Getting Help

CompuTherm LLC is committed to providing you with the best possible technical support. Please contact us via the following approaches:

Web Site
www.computherm.com

E-Mail
info@computherm.com

Phone
+1 (608) 203 8843

Fax
+1 (608) 203 8045

Mail
CompuTherm LLC
8401 Greenway Blvd., Suite 248
Middleton, WI 53562
USA

Declaration

This document is furnished by CompuTherm LLC for information purposes only to licensed users of the PanEngine product and is furnished on an “AS IS” basis without any warranties expressed or implied. Information in this document is subject to change without notice and does not represent a commitment on the part of CompuTherm LLC.
## Contents

1 Introduction to PanEngine  
1.1 What is PanEngine? .............................. 1  
1.2 Advantages of PanEngine .......................... 1  
1.3 API in PanEngine .................................. 1  

2 Getting Started with PanEngine .................. 3  
2.1 Installation of PanEngine ....................... 3  
2.2 Getting Started with PanEngine .................. 4  

3 Basic Concepts .................................. 9  
3.1 Gibbs Energy Models for Multi-Component Phases ..................................................... 9  
  3.1.1 Stoichiometric compound ..................... 9  
  3.1.2 Disordered solution phase ................... 9  
  3.1.3 Ordered intermetallic phase using the compound energy formalism .......... 10  
3.2 PanEngine Classes ................................ 10  
  3.2.1 class P_Component ........................... 11  
  3.2.2 class P_Species ............................... 11  
  3.2.3 class P_Sstatespace ......................... 11  
  3.2.4 class P_PhasePoint ......................... 12  
  3.2.5 class P_Point ................................. 12  

4 PanEngine API .................................... 13  
4.1 Functions for PanEngine Pointer ................. 15  
  4.1.1 Define a PanEngine pointer ................. 15  
  4.1.2 Delete a PanEngine pointer ................. 15  
4.2 Functions for System ........................... 15  
  4.2.1 Set system configuration .................... 15  
  4.2.2 Import a thermodynamic database ........... 16  
  4.2.3 Export a subsystem database into a file (tdb) .............................................. 16  
  4.2.4 Get component names in a database ......... 16  
  4.2.5 Deactivate a component in a database ...... 16  
  4.2.6 Get phase names in a database .............. 17  
  4.2.7 Get and set phase statuses .................. 17  
4.3 Functions for Point Calculation ................. 17  
  4.3.1 Set calculation condition .................. 17  
  4.3.2 Calculate a global phase equilibrium .... 18  
  4.3.3 Calculate a global phase equilibrium with initials ................................. 18  
  4.3.4 Calculate a local phase equilibrium ...... 18  
  4.3.5 Find the liquidus surface .................. 19  
  4.3.6 Find the liquidus slopes .................... 19
4.4 Function for Solidification Simulation .......................... 20
4.5 Thermodynamic Factors ........................................ 21
4.6 Hessian matrix of Gibbs free energy ............................. 21
4.7 Parallel Tangent Equilibrium .................................... 22

5 Examples .................................................................. 24
  5.1 Test Example 1 ..................................................... 24
  5.2 Test Example 2 ..................................................... 25
  5.3 Test Example 3 ..................................................... 25
  5.4 Test Example 4 ..................................................... 25
  5.5 Test Example 5 ..................................................... 25
  5.6 Test Example 6 ..................................................... 25
  5.7 Test Example 7 ..................................................... 26
  5.8 Test Example 8 ..................................................... 26
  5.9 Test Example 9 ..................................................... 26
1 Introduction to PanEngine

1.1 What is PanEngine?

PanEngine is a dynamically linked library (DLL) in 64-bit for multi-component thermodynamics and phase equilibrium calculations. PanEngine is the calculation engine of Pandat. It has an application program interface (API) which allows a user’s custom program to access the functions in PanEngine. It is implemented with C++TR1 standard in Microsoft Visual Studio. The examples in this manual were prepared under Microsoft Visual Studio 2015.

A library is a group of functions, classes, or other resources that can be made available to application programs that need previously implemented entities without the need to know how these functions, classes, or resources were created or how they function. A dynamic link library is a program that holds one or more functions or some functionality that other programs can use. Through PanEngine’s API, users can call the thermodynamic functions available in PanEngine and create custom software for their specific applications.

Custom Software Applications

PanEngine can be used by users to create custom software applications such as:

- Microscopic solidification simulations
  - Microstructure: e.g. the secondary dendrite arm spacing
  - Microsegregation: e.g. the concentration profile within a dendrite arm
- Macroscopic solidification simulations
  - Casting simulations: PanEngine provides enthalpy and the fraction-solid as a function of temperature as well as physical properties such as density, and thermal conductivity
- Heat treatment simulations
- Other applications where phase equilibrium information and thermodynamic properties are needed, such as the cellular automaton (CA) and phase field simulations

1.2 Advantages of PanEngine

PanEngine automatically finds the correct, stable phase equilibria without requiring the user to guess initial values. This is especially important when integrating with user’s custom program for the following reasons:

- It is very difficult for a user to provide initial values and verify results when a custom software program needs stable phase equilibrium repeatedly for thousands of points.
- It is almost impossible for a user to guess the initial values in a multi-component phase equilibrium calculation.

1.3 API in PanEngine

PanEngine’s API has many commonly used functions. Some of them are listed below and more details will be given in the following sections.

- Import databases
• Set calculation conditions
• Calculate stable equilibria
• Calculate metastable (local) equilibria
• Calculate parallel tangent equilibria for phase field modeling
• Calculate driving force of a phase
• Find liquidus surface
• Calculate liquidus and solidus slopes
• Calculate partitioning coefficients
• Simulate a solidification process using Scheil or lever rule model
• Calculate physical properties such as molar volume and density
• Calculate kinetic properties such as mobility and diffusivity
• Calculate Hessian matrix of Gibbs free energy and its eigenvalues and eigenvectors
• Calculate user-defined properties of a phase or a system
2 Getting Started with PanEngine

2.1 Installation of PanEngine

PanEngine is available only from CompuTherm LLC. Once purchased, a hardware dongle will be provided with PanEngine. PanEngine will not run if the dongle is not attached to the user’s computer’s USB port. PanEngine consists of several different types of files. As shown in Table 1, the PanEngine thermodynamic calculation interface includes .lib files, .dll files, .h files, Visual Studio Solution and Project files, and some test example files (.cpp).

Table 1: List of PanEngine Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanEngineX.dll</td>
<td>PanEngine dynamically linked library</td>
</tr>
<tr>
<td>PanSolverX.dll</td>
<td>Collection of dynamically linked libraries used by PanEngineX.dll</td>
</tr>
<tr>
<td>haspms32.dll</td>
<td></td>
</tr>
<tr>
<td>hasp_windows_57714.dll</td>
<td></td>
</tr>
<tr>
<td>PanEngineX.lib</td>
<td>PanEngine library file</td>
</tr>
<tr>
<td>PanEngineX.h</td>
<td>PanEngine header file</td>
</tr>
<tr>
<td>std.h</td>
<td>Standard header file</td>
</tr>
<tr>
<td>stl.h</td>
<td>Standard Template Library (STL) header file</td>
</tr>
<tr>
<td>Pan_Global.h</td>
<td>Other header files</td>
</tr>
<tr>
<td>Pan_Global_Def.h</td>
<td></td>
</tr>
<tr>
<td>solidification.h</td>
<td></td>
</tr>
<tr>
<td>main.cpp</td>
<td>Main program</td>
</tr>
<tr>
<td>PanEngineTest.*.cpp</td>
<td>PanEngine test example files (*=1,2,⋯,9)</td>
</tr>
<tr>
<td>AlMgZn.tdb, AlSiZn.tdb, FeNiCr.tdb, NiAlNb_Pseudo.tdb</td>
<td>Example database files in tdb format</td>
</tr>
<tr>
<td>PanEngineXTest.sln</td>
<td>Visual Studio Solution file</td>
</tr>
<tr>
<td>PanEngineXTest.vcxproj</td>
<td>VC++ Project file</td>
</tr>
<tr>
<td>PanEngineXTest.exe or PanEngineXTest_demo.exe</td>
<td>Compiled executable application file</td>
</tr>
</tbody>
</table>

The installation of PanEngine is rather straightforward. Simply copy all the files in the PanEngine
distribution CD or unzip the compressed files to any working directory where the user intends to place his/her own codes for applications. The library files can also locate in any other area, and can be accessed by specifying their appropriate paths in the application program codes. The current PanEngine can run on most of recent versions of Windows.

The recommended programming environment with PanEngine is Microsoft Visual Studio 2013 or 2015. Visual Studio 2013 or 2015 is the programming environment in which PanEngine was created. If a user uses a different C++ compiler, the PanEngine Visual Studio Solution file and corresponding project file may not work and then a completely new Solution and project files or make file need to be constructed by the user, or contact CompuTherm for solutions.

2.2 Getting Started with PanEngine

We assume that a full version of Microsoft Visual Studio 2013 or 2015 is installed on the user’s computer. Follow the steps below to run the PanEngine test examples.

1. Attach the CompuTherm hardware dongle to the computer.
3. On the Start page of Visual Studio 2013 or 2015, click on Open Project. Go to the folder /PanEngineXTest (in the user’s hard drive) and open PanEngineXTest.sln.

Here is what we will see in Visual Studio after expanding the file folders in the Solution Explorer:
4. Double click on the file \texttt{main.cpp} in the Solution Explorer window.

5. Rebuild PanEngine by clicking \texttt{Build} \rightarrow \texttt{Rebuild Solution}.
6. Press F5 to run the test examples. A **Command** window will pop up as below.

![Command window](image)

There are nine test examples to select. To select a test example, enter the example ID 1 to 9. Type “0” to exit, and type “-1” to run all the test examples together. The **Command** window will show the intermediate results of the calculations. The final status of the window looks like the following one (after selected “-1”), except that the path name on the top of the window will depend on the location of **PanEngine** on the user’s computer.

![Final status](image)
7. Press any key and return to the Visual Studio main window.

In the `main.cpp`, there is a line

```c
#define OUTPUT_TO_FILE
```

as shown in the following image.

If this `#define` line is commented out, as in above image, the intermediate calculation results will be shown in the `Command` window. Otherwise, the results will be output into a file with a name of “test_1.dat”, or “test_2.dat”, etc.

In the following, we will introduce some basic concepts used in PanEngine and describe the details of the API and the test examples.
3 Basic Concepts

In the CALPHAD approach, the Gibbs energies of all the phases in an alloy system are described by thermodynamic models, such as the ones for stoichiometric phases, the regular-solution-type model for disordered phases, and the sublattice model for ordered phases with a range of homogeneity or an order/disorder transition. These types of models have been implemented in PanEngine.

PanEngine was developed with C++ language. It consists of many C++ classes. The P_POINT C++ class refers to a system with a specified composition at a certain temperature and pressure. A P_POINT object is directly interfaced with the user’s code. The user can change the conditions (temperature or overall compositions) of the system through a P_POINT object and get back the thermodynamic properties and phase equilibrium information under the newly specified conditions. The stable (or metastable) phase equilibrium information of the system (such as phase fractions, composition of each phase, and thermodynamic properties for each phase) are described by P_PHASE_POINT Class. General information about the system, such as alloying components, alloy overall composition and temperature, are stored in P_STATESPACE class. The details on these and other classes can be found in the C++ header files of Pan_Global_Def.h and Pan_Global.h.

In the following, we will first give a brief introduction of thermodynamic models and then describe the different classes used in PanEngine.

3.1 Gibbs Energy Models for Multi-Component Phases

3.1.1 Stoichiometric compound

The Gibbs energy of a stoichiometric phase is expressed as

$$G = \sum_{i=1}^{n} x_i G_i^{\phi} + G_f$$  \hspace{1cm} (1)

where $x_i$ is the mole fraction of component $i$, $G_i^{\phi}$ is the Gibbs energy of the pure component $i$ with structure $\phi$, and $G_f$ is the Gibbs energy of formation of the stoichiometric phase referred to the structure $\phi$ for each component $i$.

3.1.2 Disordered solution phase

The Gibbs energy of a disordered solution phase is expressed as

$$G = \sum_{i=1}^{n} x_i G_i^{\phi} + RT \sum_{i=1}^{n} x_i \ln x_i + G^{ex,\phi}$$  \hspace{1cm} (2)

where $x_i$ is the mole fraction of component $i$, $G_i^{\phi}$ is the Gibbs energy of the pure component $i$ with structure $\phi$, $R$ is the gas constant, and $T$ is the absolute temperature. $G^{ex,\phi}$ is the excess Gibbs energy of the phase, defined as

$$G^{ex,\phi} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j \sum_{l=0}^{m} L_{ij}^{(l)} (x_i - x_j)^l + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} x_i x_j x_k \sum_{l=1, j, k}^{n-1} L_{ijk}^{(l)} V_{ijk}^{(l)}$$  \hspace{1cm} (3)
where the first term represents the binary interaction terms, the second represents ternary interactions. The $L_{ij}^{(l)}$'s are binary interaction parameters for the $i$-$j$ binary and the $L_{ijk}^{(l)}$'s are ternary interaction parameters. $V_{ijk}^{(l)}$ is defined as
\[ V_{ijk}^{(l)} = x_l + \frac{1 - x_i - x_j - x_k}{n} \quad (l = i, j, k) \] (4)
For a ternary system,
\[ V_{ijk}^{(l)} = x_l \quad (l = i, j, k) \] (5)
since $x_i + x_j + x_k = 1$.

### 3.1.3 Ordered intermetallic phase using the compound energy formalism

The Gibbs energy of an ordered intermetallic phase is described as
\[ G = G^{ref} + G^{id} + G^{ex} \] (6)
where $G^{ref}$ is expressed in terms of compound energies (which are constant at constant temperature) and their associated sublattice species concentrations, $y_p^i$,
\[ G^{ref} = \sum y_p^i y_q^j \cdots y_s^l G_{p,q,\ldots,s} \] (7)
$G^{id}$ is the ideal mixing term, which assumes the random mixing of species on each sublattice,
\[ G^{id} = \sum_{i=1}^l f_i \sum_{p=1}^m y_p^i \ln y_p^i \] (8)
$G^{ex}$ is also expressed as a function of species concentrations with the sublattice $L$ parameters being the numerical coefficients in the contributing terms,
\[ G^{ex} = \sum y_p^i y_q^j y_r^l L_{p,q,r} \] (9)
where
\[ L_{p,q,r} = \sum_v L_{p,q,r}^v (y_p^i - y_q^j)^v \] (10)

### 3.2 PanEngine Classes

**PanEngine** is a dynamically linked library of thermodynamic and phase equilibrium calculation functions. Most of the communications between the user’s application code and **PanEngine** are through the objects of **PanEngine** classes as mentioned at the beginning of this chapter. The headers of **PanEngine** classes are included in the files *Pan_Global.h* and *Pan_Global_def.h*. The major **PanEngine** classes with frequently used functions and properties are described below. Please refer to the header files of *Pan_Global.h* and *Pan_Global_def.h* for other classes.
### 3.2.1 class P_Component

<table>
<thead>
<tr>
<th>Class Name</th>
<th>P_Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>a component is made up of one or more elements, for example: Al or FeO</td>
</tr>
<tr>
<td>Public Functions</td>
<td>virtual ~P_Component()</td>
</tr>
<tr>
<td></td>
<td>(and other copy constructors and operators)</td>
</tr>
<tr>
<td>Public Properties</td>
<td>string m_name // component name</td>
</tr>
<tr>
<td></td>
<td>int m_id // component ID</td>
</tr>
<tr>
<td></td>
<td>(see Pan_Global_Def.h for other member variables)</td>
</tr>
<tr>
<td>Comments</td>
<td>P_Component holds information for a component</td>
</tr>
</tbody>
</table>

### 3.2.2 class P_Species

<table>
<thead>
<tr>
<th>Class Name</th>
<th>P_Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>species can be made up of one or more components, for example: O2</td>
</tr>
<tr>
<td>Public Functions</td>
<td>P_Species()</td>
</tr>
<tr>
<td></td>
<td>virtual ~P_Species()</td>
</tr>
<tr>
<td></td>
<td>(and other copy constructors and operators)</td>
</tr>
<tr>
<td>Public Properties</td>
<td>string m_name // species name</td>
</tr>
<tr>
<td></td>
<td>vector&lt;pair&lt;string, double&gt;&gt;m_c</td>
</tr>
<tr>
<td></td>
<td>// first: component name; second: amount of component</td>
</tr>
<tr>
<td></td>
<td>(see Pan_Global_Def.h for other member variables)</td>
</tr>
<tr>
<td>Comments</td>
<td>associate model uses P_Species to define the species of the associates</td>
</tr>
</tbody>
</table>

### 3.2.3 class PStatespace

<table>
<thead>
<tr>
<th>Class Name</th>
<th>PStatespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Statespace describes temperature, pressure and composition of a system or a phase</td>
</tr>
<tr>
<td>Public Functions</td>
<td>PStatespace()</td>
</tr>
<tr>
<td></td>
<td>virtual ~PStatespace()</td>
</tr>
<tr>
<td></td>
<td>(and other copy constructors and operators)</td>
</tr>
<tr>
<td>Public Properties</td>
<td>double m_T; // in K, system temperature</td>
</tr>
<tr>
<td></td>
<td>double m_P; // in pascal, system pressure</td>
</tr>
<tr>
<td></td>
<td>map&lt;string, shared_ptr&lt;P_Component&gt;&gt;m_comp</td>
</tr>
<tr>
<td></td>
<td>// collection of components</td>
</tr>
<tr>
<td></td>
<td>(see Pan_Global_Def.h for other member variables)</td>
</tr>
<tr>
<td>Comments</td>
<td>most of calculation conditions are set through this class</td>
</tr>
</tbody>
</table>
### 3.2.4 class P_phase_point

<table>
<thead>
<tr>
<th>Class Name</th>
<th>P_phase_point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>information for a phase after a calculation: the state space, thermodynamic properties, species concentrations, etc.</td>
</tr>
</tbody>
</table>
| Public Functions | P_phase_point()  
                   virtual ~P_phase_point()  
                   (and other copy constructors) |
| Public Properties | string m_phase_name // phase name  
                       int m_phase_id // phase ID  
                       (see Pan_Global_Def.h for other member variables) |
| Comments       | the objects of this class will be created by PanEngine after calculation |

### 3.2.5 class P_Point

<table>
<thead>
<tr>
<th>Class Name</th>
<th>P_Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>an equilibrium state with one or more phase points (P_phase_point)</td>
</tr>
</tbody>
</table>
| Public Functions | P_Point()  
                   virtual ~P_Point()  
                   P_Point(const P_Point&)  
                   (and other copy constructors and member functions) |
| Public Properties | shared_ptr<P_Statespace> m_st // statespace for a P_Point  
                       vector <shared_ptr<P_Phase_Point>> m_ppt;  
                       // phase point in this Point  
                       (see Pan_Global_Def.h for other member variables) |
| Comments       | for solidification, a P_Point includes the fractions of solid and liquid |
4 PanEngine API

The functions of the PanEngine application program interface (API) are defined as virtual functions in a class of PanEngine in PanEngine.h, except for the two global functions used for defining a PanEngine pointer and deleting an existing PanEngine pointer. These functions can be divided into four categories according to their purposes:

- **PanEngine Pointer**
  - define a PanEngine pointer and initialize it
  - delete an existing PanEngine pointer

- **System**
  - set system configuration
  - import a thermodynamic database
  - save a subsystem database
  - get system component names
  - get system phase names
  - get active phase names
  - set phase status
  - get phase status
  - activate a phase
  - deactivate a component
  - set calculation condition

- **Point Calculation**
  - find globally stable equilibrium
  - find globally stable equilibrium with initial
  - find metastable (local) equilibrium with initial
  - find liquidus surface
  - calculate liquidus slopes

- **Solidification Simulation**
  - lever rule model
  - Scheil model

Table 2 summarizes the functions of PanEngine API. These functions will be explained in detail in the following sections.
<table>
<thead>
<tr>
<th>Functions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern &quot;C&quot; PANENGINE_API PanEngine* definePanEngineUser(char* msg)</td>
<td>define a PanEngine pointer</td>
</tr>
<tr>
<td>extern &quot;C&quot; PANENGINE_API void deletePanEngineUser(PanEngine *pUser)</td>
<td>delete a PanEngine pointer</td>
</tr>
<tr>
<td>string pe_set_configuration(map&lt;string, string&gt;&amp; config)</td>
<td>set system configuration</td>
</tr>
<tr>
<td>string pe_import_database(pair&lt;string, string&gt;&amp; db, bool append=false, const pair&lt;string, string&gt;&amp; db_to=pair&lt;string, string&gt;());</td>
<td>import thermodynamic database</td>
</tr>
<tr>
<td>string pe_export_subsystem_database(pair&lt;string, string&gt;&amp; db, const string subsystem_name=string(&quot;sub.tdb&quot;), const vector&lt;string&gt;&amp; comp_name = vector&lt;string&gt;());</td>
<td>save a subsystem database</td>
</tr>
<tr>
<td>string pe_get_component_names(pair&lt;string, string&gt;&amp; db, vector&lt;string&gt;&amp; comp_name)</td>
<td>get component names in a database</td>
</tr>
<tr>
<td>string pe_deactivate_component(string&amp; comp_name)</td>
<td>suspend a component</td>
</tr>
<tr>
<td>string pe_get_phase_names(pair&lt;string, string&gt;&amp; db, vector&lt;string&gt;&amp; phase_name, vector&lt;string&gt; comp_name = vector&lt;string&gt;());</td>
<td>get phase names in a database with a given set of components</td>
</tr>
<tr>
<td>string pe_get_phase_status(pair&lt;string, string&gt;&amp; db, vector&lt;pair&lt;string, PAN_PHASE_STATUS&gt;&gt;&amp; phase_name_status)</td>
<td>get phase status in a database</td>
</tr>
<tr>
<td>string pe_set_phase_status(pair&lt;string, string&gt;&amp; db, vector&lt;pair&lt;string, PAN_PHASE_STATUS&gt;&gt;&amp; phase_name_status)</td>
<td>set phase status in a database</td>
</tr>
<tr>
<td>string pe_set_calculation_condition(const Pan_Calculation&amp; calc)</td>
<td>set a calculation condition</td>
</tr>
<tr>
<td>string pe_calc_point_global(shared_ptr&lt;P_Point&gt; p_pt)</td>
<td>calculate a global phase equilibrium for a point</td>
</tr>
<tr>
<td>string pe_calc_point_global_with_initial_point(shared_ptr&lt;P_Point&gt; p_pt)</td>
<td>calculate a global phase equilibrium for a point with initials</td>
</tr>
<tr>
<td>string pe_calc_point_local_with_initial_point(shared_ptr&lt;P_Point&gt; p_pt, bool given_f=false)</td>
<td>calculate a local phase equilibrium or parallel tangent equilibrium for a point with initials</td>
</tr>
<tr>
<td>string pe_find_liquidus_surface(string&amp; liquid_phase_name, shared_ptr&lt;P_Point&gt; p_pt)</td>
<td>find a liquidus surface</td>
</tr>
<tr>
<td>string pe_calc_liquidus_slope(string&amp; liquid_phase_name, string&amp; solvent_comp_name, shared_ptr&lt;P_Point&gt; p_pt)</td>
<td>calculate liquidus slopes</td>
</tr>
<tr>
<td>String pe_solidification_simulation(Solidification_Parameter&amp; s_param, Pan_Calculation&amp; calc, vector&lt;Solidification_Node&gt;&amp; solidification_result)</td>
<td>simulate solidification with lever rule or Scheil model</td>
</tr>
</tbody>
</table>
4.1 Functions for PanEngine Pointer

There are two functions associated with the PanEngine pointer: define a PanEngine pointer and delete a PanEngine pointer. These two functions are global functions.

4.1.1 Define a PanEngine pointer

<table>
<thead>
<tr>
<th>Name</th>
<th><code>extern &quot;C&quot; PANENGINE_API PanEngine* definePanEngineUser(char* msg)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>define a PanEngine pointer and initialize it</td>
</tr>
<tr>
<td>Arguments</td>
<td><code>msg</code> message returned from PanEngine</td>
</tr>
</tbody>
</table>

A PanEngine pointer must be successfully initialized before PanEngine’s other functions can be used. If the CompuTherm dongle is not attached to the computer, the initialization will fail.

4.1.2 Delete a PanEngine pointer

<table>
<thead>
<tr>
<th>Name</th>
<th><code>extern &quot;C&quot; PANENGINE_API void deletePanEngineUser(PanEngine *pUser)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>delete a PanEngine pointer after all calculations are done</td>
</tr>
<tr>
<td>Arguments</td>
<td><code>PanEngine *pUser</code> a defined and initialized PanEngine pointer</td>
</tr>
</tbody>
</table>

After a PanEngine pointer `pUser` is deleted, the system information inside PanEngine pointed to by `pUser` will be deleted and `pUser` will be a null pointer.

4.2 Functions for System

PanEngine API functions in the system level manage the system related information, such as importing a thermodynamic database and setting calculation conditions.

4.2.1 Set system configuration

<table>
<thead>
<tr>
<th>Name</th>
<th><code>string pe_set_configuration(map&lt;string, string&gt;&amp; config)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>set system configuration</td>
</tr>
<tr>
<td>Arguments</td>
<td><code>config</code> a map of pair of strings to define a configuration</td>
</tr>
</tbody>
</table>

One of the configurations is case sensitive of component names and phase names in a database. This can be set with this function as:

```cpp
map<string, string> config;
config["case_sensitive"] = "false";
s = user->pe_set_configuration(config);
```

which will convert all component and phase names into capital letters while reading the database.
4.2.2 Import a thermodynamic database

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_import_database(pair&lt;string, string&gt;&amp; db, bool append=false, const pair&lt;string, string&gt;&amp; db_to=pair&lt;string, string&gt;())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>import a thermodynamic database file in tdb format</td>
</tr>
<tr>
<td>Arguments</td>
<td>db database file name; append append the database db to the database db_to</td>
</tr>
</tbody>
</table>

Thermodynamic parameters are stored in files. The tdb type of file is a text file which can be modified by the user using a text editor. The pdb type of file is an encrypted database. If append=false, import the database in the file of db. If append=true, append the database in the file of db to the already imported database from the file of db_to.

4.2.3 Export a subsystem database into a file (tdb)

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_export_subsystem_database (pair&lt;string, string&gt;&amp; db, const string subsystem_name=string(&quot;sub.tdb&quot;), const vector&lt;string&gt;&amp; comp_name = vector&lt;string&gt;())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>export a subsystem thermodynamic database in tdb format into a file</td>
</tr>
<tr>
<td>Arguments</td>
<td>db database file name; subsystem_name subsystem database file name; comp_name component names in the subsystem</td>
</tr>
</tbody>
</table>

After a thermodynamic database (tdb) is successfully loaded, a subsystem with selected components can be exported into a database file with tdb format.

4.2.4 Get component names in a database

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_export_subsystem_database (pair&lt;string, string&gt;&amp; db, const string subsystem_name=string(&quot;sub.tdb&quot;), const vector&lt;string&gt;&amp; comp_name = vector&lt;string&gt;())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>get all component names in a database</td>
</tr>
<tr>
<td>Arguments</td>
<td>db database file name; comp_name component names</td>
</tr>
</tbody>
</table>

This function gets all component names in a database with the file name of db.

4.2.5 Deactivate a component in a database

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_deactivate_component(string&amp; comp_name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Deactivate a component in a database</td>
</tr>
<tr>
<td>Arguments</td>
<td>comp_name component to be deactivated</td>
</tr>
</tbody>
</table>
This function deactivates a component with the name of \texttt{comp\_name} in the current database.

### 4.2.6 Get phase names in a database

<table>
<thead>
<tr>
<th>Name</th>
<th>\texttt{string pe_get_phase_names(pair&lt;string, string&gt;&amp; db, vector&lt;string&gt;&amp; phase_name, vector&lt;string&gt; comp_name = vector&lt;string&gt;())}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>get phase names in a database with a given set of components</td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{db} database file name; \texttt{phase_name} phase names; \texttt{comp_name} selected component names</td>
</tr>
</tbody>
</table>

If \texttt{comp\_name} is given, this function gets the phase names in the subsystem with the components of \texttt{comp\_name} in a database with the file name of \texttt{db}. Otherwise, the function gets all phase names in a database with the file name of \texttt{db}.

### 4.2.7 Get and set phase statuses

<table>
<thead>
<tr>
<th>Name</th>
<th>\texttt{string pe_get_phase_status(pair&lt;string, string&gt;&amp; db, vector&lt;pair&lt;string, PAN_PHASE_STATUS&gt;&gt;&amp; phase_name_status)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>get phase status in a database</td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{db} database file name; \texttt{phase_name_status} vector of phase’s name and its status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>\texttt{string pe_set_phase_status(pair&lt;string, string&gt;&amp; db, vector&lt;pair&lt;string, PAN_PHASE_STATUS&gt;&gt;&amp; phase_name_status)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>set phase status in a database</td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{db} database file name; \texttt{phase_name_status} vector of phase’s name and its status</td>
</tr>
</tbody>
</table>

These two functions get and set the phase statuses in a database. Phase status takes values of \texttt{P\_PHASE\_ENTERED}, \texttt{P\_PHASE\_SUSPENDED}, \texttt{P\_PHASE\_DORMANT}, \texttt{P\_PHASE\_FIXED}, \texttt{P\_PHASE\_STATUS\_NOT\_DEFINED}. See Pandat User’s Guide for definition of the phase status.

### 4.3 Functions for Point Calculation

\texttt{PanEngine} uses a specially designed global optimization algorithm to find the most stable phase equilibrium automatically without guessing initial values. It also provides functions for performing locally metastable phase equilibrium calculations and other types of calculations. The point related calculations in \texttt{PanEngine} API are described below.
4.3.1 Set calculation condition

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_set_calculation_condition(const Pan_Calculation&amp; calc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>set a calculation condition</td>
</tr>
<tr>
<td>Arguments</td>
<td>calc calculation condition object</td>
</tr>
</tbody>
</table>

Calculation condition defines database to be used, units, selected components and phases, state space \((T, P, x_j)\). See test examples for detail.

4.3.2 Calculate a global phase equilibrium

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_calc_point_global(shared_ptr&lt;P_Point&gt; p_pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>calculate a global phase equilibrium for a point</td>
</tr>
<tr>
<td>Arguments</td>
<td>p_pt a shared_ptr of P_Point to be calculated</td>
</tr>
</tbody>
</table>

This function calculates the global phase equilibrium according to the calculation condition. Information on the calculated phase equilibrium is stored in p_pt.

4.3.3 Calculate a global phase equilibrium with initials

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_calc_point_global_with_initial_point(shared_ptr&lt;P_Point&gt; p_pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>calculate a global phase equilibrium for a point with initials</td>
</tr>
<tr>
<td>Arguments</td>
<td>p_pt a shared_ptr of P_Point to be calculated</td>
</tr>
</tbody>
</table>

This function calculates the global phase equilibrium with the initial conditions in p_pt. Information on the calculated phase equilibrium is also stored in p_pt. With the initial values in p_pt, the computational speed is usually faster.

4.3.4 Calculate a local phase equilibrium

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_calc_point_local_with_initial_point(shared_ptr&lt;P_Point&gt; p_pt, bool given_f=false)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>calculate a local phase equilibrium or parallel tangent equilibrium for a point with initials</td>
</tr>
<tr>
<td>Arguments</td>
<td>p_pt a shared_ptr of P_Point to be calculated given_f: false for a local phase equilibrium and true for a parallel tangent equilibrium</td>
</tr>
</tbody>
</table>
This function calculates the local phase equilibrium with the initial condition in \( p\_pt \). Initial values in \( p\_pt \) is required for this function. The calculated phase equilibrium is stored in \( p\_pt \).

### 4.3.5 Find the liquidus surface

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_find_liquidus_surface(string&amp; liquid_phase_name, shared_ptr&lt;P_Point&gt; p_pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>find a liquidus surface point for given composition</td>
</tr>
<tr>
<td>Arguments</td>
<td>liquid_phase_name phase name of liquid; p_pt a shared_ptr of P_Point for liquid and primary phases</td>
</tr>
</tbody>
</table>

This function calculates the liquidus surface temperature for a point of with fixed composition. PanEngine will find the stable liquidus surface.

### 4.3.6 Find the liquidus slopes

<table>
<thead>
<tr>
<th>Name</th>
<th>string pe_calc_liquidus_slope(string&amp; liquid_phase_name, string&amp; solvent_comp_name, shared_ptr&lt;P_Point&gt; p_pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>calculate liquidus slopes on the liquidus surface</td>
</tr>
<tr>
<td>Arguments</td>
<td>liquid_phase_name phase name of liquid; solvent_comp_name name of solvent component; p_pt a shared_ptr of P_Point for liquid with primary phase</td>
</tr>
</tbody>
</table>

This function calculates the liquidus slope along the directions of components on the liquidus surface for a point with given composition. The slope along the direction of component \( j \) is defined as

\[
s_j = \left( \frac{\partial T_{liq}}{\partial x_j} \right)_{x_i, i \neq j, j \neq \text{solvent}}
\]

where \( T_{liq} \) is the liquidus surface temperature and \( x_j \) is the mole fraction for the specified component \( j \). Since the molar fractions of components are dependent with each other by \( \sum_k x_k = 1 \), the solvent component must be specified. The slope along the direction of the solvent component will be treated as zero, \( s_{\text{solvent}} = 0 \).

For example, in a ternary \( A-B-C \) system, if the component \( A \) is selected as the solvent component, the slope of the liquidus surface along the direction of the component \( B \) is

\[
s_B = \left( \frac{\partial T_{liq}}{\partial x_B} \right)_{x_C}
\]

and the slope along the direction of the component \( C \) is

\[
s_C = \left( \frac{\partial T_{liq}}{\partial x_C} \right)_{x_B}
\]
The temperature change \( \delta T \) caused by the composition change of \((\delta x_B, \delta x_C)\) will be calculated by

\[
\delta T = s_B \delta x_B + s_C \delta x_C
\]  
(14)

The slopes in terms of weight fractions are also available, see test examples for detail.

Since PanEngine 2019, two new properties, \( \text{dxdT} \) and \( \text{dwdT} \), have been added into \text{P_Component} for solidification simulation. \( \text{dxdT} \) and \( \text{dwdT} \) represent the change rates of the molar fraction and weight fraction for a component in a phase with temperature during a solidification, respectively. These two variables can be found in the definition of class \text{P_Component} in the head file of Pan\_Global\_Def.h.

Since PanEngine 2020, another set of properties, \( \text{dxdT}\_S \) and \( \text{dxdT}\_L \), and the corresponding properties in weight fraction, \( \text{dwdT}\_S \) and \( \text{dwdT}\_L \), have been added into \text{P_Component} for solidification simulation. \( \text{dxdT}\_S \) and \( \text{dxdT}\_L \) represent the change rates of the molar fraction for a component in a solid phase and the liquid phase with temperature during a solidification, respectively, assuming that the solid phase is the only phase solidified from the liquid. In other words, these properties are calculated for the (local) equilibrium between the liquid phase and the only solid phase, excluding other solid phases even though they exist. Therefore, when there is more than one solid phase solidified from liquid, \( \text{dxdT}\_S \) and \( \text{dxdT}\_L \) will have different values from \( \text{dxdT} \). \( \text{dxdT}\_S \), \( \text{dxdT}\_L \), \( \text{dwdT}\_S \) and \( \text{dwdT}\_L \) are stored in the solid phase only. These variables can also be found in the definition of class \text{P_Component} in the head file of Pan\_Global\_Def.h.

When temperature decreases by \( \delta T \) during solidification, the composition changes of the solid phase and the liquid phase can be calculated by

\[
\delta x^j_s = \text{dxdT}\_S \delta T \quad (j = 1, 2, \ldots, c)
\]  
(15)

\[
\delta x^j_l = \text{dxdT}\_L \delta T \quad (j = 1, 2, \ldots, c)
\]  
(16)

Test Example 4 has the printout of those properties in the callback function, \text{progress}._4.

### 4.4 Function for Solidification Simulation

\text{PanEngine} has another API function for solidification simulations. There are two models available: lever rule and Scheil.

<table>
<thead>
<tr>
<th>Name</th>
<th>String pe_solidification_simulation (Solidification_Parameter&amp; s_param, Pan_Calculation&amp; calc, vector&lt;Solidification_Node&gt;&amp; solidification_result)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>simulate solidification with lever rule or Scheil model</td>
</tr>
<tr>
<td>Arguments</td>
<td>s_param parameters for solidification simulation; calc calculation condition; solidification_result solidification results</td>
</tr>
</tbody>
</table>

Details on how to use this function will be demonstrated in test Examples 4 and 5.
4.5 Thermodynamic Factors

In Pandat, thermodynamic factors are available with the format \( \text{ThF}(\cdot,\cdot) \), which is defined by

\[
\text{ThF}(x_i, x_j) = \frac{\partial \mu_i}{\partial x_j} \quad (i, j = 1, 2, \cdots, c)
\]

(17)

where \((x_1, x_2, \cdots, x_c)\) are treated as the independent compositional variables. If the component 1 is assumed to be the solvent component and its molar fraction \(x_1\) is taken as the dependent variable, the independent compositional variables now are \((X_2, X_3, \cdots, X_c)\). Here we use capital \(X\) to distinguish this set of variables from \((x_1, x_2, \cdots, x_c)\). Then we have

\[
\frac{\partial \mu_j}{\partial X_k} = \frac{\partial \mu_j}{\partial x_k} - \frac{\partial \mu_1}{\partial x_k} \quad (j, k = 2, 3, \cdots, c)
\]

(18)

\[
= \text{ThF}(x_j, x_k) - \text{ThF}(x_1, x_k) \quad (j, k = 2, 3, \cdots, c)
\]

(19)

Now let’s see how to express the second derivatives of Gibbs free energy in terms of the thermodynamic factors. If we use this compositional variable set \((X_2, X_3, \cdots, X_c)\), the first and second derivatives of \(G\) w.r.t. \(X_j\) are

\[
\frac{\partial G}{\partial X_j} = \sum_{i=2}^{c} \frac{\partial G}{\partial x_i} \frac{\partial x_i}{\partial x_j} + \frac{\partial G}{\partial x_1} \frac{\partial x_1}{\partial x_j} \quad (j = 2, 3, \cdots, c)
\]

(20)

\[
= \frac{\partial G}{\partial x_j} - \frac{\partial G}{\partial x_1} \quad (j = 2, 3, \cdots, c)
\]

(21)

\[
= \mu_j - \mu_1 \quad (j = 2, 3, \cdots, c)
\]

(22)

\[
\frac{\partial^2 G}{\partial X_j \partial X_k} = \sum_{i=2}^{c} \frac{\partial (\mu_j - \mu_1)}{\partial x_i} \frac{\partial x_i}{\partial x_k} \frac{\partial x_i}{\partial x_j} + \frac{\partial (\mu_j - \mu_1)}{\partial x_1} \frac{\partial x_1}{\partial x_k} \frac{\partial x_1}{\partial x_j} \quad (j, k = 2, 3, \cdots, c)
\]

(23)

\[
= \frac{\partial^2 G}{\partial X_j \partial X_k} - \frac{\partial (\mu_j - \mu_1)}{\partial x_1} \frac{\partial x_1}{\partial x_k} - \frac{\partial (\mu_j - \mu_1)}{\partial x_1} \frac{\partial x_1}{\partial x_j} \quad (j, k = 2, 3, \cdots, c)
\]

(24)

\[
= \text{ThF}(x_j, x_k) - \text{ThF}(x_1, x_k) - \text{ThF}(x_j, x_1) + \text{ThF}(x_1, x_1) \quad (j, k = 2, 3, \cdots, c)
\]

(25)

Thermodynamic factors, \( \text{ThF}(\cdot,\cdot) \), are stored in the member variable of \texttt{m.thermodynamic} in the class \texttt{PPhasePoint} in the head file of \texttt{Pan_Global_Def.h}. Test Example 8 shows the thermodynamic factors of the components in each stable phase point.

4.6 Hessian matrix of Gibbs free energy

From the second derivatives of Gibbs free energy in previous section, Pandat has the Hessian matrix of Gibbs free energy of a phase. Pandat also calculates the determinant, the eigenvalues and eigenvectors of the Hessian matrix.

Since there is one dependent molar fraction for the molar fraction variables \((x_1, x_2, \cdots, x_n)\), one of the components is selected as the dependent one. Without loss of generality, \(x_n\) is selected as the one, i.e., the
last component is considered as the solvent. Then, the second derivatives of Gibbs free energy of a phase form the Hessian matrix, which is an \((n-1) \times (n-1)\) symmetrical matrix.

\[
\text{HSN} = \begin{pmatrix}
\frac{\partial^2 G}{\partial x_1^2} & \frac{\partial^2 G}{\partial x_1 \partial x_2} & \frac{\partial^2 G}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_1 \partial x_{n-1}} \\
\frac{\partial^2 G}{\partial x_2 \partial x_1} & \frac{\partial^2 G}{\partial x_2^2} & \frac{\partial^2 G}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_2 \partial x_{n-1}} \\
\frac{\partial^2 G}{\partial x_3 \partial x_1} & \frac{\partial^2 G}{\partial x_3 \partial x_2} & \frac{\partial^2 G}{\partial x_3^2} & \cdots & \frac{\partial^2 G}{\partial x_3 \partial x_{n-1}} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 G}{\partial x_{n-1} \partial x_1} & \frac{\partial^2 G}{\partial x_{n-1} \partial x_2} & \frac{\partial^2 G}{\partial x_{n-1} \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_{n-1}^2}
\end{pmatrix}
\]

Its determinant is given by

\[
|\text{HSN}| = \begin{vmatrix}
\frac{\partial^2 G}{\partial x_1^2} & \frac{\partial^2 G}{\partial x_1 \partial x_2} & \frac{\partial^2 G}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_1 \partial x_{n-1}} \\
\frac{\partial^2 G}{\partial x_2 \partial x_1} & \frac{\partial^2 G}{\partial x_2^2} & \frac{\partial^2 G}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_2 \partial x_{n-1}} \\
\frac{\partial^2 G}{\partial x_3 \partial x_1} & \frac{\partial^2 G}{\partial x_3 \partial x_2} & \frac{\partial^2 G}{\partial x_3^2} & \cdots & \frac{\partial^2 G}{\partial x_3 \partial x_{n-1}} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 G}{\partial x_{n-1} \partial x_1} & \frac{\partial^2 G}{\partial x_{n-1} \partial x_2} & \frac{\partial^2 G}{\partial x_{n-1} \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_{n-1}^2}
\end{vmatrix}
\]

The determinant of Hessian matrix for phase \(f\) is available from \(\text{HSN}(\theta f)\). The value of \(\text{HSN}(\theta f)\) is independent of the selection of the solvent component. However, eigenvalues and eigenvectors are dependent on the selection of the solvent component.

A Hessian matrix has real eigenvectors and each eigenvalue has a corresponding eigenvector. The eigenvalues and their eigenvectors are available from \(\text{eVal}(\#*\theta f)\) and \(\text{eVec}(\#*\theta f)\).

Above Hessian matrix has eigenvalues of \(\text{eVal}(\#1\theta f)\), \(\text{eVal}(\#2\theta f)\), \(\ldots\), \(\text{eVal}(\#n-1\theta f)\). Each eigenvalue has an eigenvector. For example, \(\text{eVal}(\#1\theta f)\) has an eigenvector of \((\text{eVec}(C_1\#1\theta f), \text{eVec}(C_2\#1\theta f), \ldots, \text{eVec}(C_{n-1}\#1\theta f))\), where \(C_k\) is the name of the \(k^{th}\) component.

Test Examples 1 and 8 show how to get the values related to the Hessian matrix of Gibbs free energy of a phase.

### 4.7 Parallel Tangent Equilibrium

In the phase field modeling of microstructure evolution, the equilibrium among phases at a interface is not considered as a true phase equilibrium, where same component has same equilibrium chemical potential, i.e., common tangent (See Fig. 1(a)). Instead, the parallel tangent equilibrium is used at the interface during phase field modeling.

Parallel tangent equilibrium assumes that the difference of the chemical potential of same component in the phases at interface are same for all components, i.e.,

\[
\mu_i^\alpha - \mu_i^\beta = \mu_j^\alpha - \mu_j^\beta \quad (i, j = 1, 2, \ldots, c)
\]

This constraint is equivalent to that the tangent lines to Gibbs free energy curves of phases are parallel to each other, see Fig. 1(b). Therefore, this type of equilibrium is called parallel tangent equilibrium. For the
example of a parallel tangent equilibrium in Fig. 1(b), the overall composition, $x^\circ$, and the phase fractions, $f^\alpha$ and $f^\beta$, are given and then the parallel tangents (dash lines) are calculated to get the equilibrium composition of each phase, $x^\alpha$ and $x^\beta$.

Test Example 9 shows how to calculate a parallel tangent equilibrium.

![Common Tangent](image1)

![Parallel Tangent](image2)

(a) Common tangent equilibrium  
(b) Parallel tangent equilibrium

Figure 1: Two types of equilibria
5 Examples

This chapter demonstrates how to use PanEngine API functions with examples. There are seven test examples. The main program (main.cpp) is for user to select which example to run. All the source files (*.cpp) can be found in the folder /source/. Following sections give a brief review of each example.

5.1 Test Example 1

This first test example is in file of PanEngineTest_1.cpp. It uses a number of point calculations to demonstrate the following functions:

```cpp
// Define a PanEngine user pointer
user = definePanEngineUser(msg);

// Set PanEngine configuration
s = user->pe_set_configuration(config);

// Import a database
s = user->pe_import_database(db);

// Set calculation properties
s = user->pe_set_calc_property_type(calc_prop);

// Set calculation condition
s = user->pe_set_calculation_condition(calc);

// Calculate a global phase equilibrium
s = user->pe_calc_point_global(calc_point);

// Calculate a global phase equilibrium with initial
s = user->pe_calc_point_global_with_initial_point(calc_point);

// Delete a PanEngine user pointer
deletePanEngineUser(user);
```

It demonstrates how to set up calculation conditions. The example shows how to get the physical properties of molar volume and density. The Hessian matrix of Gibbs free energy of a phase, its determinant, eigenvalues and corresponding eigenvectors are also available. After `sys.ptr->m_condition->m_driving_force` is set to be true, the driving force for a phase with respect to an equilibrium state can be obtained.
5.2 Test Example 2

The second test example is in file of `PanEngineTest_2.cpp`. It mainly demonstrates how to use a “for” loop to calculate multiple points using the function `pe_calc_point_global_with_initial_point`. However, the first point has to be calculated with the function `pe_calc_point_global` to get initial point.

5.3 Test Example 3

The third test example is in file of `PanEngineTest_3.cpp`. It demonstrates how to calculate local phase equilibria for a point at different temperatures and multiple points with different compositions using the function of `pe_calc_point_local_with_initial_point`. Similar to the previous example (Test Example 2), the first point has to be calculated with the function `pe_calc_point_global` to get initial point.

5.4 Test Example 4

The fourth test example is in file of `PanEngineTest_4.cpp`. Most of the codes in this file are for setting solidification conditions and parameters, such as database name, units, alloy compositions, solvent component, solidification model, and step size. The major function used in this test example is `pe_solidification_simulation`. The solidification result could be output to a file with the function

```
string output_result(string file_name, vector<Solidification_Node>& solidification_result)
```

which is defined in this example file.

The function

```
bool progress_4(char* msg, Solidification_Parameter& sp, vector<Solidification_Node>& node)
```

is a callback function which will be called from `PanEngine` to send back the intermediate results during simulation.

5.5 Test Example 5

The fifth test example is in file of `PanEngineTest_5.cpp`. This example is also for solidification simulation and similar to the previous one. It demonstrates how to simulate solidification processes for multiple alloys with different compositions.

The function

```
bool progress_5(char* msg, Solidification_Parameter& sp, vector<Solidification_Node>& node)
```

is the callback function.

5.6 Test Example 6

The sixth test example is in file of `PanEngineTest_6.cpp`. This example demonstrates how to find the liquidus surface and calculate liquidus slopes using the following two functions:

```
s=user-&gt;pe_find_liquidus_surface(liquid_name, calc_point)
```

and
s=user->pe_calc_liquidus_slope(liquid_name, solvent_name, p_pt)

The example uses a double “for” loops to calculate the liquidus surface points and slopes for a series of points in the Al-Mg-Zn system.

5.7 Test Example 7

The seventh test example is in file of PanEngineTest_7.cpp. This example uses three stages to locate a point on the liquidus surface in the Al-Mg-Zn ternary system where the temperature reaches the minimum. The first stage of search uses the step size of dx=0.1. The second stage uses step size of dx=0.01 to search in the compositional neighborhood of the found composition in the first stage. And the third stage uses the step size of dx=0.001 to search in the compositional neighborhood of the found composition in the second stage. Even through this method is of brute-force, it works fine for such a simple problem. The major function used in this example is pe_find_liquidus_surface.

5.8 Test Example 8

The eighth test example is in file of PanEngineTest_8.cpp. This example demonstrates how to obtain the thermodynamic properties such as thermodynamic factors and Hessian matrix, and the kinetic properties such as mobility and diffusivity. The example calculates the thermodynamic factor, Hessian matrix, mobility, and diffusivity for an alloy in the Fe-Ni-Cr system. For the calculation of Hessian matrix, mobility and diffusivity, the name of the solvent component has to be defined, as is given below,

sys_ptr->m_condition->m_solvent = string("Ni");

which is in the function of set_calculation_condition_8.

The example shows how to extract those properties from a calculated point.

5.9 Test Example 9

This last test example is in file of PanEngineTest_9.cpp. This example demonstrates how to calculate a parallel tangent equilibrium.

The example calculates the two-phase parallel tangent equilibrium between Fcc and gamma_double_prime (γ'') in a pseudo-ternary Ni-Al-Nb. The common tangent equilibrium is first calculated from

s = user->pe_calc_point_global(calc_point);

to get the initial values of the parallel tangent equilibrium. Then, reset the phase fractions of the two phases by

calc_point->m_ppt[0]->m_f = 0.5;
calc_point->m_ppt[1]->m_f = 0.5;

Call the function of calculating point local with initial point

s = user->pe_calc_point_local_with_initial_point(calc_point, true);
with a true value for the second argument to calculate the parallel tangent equilibrium. When the second argument is true, the function will calculate the parallel tangent equilibrium. If the second argument is false, the function will calculate the normal local common tangent equilibrium.

Example 9 continues with a set of randomly generated temperatures, overall compositions and phase fractions to calculate the parallel tangent equilibria.