
// phase in chemistry.xml
int id8 = chemsys_->getMicid(8);

// Set the microstructure id number of the fourth microstructure
// phase to the value 7
```

[1] C_4AcH_9, $C_4Ac_{0.5}H_{10.5}$, $C_4Ac_{0.5}H_{12}$, $C_4Ac_{0.5}H_9$, C_4AcH_{11}, $C_4Fc_{0.5}H_{10}$, and C_4FcH_{12}.

```
chemsys_->setMicid(3,7);
```

THAMES has some functions to keep track of the mapping between "GEM phases" and "microstructure phases":

```
// Get the list of all GEM phase id numbers
// belonging to the microstructure phase called AFMC
vector<int> gpids = chemsys_->getMicphasemembers("AFMC");

// Get the second GEM phase id that is associated with the
// microstructure phase called HEMIANH
int gpid = chemsys_->getMicphasemembers("HEMIANH",1);
```

One can also directly access all of the DCs that are associated with a given microstructure phase with these functions:

- `chemsys_->getMicDCmembers(i)` will return the vector of all DC id numbers that are associated with the microstructure phase id number `i`.
- `chemsys_->getMicDCmembers(mname)` will return the vector of all DC id numbers that are associated with the microstructure phase name `mname`.
- `chemsys_->getMicDCmembers(i,j)` will return the id number of the `j`-th DC that is associated with the microstructure phase id number `i`.

2.3 Microstructure phases grouped by their kinetic behavior

Among all the microstructure phases that are defined, we categorize them in terms of how the changes to their number of moles are determined. There are currently three of these categories:

1. **Kinetically controlled phases** are microstructure phases that change moles according to some kind of kinetic rate equation. The list of the id numbers of these phases is stored in the `KineticModel` object in a vector called `kineticphase_`.
2. **Readily soluble phases** are microstructure phases that will dissolve completely during the first time step. The list of the id numbers of these phases is stored in the `KineticModel` object in a vector called `solublephase_`
3. **Thermodynamically controlled phases** are microstructure phases that have their moles determined by chemical equilibrium in the GEMS model. The list of the id numbers of these phases is stored in the `KineticModel` object in a vector called `thermophase_`

Example 2. Suppose I have a microstructure with five microstructure phases given *in this order* in the `chemistry.xml` file:

- Name = C3S, id = 2 (kinetically controlled)
- Name = C3A, id = 4 (kinetically controlled)

- Name = Calcite, id = 5 (thermodynamically controlled)
- Name = K2SO4, id = 7 (readily soluble)
- Name = Ca(OH)2, id = 8 (thermodynamically controlled)

Here is how these data will be stored in the `KineticModel` object. See if you can understand each one.

Structure	Contents
`kineticphase_`	`kineticphase_[0] = 2` `kineticphase_[1] = 4`
`solublephase_`	`solublephase_[0] = 7`
`thermophase_`	`thermophase_[0] = 5` `thermophase_[1] = 8`

Why do we keep these different categories in separate lists? Because frequently we may want to perform a set of operations on all the kinetically controlled microstructure phases without doing anything to the other ones. To do that, we would go through the list `kineticphase_` one by one until we get to the end of that list. Otherwise, we would need to go through every microstructure phase and check for each one whether it is kinetically controlled or not.

2.4 ChemicalSystem keeps track of these categories, too

The `ChemicalSystem` object has several data structures and several functions that keep track of whether a phase is kinetically controlled, thermodynamically controlled, or readily soluble. Here are the functions:

- `chemsys_->getMic2kinetic(i)` will return the `kineticphase_` id number of the microstructure phase `i`. If the microstructure phase is not kinetically controlled, this function will a value of -1 to indicate that it is not kinetically controlled.
- `chemsys_->getKinetic2mic(i)` will return the microstructure phase id number stored at `kineticphase_[i]`.
- `chemsys_->isKineticphase(i)` will return `true` if microstructure phase id number `i` is kinetically controlled, or `false` if not.
- `chemsys_->getMic2thermo(i)` will return the `thermophase_` id number of the microstructure phase `i`. If the microstructure phase is not thermodynamically controlled, this function will a value of -1 to indicate that it is not thermodynamically controlled.
- `chemsys_->getThermo2mic(i)` will return the microstructure phase id number stored at `thermophase_[i]`.
- `chemsys_->getMic2soluble(i)` will return the `solublephase_` id number of the microstructure phase `i`. If the microstructure phase is not readily soluble, this function will a value of -1 to indicate that it is not readily soluble.

- `chemsys_->getSoluble2mic(i)` will return the microstructure phase id number stored at `solublephase_[i]`.

3 Exercises

1. Write a line or two of C++ code that will determine the atomic mass of an IC named "C".
2. Write a line or two of C++ code that will determine the atomic mass of a DC named "SiO2".
3. Write a line or two of C++ code that will determine how many moles of oxygen are contained in every mole of the DC named "Al2O3".
4. Write a line or two of C++ code to change the mass of a microstructure phase called "SFUME".
5. (Challenging) Write some C++ code that will determine how many moles of an IC named "S" are associated with GEM phase called "ettr30".
6. (Very challenging) Write some C++ code to increase the number of moles of an IC called "Mg" due to the dissolution of 0.5 moles of a microstructure phase called "MYPHASE" that is associated with a GEM phase called "Mphs"