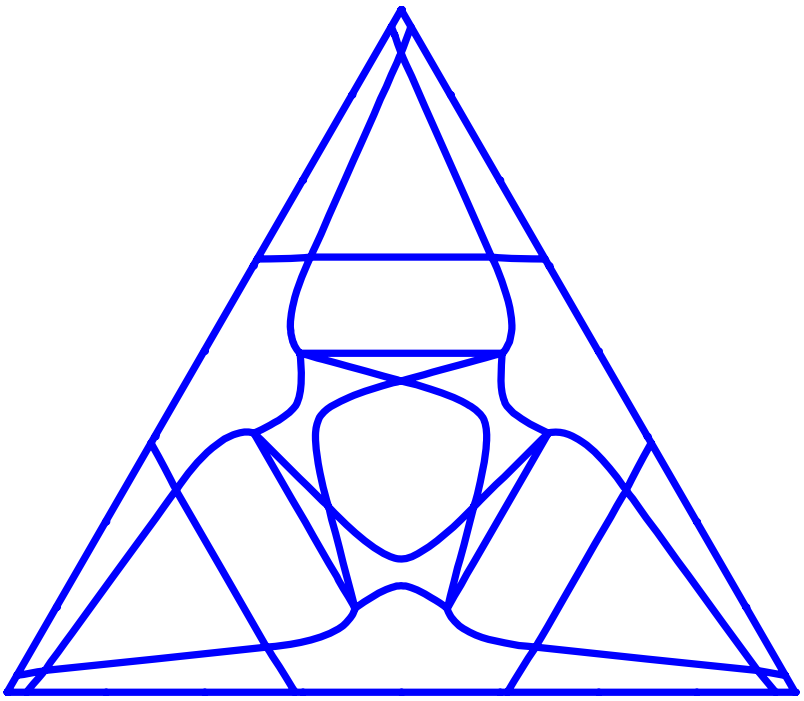
PanEngine™ 6.0
User's Guide



PanEngine 6.0 User's Guide

Getting Help

CompuTherm LLC is committed to providing you with the best possible product support. Please select from the following options:

✓ Web Site

www.computherm.com

✓ E-Mail

info@computherm.com

✓ Phone

+1 (608) 274-1414

✓ FAX

+1 (608) 274-6045

✓ Mail

CompuTherm LLC
437 S. Yellowstone Dr., Suite 217
Madison, WI 53719
USA

PanEngine 6.0 User's Guide

Contents

1. Introduction to PanEngine

What is PanEngine?

PanEngine is a dynamically linked library (DLL) for multi-component thermodynamics and phase equilibrium calculations. It has an application program interface (API) for a user's custom programs to access the functions in PanEngine.

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 create custom software to meet their specific needs in thermodynamics and phase equilibria calculations.

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
- Heat treatment simulations
- Other applications where phase equilibrium information and thermodynamic properties are needed, such as the cellular automaton (CA) and phase field simulation

Advantages of PanEngine

PanEngine finds the *correct, stable* phase equilibria without requiring the user to guess initial values. This is especially important because it is very difficult for the user to enter initial values and verify results when a custom software program needs stable phase equilibrium information for thousands of points. For multi-component systems, it is almost impossible to guess initial values for stable phase equilibria.

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.

- Load database
- Set components
- Suspend/restore phases
- Set calculation conditions
- Calculate stable equilibrium
- Find liquidus surface
- Calculate liquidus slope
- Calculate partition coefficient
- Calculate thermodynamic factor
- Simulate solidification using Scheil or lever rule model

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 printer parallel port or USB port. PanEngine consists of several different types of files. As shown in Table 2.1, the PanEngine thermodynamic calculation interface includes two DLL files, two library files, three header files and some example programs.

Table 2.1 List of PanEngine Files

File Name	Comment
`PanEngine.dll`	PanEngine dynamically linked library
`FrontDLL.dll`	Numerical methods related dynamically linked library
`PanEngine.lib`	PanEngine library file
`FrontDLL.lib`	FrontDLL library file
`PanEngine.h`	PanEngine header file
`std.h`	Standard header file
`user.h`	User API header file
`main.cpp`	Main program
`misc.cpp`	Miscellaneous functions
`PanEngineTest.cpp`	PanEngine test examples
`AlMgZn.tdb`	Example database file in tdb format
`AlMgZn.pdb`	Example database file in pdb format
`EngineTest.dsw`	VC 6.0 project workspace file
`EngineTest.dsp`	VC 6.0 project file

The installation of PanEngine is rather straightforward. Simply copy all the files in the folder *PanEngine6* in the PanEngine distribution CD to any working

directory where the user intends to place his/her own codes for applications. The library files can also be located in any other area, and can be accessed by specifying their appropriate paths in the application program codes. The current PanEngine can run on Windows 95, 98, ME, NT 4.0, 2000, XP or higher.

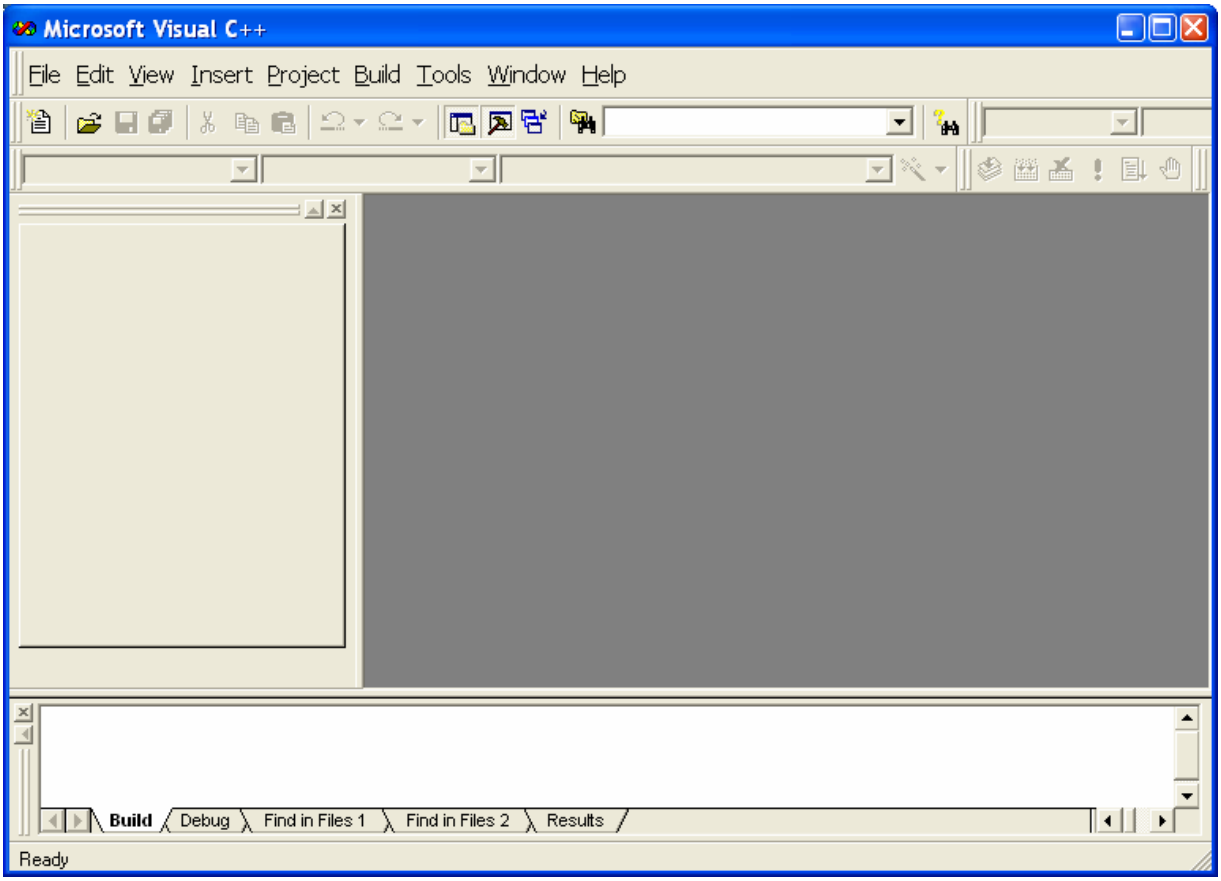
The recommended programming environment with PanEngine is Microsoft Visual Studio C++ 6.0, which is the programming environment in which PanEngine was created. If a user uses a different C++ compiler, the PanEngine workspace file and project file may not work and then a completely new workspace and project files or make file need to be constructed by the user.

2.2 Getting Started with PanEngine Using Microsoft Visual C++ 6.0

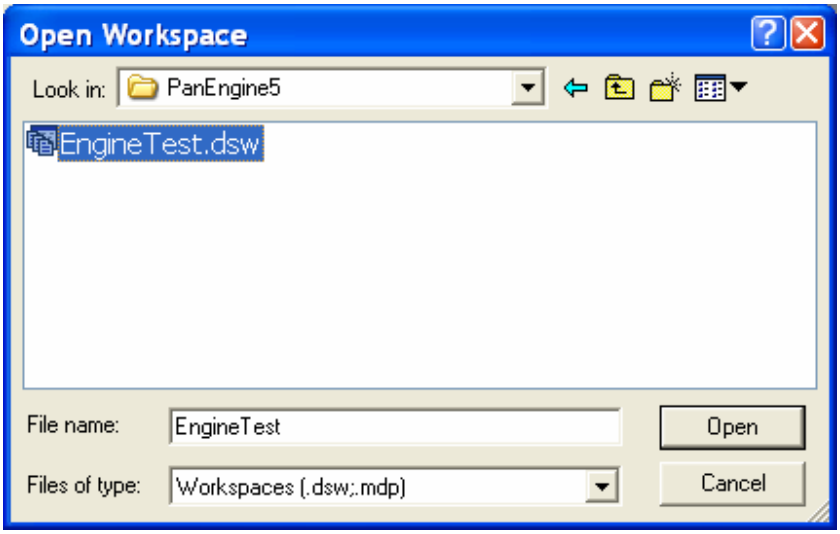
We assume that a full version of Microsoft Visual Studio C++ 6.0 is installed on the user's computer. Follow the steps below to run the PanEngine test examples.

1. Attach the CompuTherm hardware dongle to the computer.

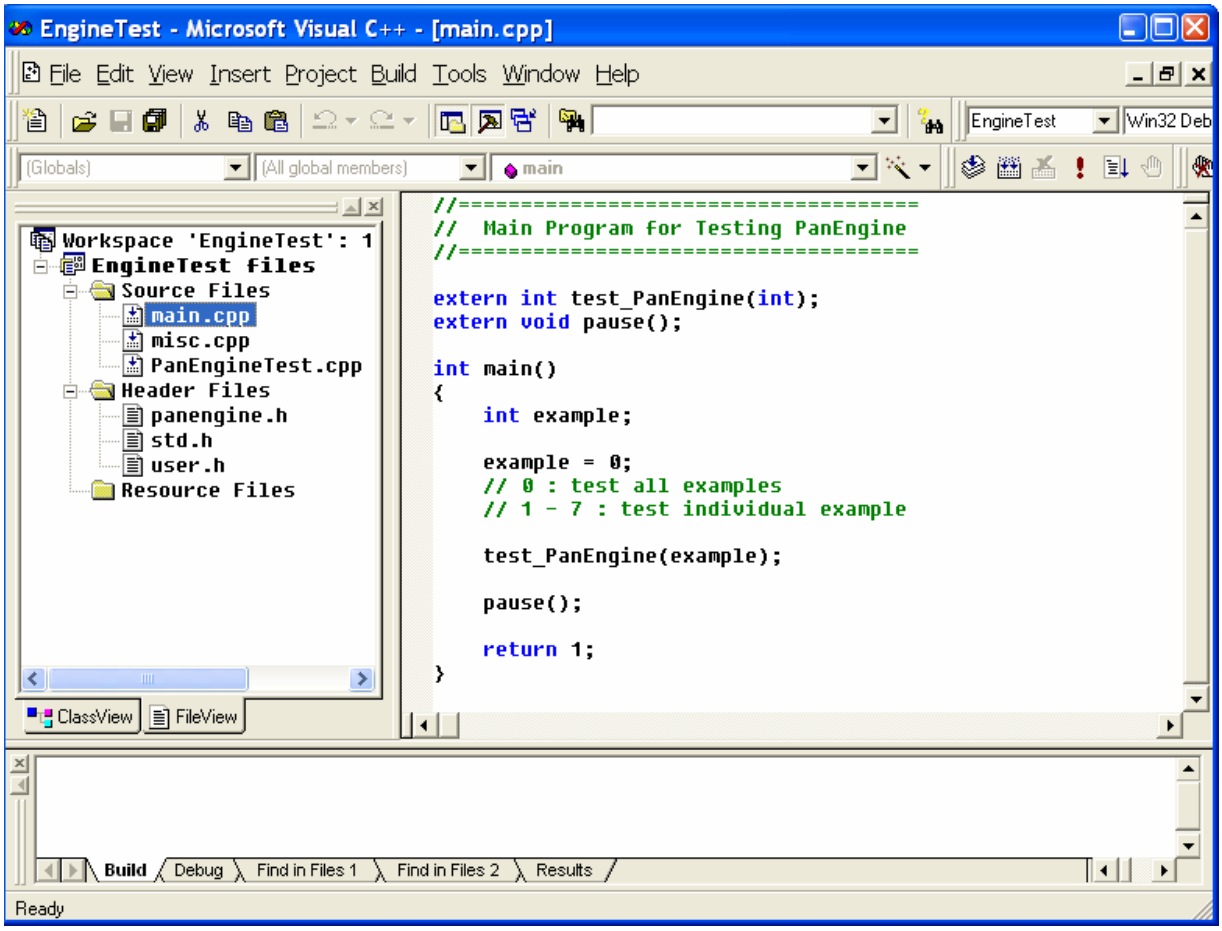
2. Start Microsoft Visual C++ 6.0.



3. On the main menu of Microsoft Visual C++ 6.0, click **`File`** → **`Open Workspace`**. Go to the folder `PanEngine6` (in the user's hard drive) and open `EngineTest.dsw`.

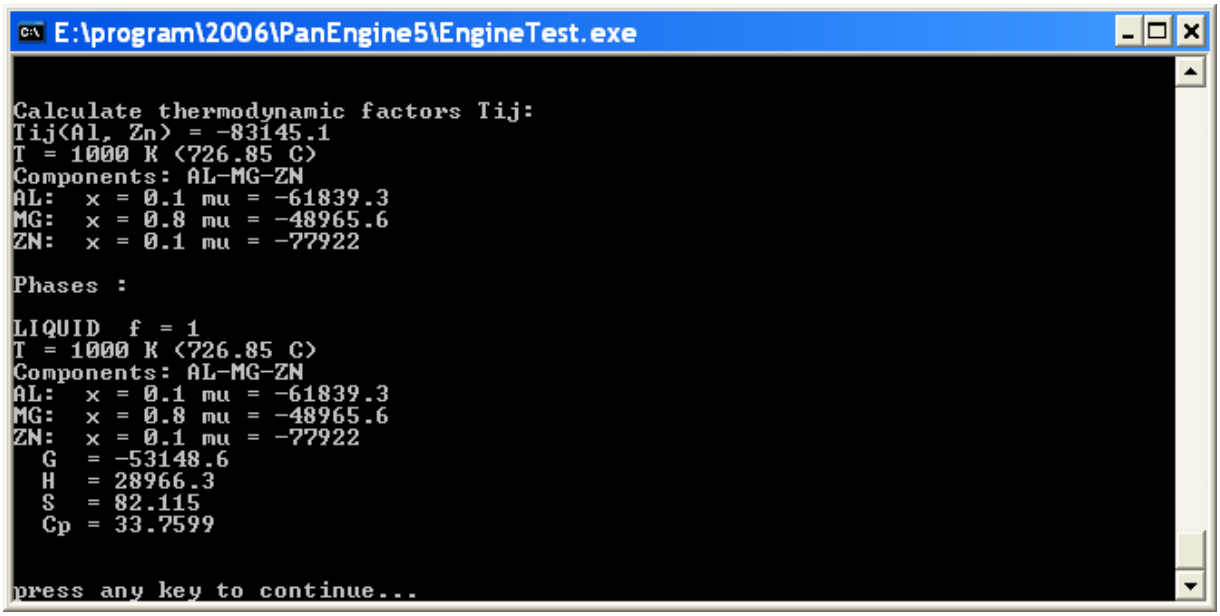


4. Go to File View in the left window of the VC6 main window and double click on the file `main.cpp` in the folder of Source Files.



5. Rebuild EngineTest by click **`Build`** → **`Rebuild All`**.

6. Press F5 to run the test examples. A DOS window will pop up and intermediate results of the calculations will be shown in this window. The final status of the window looks like the following one, except that the path name on the top of the window will depend on the location of PanEngine6.

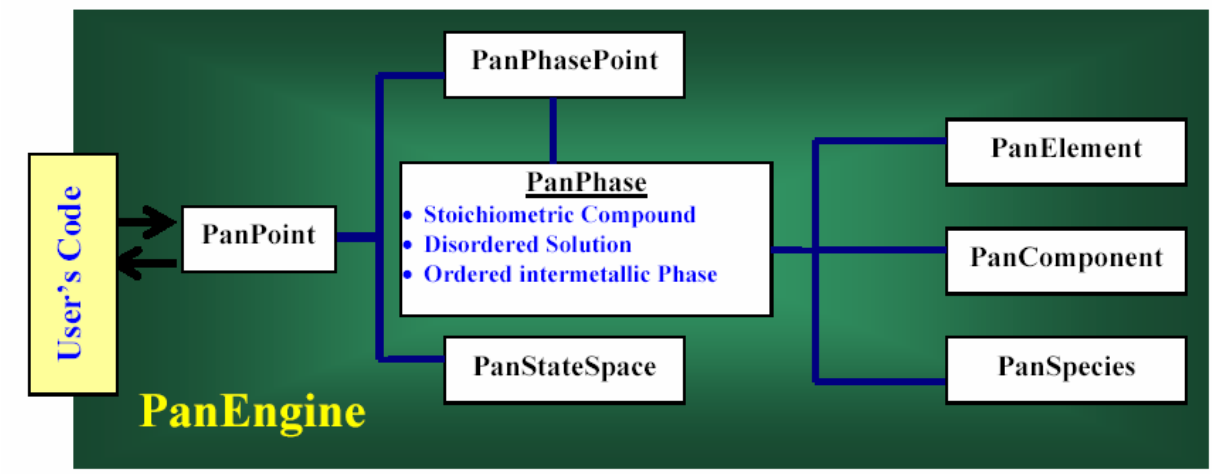


7. Press any key and return to the VC6 main window.

The default option for testing PanEngine is to test all seven examples. To test an individual example, change the integer variable **`example`** in main function (see the image in Step 4).

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.

In ***PanEngine***, as shown in the above diagram, ***PanPoint*** refers to a system with a specified composition at a certain temperature and pressure. A ***PanPoint*** object is directly interfaced with the user's code. The user can change the conditions (temperature or overall compositions) of the system through a ***PanPoint*** object and get back the thermodynamic properties and phase equilibrium information under the newly specified conditions. The stable (or metastable) phase equilibria information of the system (such as phase fractions, composition of each phase, and thermodynamic properties for each phase) are described by ***PanPhasePoint*** Class. General information about the system, such as alloying components, alloy overall composition and temperature, are stored in ***PanStateSpace*** class.

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

$$G = \sum_{i=1}^{n} x_i G_i^{o,\phi} + G_f$$

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

3.1.2 Disordered solution phase

The Gibbs energy of a disordered solution phase is expressed as

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

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

$$G^{ex,\phi} = \sum_{i,j=1(i\neq j)}^{n} x_i x_j \sum_{k=0}^{m} L_{(i,j)}^{k} (x_i - x_j)^k + \sum_{i,j,l=1(i\neq j\neq l)}^{n} x_i x_j x_l \sum_{k=i,j,l} L_k V_k$$

where the first term represents the binary interaction terms, the second represents ternary interactions. The $L_{(i,j)}^{k}$'s are binary interaction parameters for the *i-j* binary and the L_k's are ternary interaction parameters. V_k is defined as

$$V_k = x_k + \frac{1 - \sum_{p=i,j,l} x_p}{n}$$

3.1.3 Ordered intermetallic phase using the compound energy formalism

The Gibbs energy of an ordered intermetallic phase is described as

$$G = G^{ref} + G^{id} + G^{ex}$$

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,

$$G^{ref} = \sum y_p^i y_q^j \cdots y_s^l G^o_{(p:q:\ldots:s)}$$

G^{id} is the ideal mixing term, which assumes the random mixing of species on each sublattice,

$$G^{id} = \sum_{i=1}^{l} f_i \sum_{p=1}^{m} y_p^i \ln(y_p^i)$$

G^{ex} is also expressed as a function of species concentrations with the sublattice L parameters being the numerical coefficients in the contributing terms,

$$G^{ex} = \sum y_p^i y_q^i y_r^j L_{(p,q:r)}$$

where

$$L_{(p,q:r)} = \sum_{\nu} L^{\nu}_{(p,q:r)} (y_p^i - y_q^j)^{\nu}$$

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 classses are included in the file *PanEngine.h*. The major PanEngine classes with frequently used functions and properties are described below.

3.2.1 class PanElement

Class Name	PanElement
Definition	an element from the periodic table, such as *Al*

Public Functions	`PanElement()`
	`PanElement(const PanElement&)`
	`~PanElement()`

Public Properties	`string name` // element name
Comments	define an element from the periodic table

3.2.2 class PanComponent

Class Name	PanComponent
Definition	a component is made up of one or more elements, for example: *Al* or *FeO*

Public Functions	`PanComponent()`
	`PanComponent(const PanComponent&)`
	`~PanComponent()`

Public Properties	`string name` // component name `vector <PanElement> ele` //element(s) in the component `double mu` // chemical potential of the component `double h` // enthalpy of the component `double s` // entropy of the component `double cp` // heat capacity of the component `double x` // atomic fraction of this component `double wt` // weight fraction of this component
Comments	PanComponent holds information for a component

3.2.3 class PanSpecies

Class Name	PanSpecies
Definition	species can be made up of one or more components, for example: *O2*

Public Functions	`PanSpecies()`
	`PanSpecies(const PanSpecies&)`
	`~PanSpecies()`

Public Properties	`string name` // species name `vector <PanComponent> comp` // components in PanSpecies `vector <double> b` // stoichiometric corresponding to `comp`
Comments	associate model uses PanSpecies to define the species of the associates

3.2.4 class PanStateSpace

Class Name	PanStateSpace
Definition	PanStateSpace describes temperature and composition of a system or a phase

Public Functions	`PanStateSpace()`
	`~PanStateSpace()`
	`PanStateSpace(const PanStateSpace&)`
	
	`void setX(string &name, double x)` // set composition in mole fraction for specified component
	`double getX(string &name)` // return composition in mole fraction for specified component
	`void setWt(string &name, double wt)` // set composition in weight fraction for specified component
	`double getWt(string &name_)` // return composition in mole fraction for specified component

Public Properties	`vector <PanComponent> comp` // components in a system or components in a phase `double T` // system temperature
Comments	most of calculation conditions are set through this class

3.2.5 class PanPhase

Class Name	PanPhase
Definition	information about a phase: the name, the model type, etc.

Public Functions	`PanPhase()`
	`~PanPhase()`
	`PanPhase(const PanPhase&)`

Public Properties	`string name` // name of the phase `MODEL_TYPE model` // thermodynamic model for the phase
Comments	phase names are defined in the database

3.2.6 class PanPhasePoint

Class Name	PanPhasePoint
Definition	more information for a phase after a calculation: the state space, thermodynamic properties, species concentrations, etc.

Public Functions	`PanPhasePoint()`
	`~PanPhasePoint()`
	`PanPhasePoint(const PanPhasePoint&)`
	`int getNumComp()` // number of components in the phase

Public Properties	`string name` // phase name `PanStateSpace st` // statespace of the phase point `double G` // Gibbs free energy `double H` // enthalpy of the phase `double S` // entropy `double Cp` // heat capacity `vector <double> y` // species fraction on sublattice or internal variables `vector <string> yName` // species name `double fraction` // phase fraction (default in atomic scale)
Comments	the objects of this class will be created by PanEngine after calculation

3.2.7 class PanPoint

Class Name	PanPoint
Definition	an equilibrium state with one or more phase points (PanPhasePoint)

Public Functions	`PanPoint()`
	`~PanPoint()`
	`PanPoint(const PanPoint&)`
	`void setT(double)` // set temperature (in K)
	`void setUnitX(bool)` // true for mole fraction, false for weight fraction
	`void setX(char *compName, double x)` //set composition in mole fraction for a specified component
	`void setWt(char *compName, double wt)` // set composition in weight fraction for a specified component

Public Properties	`PanStateSpace st` // statespace for a PanPoint `vector <PanPhasePoint> phasePt` // information for each phase `double fs` // solid fraction solidified so far (including this step) `double fl` // liquid fraction remained (after this step) `double Hm` // summation of `H` for the system (solid + liquid phases) `double Cp` // summation of `Cp` for the system (solid + liquid phases)
Comments	for solidification, a PanPoint includes the fraction 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 `User` in *user.h*, except for the two global functions used for defining a `User` pointer and deleting an existing `User` pointer. These functions can be divided into four categories according to their purposes:

1. User Pointer
   - define a `User` pointer and initialize it
   - delete an existing `User` pointer

2. System
   - load a thermodynamic database
   - define a sub-system
   - suspend one phase
   - restore one phase
   - suspend all phases
   - restore all phases
   - set an option

3. Point Calculation
   - define a `PanPoint` pointer and initialize it
   - find the globally stable equilibrium
   - find the metastable equilibrium
   - find a local thermodynamic properties of a phase
   - other functions
     i find the liquidus surface
     ii calculate the liquidus slope
     iii calculate the equilibrium partition coefficient
     iv calculate the thermodynamic factor

4. Solidification Simulation
   - lever rule
   - Scheil model

The following table summarizes the functions of PanEngine API. These functions will be explained in detail in the next section.

Table 4.1 List of PanEngine API Functions

Functions		Comments
User Pointer	`extern "C" DECLEXPORT User* pe_defineUser( char* msg)`	define a User object
	`extern "C" DECLEXPORT void pe_deleteUser(User *pUser)`	delete a User object
System	`bool pe_readTDBFile(char* tdbFileName, char* err_msg = NULL)`	load thermodynamic database
	`bool pe_readPDBFile(char* pdbFileName, char* err_msg = NULL)`	load encrypted thermodynamic database
	`bool pe_defineSubSystem(int numComponents, ...)`	define a sub-system with *n* components (*A*, *B*, *C*, …)
	`bool pe_printMasterSystem(char* fileName, char* msg = NULL)`	output master system (database) information to a file
	`bool pe_printSubSystem(char* fileName, char* msg = NULL)`	output subsystem information to a file
	`bool pe_suspendPhase(char* phaseName, char* msg = NULL)`	suspend one phase
	`pe_suspendAllPhases(char* msg = NULL)`	suspend all phases
	`bool pe_restorePhase(char* phaseName, char* msg = NULL)`	restore one phase
	`bool pe_restoreAllPhases(char* msg = NULL)`	restore all phases
	`bool pe_setOption(int id, int value, char* msg = NULL)`	set option
Point Calculation	`bool pe_initializePanPoint(PanPoint* panPt, char* msg = NULL)`	initialize a PanPoint
	`bool pe_stateSpacePointConversion_X_Wt( PanStateSpace* panSt, char* msg = NULL)`	convert a state space between mole fraction and weight fraction
	`bool pe_findStablePoint(PanPoint* panPt, char* msg = NULL)`	find globally stable equilibrium
	`bool pe_findLocalPoint(PanPoint* panPt, char* msg = NULL)`	find meta-stable equilibrium
	`bool pe_getThermodynamicProperty (PanPoint* panPt, char* phaseName, char* msg = NULL)`	find local thermodynamic properties of a phase
	`bool pe_findLiquidusSurface(PanPoint* panPt, char* msg = NULL) = 0;`	find liquidus surface
	`bool pe_getLiquidusSlope(PanPoint* panPt, char* compName, double& slope, char* msg)`	calculate liquidus slope
	`bool pe_getK(PanPoint* panPt, char* phaseName1, char* phaseName2, char* compName, double& k, char* msg)`	calculate equilibrium partition coefficient
	`virtual bool pe_getThermodynamicFactor (PanPoint* panPt, char* compName_solvent, char* compName_i, char* compName_j, double& Tij, char* msg)`	calculate thermodynamic factor of one phase
Solidification	`bool pe_solidificationSimulation (SOLIDIFICATION_MODEL model, PanPoint* panPt, ProgressPtr prog_ptr, char* msg = NULL)`	simulate solidification with lever rule or Scheil model

4.1 Functions for User Pointer

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

4.1.1 Define a User pointer

Full Name	`extern "C" DECLEXPORT User* pe_defineUser(char* msg)`
Purpose	define a `User` pointer and initialize it
Arguments	`msg` message returned from PanEngine

A PanEngine `User` 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 User pointer

Full Name	`extern "C" DECLEXPORT void pe_deleteUser(User *pUser)`
Purpose	delete a `User` pointer after all calculations are done
Arguments	`User *pUser` a defined and initialized `User` pointer

After a `User` 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 loading a thermodynamic database, suspending or restoring phases.

4.2.1 Load a thermodynamic database (tdb)

Name	`bool pe_readTDBFile(char* tdbFileName, char* msg = NULL)`
Purpose	load a thermodynamic database file in *tdb* format
Arguments	`tdbFileName` database file name; `msg` message

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.

4.2.2 Load a thermodynamic database (pdb)

Full Name	`bool pe_readPDBFile(char* pdbFileName, char* msg = NULL)`
Purpose	load a thermodynamic database file in encrypted `pdb` format
Arguments	`tdbFileName` database file name; `msg` message

Thermodynamic databases in `pdb` format are encrypted and cannot be viewed nor edited. This function reads the `pdb` type of database.

4.2.3 Define a subsystem

Full Name	`bool pe_defineSubSystem(int numComponents, ...)`
Purpose	define a subsystem for current calculation
Arguments	`NumComponents` number of components to be selected; `...` component names

After a thermodynamic database is successfully loaded, the components are selected for the subsystem on which the subsequent calculations will be performed. At least two components have to be selected.

4.2.4 Output the master system information

Full Name	`bool pe_printMasterSystem(char* fileName, char* msg = NULL)`
Purpose	output master system (database) information to a file
Arguments	`fileName` file name to output; `msg` message

This function outputs the master system information, i.e., the database loaded into PanEngine, into a text file with a given name. If the database is a type of `pdb`, only information on phase names and phase types will be written to the file.

4.2.5 Output the sub-system information

Full Name	`bool pe_printSubSystem(char* fileName, char* msg = NULL)`

Purpose	output the currently selected sub-system information to a file
Arguments	`fileName` file name to output; `msg` message

This function outputs the currently selected sub-system information into a text file with a given name. If the database is of the type `pdb`, only the information on phase names and phase types will be written to the file.

4.2.6 Suspend a phase

Full Name	`bool pe_suspendPhase(char* phaseName, char* msg = NULL)`
Purpose	suspend one phase
Arguments	`phaseName` the name of the phase to be suspended; `msg` message

Some calculations require some phases to be suspended. This function suspends the phase with a specified phase name. Only one phase can be suspended with this function. If the phase with the name does not exist, this function returns *false*.

4.2.7 Suspend all phases

Full Name	`pe_suspendAllPhases(char* msg = NULL)`
Purpose	suspend all phases
Arguments	`msg` message

All the phases in the subsystem will be suspended. Combination of this function with the following function suspends all but one phase.

4.2.8 Restore a phase

Full Name	`bool pe_restorePhase(char* phaseName, char* msg = NULL)`
Purpose	restore one phase
Arguments	`phaseName` the name of the phase to be restored; `msg` message

This function restores a phase with the given phase name which has previously been suspended. If the phase with the name does not exist in database, this function returns *false*.

4.2.9 Restore all phases

Full Name	`bool pe_restoreAllPhases(char* msg = NULL)`
Purpose	restore all phases
Arguments	`msg` message

This function restores all phases in the subsystem.

4.2.10 Set option

Full Name	`bool pe_setOption(int id, int value, char* msg = NULL)`
Purpose	set calculation options
Arguments	`id` option index; `value` option value; `msg` message

This function sets the calculation options. In the current version, only the global search algorithm has the option to set. `id` is *4001* for the global search algorithm option and `value` takes a value of *0* or *1*. The default value for the global search algorithm option is *0*. The option value *1* is for a more intensive global search.

4.3 Functions for Point Calculation

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

4.3.1 Initialize a point

Full Name	`bool pe_initializePanPoint(PanPoint* panPt, char* msg = NULL)`
Purpose	Initialize a PanPoint
Arguments	`panPt` PanPoint pointer to be initialized; `msg` message

Calculation results will be stored in an object of class `PanPoint`. Before using a

`PanPoint` object or a pointer, it has to be initialized with this function.

4.3.2 Convert compositions between units

Full Name	`bool pe_stateSpacePointConversion_X_Wt( PanStateSpace* panSt, char* msg = NULL)`
Purpose	convert compositions between mole fraction and weight fraction
Arguments	`panSt` PanStateSpace pointer; `msg` message

This function converts the compositions in `*panSt` between mole fraction and weight fraction. If `panSt→unitX` is `true`, the composition is given in the unit of mole fraction and this function converts the composition from the unit of mole fraction to that of weight fraction. If `panSt→unitX` is `false`, the composition is given in the unit of weight fraction and this function converts the composition from the unit of weight fraction to that of mole fraction.

4.3.3 Find the globally stable equilibrium

Full Name	`bool pe_findStablePoint(PanPoint* panPt, char* msg = NULL)`
Purpose	find globally stable equilibrium
Arguments	`panPt` PanPoint pointer; `msg` message

Before calling this function to calculate the stable phase equilibrium, the state space for this point, *i.e.*, the calculation conditions, have to be defined. See the test examples in the next section for how to define such calculation conditions.

4.3.4 Find a metastable equilibrium

Full Name	`bool pe_findLocalPoint(PanPoint* panPt, char* msg = NULL)`
Purpose	find metastable equilibrium
Arguments	`panPt` PanPoint pointer; `msg` message

This function uses a local optimization algorithm for solving the phase equilibrium. It requires that *`panPt`* have initial values from a previous calculation or the user's input. The example given in Section 5.3 uses the function of `pe_findStablePoint` to find the initial values for the starting point in a series of metastable phase equilibrium calculations. However, the calculated phase equilibrium can be globally stable one or a metastable one, which depends upon the initial values from the previous calculation.

4.3.5 Find the local thermodynamic properties

Full Name	`bool pe_getThermodynamicProperty(PanPoint* panPt, char* phaseName, char* msg = NULL)`
Purpose	find local thermodynamic properties of a phase
Arguments	`panPt` PanPoint pointer; `phaseName` phase name; `msg` message

This function calculates the thermodynamic properties for a given phase point with the default reference states defined in the database. It does not check the possibility of a miscibility gap being formed by the phase.

4.3.6 Find the liquidus surface

Full Name	`bool pe_findLiquidusSurface(PanPoint* panPt, char* msg = NULL)`
Purpose	find liquidus surface temperature for given composition
Arguments	`panPt` PanPoint pointer; `msg` message

This function calculates the liquidus surface temperature for a given point of selceted composition. A good initial liquidus surface temperature value helps to find the liquidus surface faster. A poor initial value does not give difficulty in finding the liquidus temperature. If the initial temperature is not provided, a default value of 1000K will be used in PanEngine. No matter what the initial temperature is, PanEngine will find the stable liquidus surface.

4.3.7 Find the liquidus slope

Full Name	`bool pe_getLiquidusSlope(PanPoint* panPt, char* compName, double& slope, char* msg)`
Purpose	calculate liquidus slope for a component on the liquidus surface
Arguments	`panPt` PanPoint pointer; `compName` component name; `slope` slope; `msg` message

This function calculates the liquidus slope along the direction of a specified component on the liquidus surface for a point with given composition. The slope is defined as $\frac{\partial T^{liq}}{\partial x_j}$ where T^{liq} is the liquidus surface temperate and x_j is the mole fraction for the specified component *j*.

4.3.8 Find a partition coefficient

Full Name	`bool pe_getK(PanPoint* panPt, char* phaseName1, char* phaseName2, char* compName, double& k, char* msg)`
Purpose	calculate the partition coefficient k between `phaseName1` and `phaseName2` for a specified component `compName` at given state space
Arguments	`panPt` PanPoint pointer; `phaseName1` first phase name; `phaseName2` second phase name; `compName` component name; `k` partition coefficient; `msg` message

This function calculates the partition coefficient for a component between two phases. The partition coefficient is defined as $k = \frac{x_j^{\phi 1}}{x_j^{\phi 2}}$, where $x_j^{\phi 1}$ and $x_j^{\phi 2}$ are the equilibrium compositions for the component j in the phases $\phi 1$ and $\phi 2$, respectively.

4.3.9 Calculate a thermodynamic factor

Full Name	`bool pe_getThermodynamicFactor (PanPoint* panPt, char* compName_solvent, char* compName_i, char* compName_j, double& Tij, char* msg)`
Purpose	calculate the thermodynamic factor `Tij`
Arguments	`panPt` PanPoint pointer; `compName_solvent` component name for solvent; `compName_i` name for component `i`; `compName_j` name for component `j`; `Tij` thermodynamic factor; `msg` message

The thermodynamic factor is defined as $T_{ij} = \frac{\partial \mu_i}{\partial x_j}$.

4.4 A Function for Solidification Simulation

PanEngine has another API function used in solidification simulations. There are two models available: lever rule and Scheil. During each simulation step, results can be monitored through a calculation progress function pointer.

Full Name	`bool pe_solidificationSimulation (SOLIDIFICATION_MODEL model, PanPoint* panPt, ProgressPtr prog_ptr, char* msg = NULL)`
Purpose	solidification simulation
Arguments	`model` simulation model; `panPt` point with specified composition; `prog_ptr` calculation progress function pointer; `msg` message

Details on how to use this function will be demonstrated in test example 5.

5. Examples

This chapter demonstrates how to use PanEngine API functions. All the examples given in this chapter can be found in the file *PanEngineTest.cpp*.

5.1 Define User Pointer, Load Database, etc.

```
//================================
//    PanEngine Test Example 1
//
// define a PanEngine user pointer
// read database
// define subsystem
// suspend phases
// restore phases
// delete a PanEngine user pointer
//================================

int PanEngine_Test_1()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //========================================
        // Load thermodynamic database in tdb Format
        //========================================
        char* tdbFileName = "AlMgZn.tdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readTDBFile(tdbFileName, msg);

        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //========================================
        // Define a subsystem for current calculation
        //========================================
        cout << endl << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }
```

```
        //=====================================================================
        // Write master system and selected subsystem information into files
        //=====================================================================
        cout << "Write database information to master.dat" << endl;
        user->pe_printMasterSystem("master.dat");

        cout << "Write subsystem database information to sub.dat" << endl;
        user->pe_printSubSystem("sub.dat");

        //=================================
        // Suspend all phases in subsystem
        //=================================
        cout << "\nSuspend all phases" << endl;
        result = user->pe_suspendAllPhases(msg);
        if (!result) {
                cout << "Error:  " << msg << endl;
                return 0;
        } else {
                cout << msg << endl;
        }

        //===================
        // Restore all phases
        //===================
        cout << "\nRestore all phases" << endl;
        result = user->pe_restoreAllPhases(msg);
        if (!result) {
                cout << "Error:  " << msg << endl;
                pause();
                return 0;
        } else {
                cout << msg << endl;
        }

        //==================
        // Suspend one phase
        //==================
        cout << "\nSuspend the fcc phase" << endl;
        result = user->pe_suspendPhase("FCC_A1", msg);
        if (!result) {
                cout << "\n\nError: " << msg << endl;
                return 0;
        }

        //==================
        // Restore one phase
        //==================
        cout << "\nRestore the fcc phase" << endl;
        result = user->pe_restorePhase("FCC_A1",  msg);
        if (!result) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.2 Calculate the Stable Phase Equilibrium

```
//====================================
//    PanEngine Test Example 2
//
// stable phase equilibrium for a point
//====================================

int PanEngine_Test_2()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //==========================================
        // Load thermodynamic database in tdb Format
        //==========================================
        char* pdbFileName = "AlMgZn.pdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readPDBFile(pdbFileName, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //==========================================
        // Define a subsystem for current calculation
        //==========================================
        cout << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }

        //==========================================
        // Define a PanPoint pointer and initialize it
        //==========================================
        cout << "Define and initialize a PanPoint:" << endl;

        PanPoint *pt;
        pt = new PanPoint;
        user->pe_initializePanPoint(pt);

        //=======================
        // Set temperature (in K)
        //=======================
        cout << "Set temperature" << endl;
        pt->setT(700);
```

```
        //=====================================================
        //  Set overall composition
        //
        // setUnitX(true): true for mole fraction
        // setUnitX(false): for weight fraction
        // if weight fraction, must call the function:
        // user->pe_stateSpacePointConversion_X_Wt(&pt->st, msg)
        // to convert weight fraction to mole fraction
        //=====================================================
        cout << "Set composition" << endl;
        pt->setUnitX(true);    // use mole fraction

        pt->setX("Al", 0.1);
        pt->setX("Mg", 0.8);
        pt->setX("Zn", 0.1);

        cout << "Output the point conditions" << endl;
        pt->print(cout);

        cout << "\nFind Stable Point:\n\n";
        result = user->pe_findStablePoint(pt, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        } else {
                pt->print(cout);
        }

        //=====================================================================
        // Calculate a series of stable points while temperature is changing
        //=====================================================================
        cout << "Calculating a series of stable points, T changes:\n";
        for (int i = 0; i < 10; i++) {
                pt->st.TK = 850 - 20 * i;
                result = user->pe_findStablePoint(pt, msg);
                cout << "----------------------\n";
                pt->print(cout);
        }

        //================================================
        // Calculate a series of stable equilibrium points
        // while temperature and composition are changing
        //================================================
        cout << "Calculating a series of stable points, T and x change:\n";
        double xAl, xMg, xZn;
        for (i = 0; i < 10; i++) {
                pt->st.TK = 850 - 20 * i;
                xMg = 0.1 + 0.01 * i;
                xZn = 0.2 + 0.01 * i;
                xAl = 1 - xMg - xZn;
                pt->setX("Al", xAl);
                pt->setX("Mg", xMg);
                pt->setX("Zn", xZn);

                result = user->pe_findStablePoint(pt, msg);
                cout << "----------------------\n";
                pt->print(cout);
        }

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.3 Find a Local Phase Equilibrium

```
//============================
//    PanEngine Test Example 3
//
// stable phase equilibrium
// local phase equilibrium
//============================

int PanEngine_Test_3()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //=========================================
        // Load thermodynamic database in tdb Format
        //=========================================
        char* tdbFileName = "AlMgZn.tdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readTDBFile(tdbFileName, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //=========================================
        // Define a subsystem for current calculation
        //=========================================
        cout << endl << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }

        //=========================================
        // Define a PanPoint pointer and initialize it
        //=========================================
        cout << "Define and initialize a PanPoint:" << endl;

        PanPoint *pt;
        pt = new PanPoint;
        user->pe_initializePanPoint(pt);

        //=======================
        // Set temperature (in K)
        //=======================
        cout << "Set temperature" << endl;
        pt->setT(700);

        cout << "Set composition" << endl;
        pt->setUnitX(true);
```

```
        pt->setX("Al", 0.1);
        pt->setX("Mg", 0.8);
        pt->setX("Zn", 0.1);

        cout << "Output the point conditions" << endl;
        pt->print(cout);

        //=========================
        // Calculate a stable point
        //=========================
        cout << "\nFind Stable Point:\n\n";
        result = user->pe_findStablePoint(pt, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        } else {
                pt->print(cout);
        }

        //============================================
        // Calculate a series of local phase equilibria
        // Use the previous phase equilibrium results
        // as the initial values for local equilibrium
        //============================================
        cout << "Calculating a series of local points, T changes:\n";
        for (int i = 0; i < 10; i++) {
                pt->st.TK = 850 - 20 * i;
                result = user->pe_findLocalPoint(pt, msg);
                cout << "----------------------\n";
                pt->print(cout);
        }

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.4 Find the Liquidus Surface, etc.

```
//==================================
//    PanEngine Test Example 4
//
// Calculate liquidus surface
// Calculate liquidus slope
// Calculate distribution coefficient
//==================================

int PanEngine_Test_4()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);
```

```
cout <<msg << endl;
if (user == NULL) {
        cout << "Error in defineUser" << endl;
        return 0;
}

//=========================================
// Load thermodynamic database in tdb Format
//=========================================
char* tdbFileName = "AlMgZn.tdb";
bool result;

cout << "Reading database file:" << endl;
result = user->pe_readTDBFile(tdbFileName, msg);
if (result == false) {
        cout << "Error: " << msg << endl;
        return 0;
}

//=========================================
// Define a subsystem for current calculation
//=========================================
cout << endl << "Define which elements should be used in the calculation:" << endl;
result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
if (result == false) {
        cout << "Error" << endl;
        return 0;
}

//=========================================
// Define a PanPoint pointer and initialize it
//=========================================
cout << "Define and initialize a PanPoint:" << endl;

PanPoint *pt;
pt = new PanPoint;
user->pe_initializePanPoint(pt);

//==================================
// Set T and composition for a point
//==================================
pt->setT(1000);                       // T is an initial guess for the liquidus surface
pt->setUnitX(false);  // set unit for composition

pt->setWt("Al", 0.1);
pt->setWt("Mg", 0.8);
pt->setWt("Zn", 0.1);
user->pe_stateSpacePointConversion_X_Wt(&pt->st, msg);

cout << "Output the point conditions:" << endl;
pt->print(cout);

//=============================================================
// Find the liquidus surface for given T and overall composition
//=============================================================
cout << "Find the liquidus surface for the given composition:" << endl;
result = user->pe_findLiquidusSurface(pt, msg);
if (result == false) {
        cout << "Error: " << msg << endl;
        pause();
        return 0;
} else {
        pt->print(cout);
}

//==================================
// Calculate the liquidus slope dT/dxi
//==================================
cout << "GetLiquidusSlope:" << endl;
double slope;
result = user->pe_getLiquidusSlope(pt, "Al", slope, msg);
```

```
        if (!result) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        } else {
                cout << "slope = " <<  slope << endl;
                pt->print(cout);
        }

        //=====================================
        // Calculate the partition coefficient k
        //=====================================
        double k;
        cout << "Calculate partition coefficient k:" << endl;
        result = user->pe_getK(pt, "Liquid", "HCP_A3", "Al", k, msg);
        if (!result) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        } else {
                cout << "k = " << k << endl;
        }

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.5 Solidification Simulation

```
//=========================
//  PanEngine Test Example 5
//
// Solidification Simulation
//=========================

int PanEngine_Test_5()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //=========================================
        // Load thermodynamic database in tdb Format
        //=========================================
        char* tdbFileName = "AlMgZn.tdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readTDBFile(tdbFileName, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
```

```
        }

        //=========================================
        // Define a subsystem for current calculation
        //=========================================
        cout << endl << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }

        //=========================================
        // Define a PanPoint pointer and initialize it
        //=========================================
        cout << "Define and initialize a PanPoint:" << endl;

        PanPoint *pt;
        pt = new PanPoint;
        user->pe_initializePanPoint(pt);

        //=================================
        // Set T and composition for a point
        //=================================
        pt->setT(1000);                       // T is an initial guess for the liquidus surface
        pt->setUnitX(false);   // using weight fraction

        pt->setWt("Al", 0.1);
        pt->setWt("Mg", 0.8);
        pt->setWt("Zn", 0.1);
        user->pe_stateSpacePointConversion_X_Wt(&pt->st, msg);

        cout << "Output the point conditions" << endl;
        pt->print(cout);

        //=================================================
        // Solidification simulation, with lever rule model
        //=================================================
        SOLIDIFICATION_MODEL sModel;
        sModel = LEVER;

        cout << "\nSolidification simulation, with lever rule model:" << endl;
        result = user->pe_solidificationSimulation(sModel, pt, calculationProgress, msg);

        if (result == false) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        }

        //=============================================
        // Solidification simulation, with SCHEIL model
        //=============================================
        sModel = SCHEIL;

        cout << "\nSolidification simulation, with SCHEIL model:" << endl;
        result = user->pe_solidificationSimulation(sModel, pt, calculationProgress, msg);

        if (result == false) {
                cout << "Error: " << msg << endl;
                pause();
                return 0;
        }

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.6 Calculate Thermodynamic Properties

```
//==================================
//    PanEngine Test Example 6
//
// Calculate thermodynamic properties
//==================================

int PanEngine_Test_6()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //=========================================
        // Load thermodynamic database in tdb Format
        //=========================================
        char* tdbFileName = "AlMgZn.tdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readTDBFile(tdbFileName, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //=========================================
        // Define a subsystem for current calculation
        //=========================================
        cout << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }

        //=========================================
        // Define a PanPoint pointer and initialize it
        //=========================================
        cout << "Define and initialize a PanPoint:" << endl;

        PanPoint *pt;
        pt = new PanPoint;
        user->pe_initializePanPoint(pt);

        //================================
        // Set T and composition for a point
        //================================
        pt->setT(1000);
        pt->setUnitX(false);
```

```
pt->setWt("Al", 0.1);
pt->setWt("Mg", 0.8);
pt->setWt("Zn", 0.1);
user->pe_stateSpacePointConversion_X_Wt(&pt->st, msg);

cout << "Output the point conditions" << endl;
pt->print(cout);

//==================================
// Calculate thermodynamic properties
//==================================
user->pe_setOption(1, 1);

cout << "\nThermodynamic Property Calculation:" << endl;
result = user->pe_getThermodynamicProperty(pt, "Liquid", msg);

if (result == false) {
        cout << "Error: " << msg << endl;
        pause();
        return 0;
}
pt->print(cout);

//=============================================================================
// Calculate a series of points for thermodynamic properties for phase FCC_A1
//=============================================================================
int i;
ofstream out;
out.open("GHS_FCC_A1.dat", ios::out);
out << "xAl\txMg\txZn\tG\tH\tS\n";
for (i = 0; i < 100; i++) {
        double wtAl, wtMg, wtZn;
        wtMg = 0.01;
        wtZn = 0.01 * i;
        wtAl = 1 - wtMg - wtZn;

        pt->setT(300);
        pt->setUnitX(false);

        pt->setWt("Al", wtAl);
        pt->setWt("Mg", wtMg);
        pt->setWt("Zn", wtZn);

        result = user->pe_getThermodynamicProperty(pt, "fcc_A1", msg);
        out << pt->phasePt[0].st.comp[0].x << "\t" << pt->phasePt[0].st.comp[1].x
                << "\t" << pt->phasePt[0].st.comp[2].x << "\t";
        out << pt->phasePt[0].G << "\t" << pt->phasePt[0].H
                << "\t" << pt->phasePt[0].S << endl;
        pt->print(cout);
}
out.close();

//=========================================================================
// Calculate a series of points for thermodynamic properties for phase PHI
// PHI phase has x(Al):x(Mg):x(Zn) = 2 : 5 : 2
//=========================================================================
out.open("GHS_PHI.dat", ios::out);
out << "xAl\txMg\txZn\tG\tH\tS\n";
for (i = 0; i < 100; i++) {
        double wtAl, wtMg, wtZn;
        wtMg = 0.01;
        wtZn = 0.01 * i;
        wtAl = 1 - wtMg - wtZn;

        pt->setT(300);
        pt->setUnitX(false);

        pt->setWt("Al", wtAl);
        pt->setWt("Mg", wtMg);
        pt->setWt("Zn", wtZn);
```

```
                result = user->pe_getThermodynamicProperty(pt, "PHI", msg);
                out << pt->phasePt[0].st.comp[0].x << "\t" << pt->phasePt[0].st.comp[1].x
                        << "\t" << pt->phasePt[0].st.comp[2].x << "\t";
                out << pt->phasePt[0].G << "\t" << pt->phasePt[0].H << "\t"
                        << pt->phasePt[0].S << endl;
                pt->print(cout);
        }
        out.close();

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```

5.7 Calculate a Thermodynamic Factor

```
//================================
//    PanEngine Test Example 7
//
// Calculate thermodynamic factor
//================================

int PanEngine_Test_7()
{
        //================================
        // Define a PanEngine User pointer
        //================================
        User* user;
        char msg[80];

        cout << "Defining User pointer:" << endl;
        user = pe_defineUser(msg);

        cout <<msg << endl;
        if (user == NULL) {
                cout << "Error in defineUser" << endl;
                return 0;
        }

        //========================================
        // Load thermodynamic database in tdb Format
        //========================================
        char* tdbFileName = "AlMgZn.tdb";
        bool result;

        cout << "Reading database file:" << endl;
        result = user->pe_readTDBFile(tdbFileName, msg);
        if (result == false) {
                cout << "Error: " << msg << endl;
                return 0;
        }

        //========================================
        // Define a subsystem for current calculation
        //========================================
        cout << "Define which elements should be used in the calculation:" << endl;
        result = user->pe_defineSubSystem(3, "Al", "Mg", "Zn");
        if (result == false) {
                cout << "Error" << endl;
                return 0;
        }
```

```
        //=========================================
        // Define a PanPoint pointer and initialize it
        //=========================================
        cout << "Define and initialize a PanPoint:" << endl;

        PanPoint *pt;
        pt = new PanPoint;
        user->pe_initializePanPoint(pt);

        //================================
        // Suspend all phases in subsystem
        //================================
        cout << "\nSuspend all phases" << endl;
        result = user->pe_suspendAllPhases(msg);
        if (!result) {
               cout << "Error:  " << msg << endl;
               return 0;
        } else {
               cout << msg << endl;
        }

        //==================
        // Restore one phase
        //==================
        cout << "\nRestore the HCP_A3 phase" << endl;
        result = user->pe_restorePhase("HCP_A3",  msg);
        if (!result) {
               cout << "Error: " << msg << endl;
               return 0;
        }

        //=================================
        // Set T and composition for a point
        //=================================
        pt->setT(400);
        pt->setUnitX(true);
        pt->setX("Al", 0.1);
        pt->setX("Mg", 0.8);
        pt->setX("Zn", 0.1);

        //================================
        // Calculate thermodynamic factor
        //================================
        cout << "\nCalculate thermodynamic factors Tij:" << endl;
        double Tij;
        result = user->pe_getThermodynamicFactor(pt, "Al", "Zn", "Mg", Tij, msg);
        if (!result) {
               cout << "Error: " << msg << endl;
               pause();
               return 0;
        }
        cout << "Tij(Mg, Zn) = " << Tij << endl;
        pt->print(cout);

        cout << "\nCalculate thermodynamic factors Tij:" << endl;
        result = user->pe_getThermodynamicFactor(pt, "Al", "Al", "Zn",  Tij, msg);
        if (!result) {
               cout << "Error: " << msg << endl;
               pause();
               return 0;
        }
        cout << "Tij(Al, Zn) = " << Tij << endl;
        pt->print(cout);

        //==============================
        // Delete PanEngine User pointer
        //==============================
        pe_deleteUser(user);

        return 1;
}
```