ORNL/TM-2012/576

# Thermochimica User Manual v1.0

## December 2012

**Prepared by**

**M.H.A. Piro**
**S. Simunovic**
**T.M. Besmann**

ORNL/TM-2012/576

Materials Science and Technology Division
Computer Science and Mathematics Division

# THERMOCHIMICA USER MANUAL – v1.0

Markus H.A. Piro
Srdjan Simunovic
Theodore M. Besmann

Date Published: December 2012

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

## CONTENTS

**Page**

## LIST OF FIGURES

**Figure** **Page**

# LIST OF TABLES

**Table** **Page**

# 1. INTRODUCTION

## 1.1 OVERVIEW

A new software library called Thermochimica has been developed that determines a unique combination of phases and their compositions at thermochemical equilibrium. The development of Thermochimica in this research is aimed at direct integration in multi-physics codes and to address several concerns with using commercially available software, including: licensing entanglements associated with code distribution, access to the source code, convenient incorporation into other codes with quality assurance considerations, and computational performance. In particular, significant research efforts have been dedicated to enhance computational performance through advanced algorithm development, such as improved estimation techniques and non-linear solvers. The overall goal of this undertaking is to provide an open source computational tool to enhance the predictive capability of multi-physics codes without significantly impeding computational performance.

The purpose of this document is to describe the design of Thermochimica and the basic use of the software library. A brief background to computational thermodynamics and the conditions for thermodynamic equilibrium is given in §2, details of the code development is discussed in §3, details of building and compiling Thermochimica is given in §4, specific programming details for each subroutine in the library is given in the Appendix. The open license for Thermochimica is provided with the source code.

As is always the case, software quality assurance (SQA) is an extremely important subject in software development. The importance of this warrants an independent document dedicated to SQA, which has been developed in parallel to this document. The subject of thermodynamic model validation is also of extreme importance in scientific computing and must be given careful attention. To be clear, this document is concerned with the development and use of a *general thermodynamic solver* and the validation of any models should be included in the discussion of the development of a *thermodynamic model* that is specific to a particular system.

## 1.2 PURPOSE

Thermochimica can be used for stand-alone calculations or it can be directly coupled to other codes. The initial release of the software does not have a graphical user interface (GUI) and it can be executed from the command line or from an Application Programming Interface (API). Also, it is not intended for thermodynamic model development or for constructing phase diagrams. The main purpose of the software is to be directly coupled with a multi-physics code to provide material properties and boundary conditions for various physical phenomena.

Temperature, hydrostatic pressure and the mass of each chemical element are provided as input in addition to a thermodynamic database (in the form of a ChemSage data-file). ChemSage data-files can be generated by the commercial software package FactSage [1] or through the auxiliary code CSFAP.

Various useful parameters can be provided as output from Thermochimica, such as:

- determination of which phases are stable at equilibrium,
- the mass of solution species and phases at equilibrium,
- mole fractions of solution phase constituents,

- thermochemical activities (which are related to partial pressures for gaseous species),
- chemical potentials of solution species and phases, and
- integral Gibbs energy (referenced relative to standard state)

All input/output operations are performed via Fortran modules, specifically ModuleThermo.f90 and ModuleThermoIO.f90. An example is given in §3.6 of how one can make use of the aforementioned variables for the uranium-oxygen system. Details for all variables used by all modules are documented in a dOxygen report, which is provided in the Appendix. dOxygen [2] is an automatic documenting tool that generates an HTML web site and Latex report from comments embedded in the source code. Thus, all documentation is consolidated directly in the source code in a convenient manner.

## 1.3 CAPIBILITIES, LIMITATIONS AND ASSUMPTIONS

All calculations performed by Thermochimica apply at thermodynamic equilibrium and do not take chemical kinetic information into consideration. Although chemical equilibrium is not achieved instantaneously, chemical kinetics may not have a large part to play in some situations, depending on temperature, pressure and the particular system under consideration. Generally, thermodynamic calculations are more appropriate at high temperatures, over a sufficiently long period of time and when atoms of the various chemical elements are randomly mixed within the thermodynamic system. For instance, chemical equilibrium is achieved in relatively short time periods in nuclear fuel under normal operating conditions due to the high temperatures experienced in a reactor and the relatively long time periods between refueling[1]. Also, due to the nature of fission, the atoms of the various elements representing the fission and activation products are randomly mixed in irradiated nuclear fuel[2]. Many engineering fields have relied on the same assumption of local thermodynamic equilibrium in multi-physics codes, such as combustion, geology and metallurgy. The onus is on the user for judging whether equilibrium calculations are appropriate for a particular simulation.

The current capabilities of Thermochimica 1.0 and the data-file parser are summarized below[3]:

- Systems comprised of various combinations of pure stoichiometric phases, and ideal and non-ideal solution phases,
- Non-ideal regular solution phases based on the “QKTO” identification tag,
- Between 2 and 4 constituents per mixing term in a QKTO phase,
- Systems with a minimum number of 2 elements and a maximum of 48 elements,
- Systems with up to 42 solution phases,
- Systems with up to 1500 species,
- Systems with fewer chemical elements than what is represented by the data-file,
- Species with magnetic contributions to the standard molar Gibbs energy terms, and

[1] The simulation of transient behavior in a nuclear reactor may prove less appropriate for thermochemical equilibrium calculations. However, this approximation may still be significantly better than neglecting thermochemistry entirely, as is the case in many traditional nuclear fuel performance codes.

[2] For the case of simulating nuclear fuel behavior in a multi-physics code, a thermodynamic system is represented by a finite element, which is sufficiently discretized to capture spatial variations in the local fission product inventory (resulting from neutron flux depression and mass diffusion across the fuel). Thus, the statement of atoms of the different elements being randomly mixed in nuclear fuel is valid in this context.

[3] Thermochimica 1.0 is limited by the ChemSage data-files that are used as input, which is a function of the FactSage software. An earlier version of the software has been tested with a fictive system representing 118 elements, 1500 species and 20 solution phases [3].

- Species with multiple particles per mole.

Several limitations in the current version include:

- Only ChemSage data-files can be parsed, although the CSFAP code can convert ThermoCalc (*.tdb) data files to the ChemSage (*.dat) format.
- The only non-ideal solution phase model that can be used has the QKTO tag,
- Real gases cannot be considered,
- Aqueous solution phases cannot be used in the current version[4],
- The Muggianu interpolation scheme has not been implemented,

Acceptable input units include [character strings expected by software are in square brackets]:

- Temperature units in Kelvin [K], Celsius [C], Fahrenheit [F] or Rankine [R],
- Hydrostatic pressure units can be in atmospheres [atm], pounds per square inch [psi], bar [bar], Pascals [Pa] and kilopascals [kPa], and
- Mass units can be in moles [mol], mole fraction [mole fraction], atoms [atoms], gram-atoms [gram-atoms], atom fraction [atom fraction], kilograms [kilograms], grams [grams], pounds [pounds], and mass fraction [mass fraction].

The units for temperature, hydrostatic pressure and mass are converted to Kelvin, atmospheres and atom fractions, respectively. Acceptable input quantities are as follows:

- Temperature, $295\ \text{K} \leq T \leq 6{,}000\ \text{K}$,
- Hydrostatic pressure, $10^{-6}\ \text{atm} \leq P \leq 10^{6}\ \text{atm}$, and
- Atom fraction, $10^{-10} \leq \chi_j \leq 1$.

The motivation for applying a minimum atom fraction of $10^{-10}$ is for computational reasons and is discussed in §3.2.

As is the case in any scientific or engineering software, the user is ultimately responsible for judging whether the results produced by the software are reasonable for a particular application. Significant efforts have been dedicated to ensure that the results produced by Thermochimica are numerically correct; however, the results produced by the software depend on thermodynamic data and models provided. The user should be aware of limitations in using certain thermodynamic datasets and models in addition to the limitation of performing *equilibrium* calculations.

[4] An earlier version of the software was capable of handling aqueous solution phases [3].

# 2. BACKGROUND

A brief background in classical thermodynamics and the conditions for thermodynamic equilibrium is provided for sake of completion. A thorough discussion of the conditions for thermodynamic equilibrium is provided by Piro [3] and details of the numerical methods employed by Thermochimica are given by Piro *et al* [4].

## 2.1 GLOSSARY

Table 2.1 summarizes many thermodynamic terms that are used in this document and in comments located in the source code.

**Table 2.1** – A glossary of thermodynamic terms relevant to this document.

| Term | Symbol | Definition |
|---|---|---|
| Activity | $a_i$ | A dimensionless quantity related to the chemical potential of a substance. The activity is equivalent to mole fraction, $x_i$, for an ideal solution phase and unity for a condensed phase. |
| Activity Coefficient | $\gamma_i$ | The activity coefficient accounts for the departure of a substance from ideal behavior and is defined by $\gamma_i = a_i / x_i$. This is related to the partial molar excess Gibbs energy of mixing. |
| Atomic Number | $Z$ | The number of protons in the nucleus of an atom, and also its positive charge. Each chemical element has its characteristic atomic number. |
| Chemical Potential | $\mu_i$ | A measure of the effect on the change of the Gibbs energy of the system by the introduction of a substance. The chemical potential is formally defined as $\mu_i = \left(\partial G / \partial n_i\right)_{T,P,n_{k\neq i}}$. |
| Closed system | | A system that permits the exchange of heat and work with its surroundings at constant mass. |
| Component | | A phase component refers to a constituent of a solution phase that can be varied independently. A system component refers to the "minimum number of independent species necessary to define the composition in all phases of a system" [5]. |
| Constituent | | A constituent of a solution phase refers to a particular species in a particular phase. For example, "$H_2O$" is a species, whereas "$H_2O_{(g)}$" refers to $H_2O$ in the gaseous phase and "$H_2O_{(l)}$" refers to $H_2O$ in a liquid phase. |
| Element | | A chemical element as it appears on the Periodic Table of the Elements, which is not to be confused with a nuclear fuel element or an element of a vector/matrix. |
| Gibbs energy | $G$ | A thermodynamic potential measuring the maximum amount of useful work obtainable from an isothermal-isobaric closed system. The Gibbs energy is defined as the difference between enthalpy and the product of temperature and entropy [5]. |
| Ideal solution | | A solution phase that is comprised of constituents that mix ideally, i.e., the activity coefficient is equal to unity. |
| Ion | | An atom or molecule where the number of electrons does not equal the number of protons, and thus has a positive or negative charge. |
| Isobaric | | A system at constant hydrostatic pressure. |
| Isothermal | | A system at constant temperature. |
| Isotope | | One of several nuclides with the same atomic number (i.e., the same chemical element), but with different atomic weights and thus different nuclear properties. |
| Molality | | Molality denotes the number of moles of a solute *i* per kilogram of solvent (not solution). |

| | | |
|---|---|---|
| Mole | $n_i$ | A quantity of mass measured as 6.02214179E23 atoms. Equivalent to one gram-atom for a pure element. |
| Mole fraction | $x_i$ | The fraction of moles of a particular species in a particular solution phase. |
| Non-ideal solution | | A solution phase that is comprised of constituents that do not mix ideally and the activity coefficient is not equal to unity. The departure from ideal mixing behavior is represented by the partial molar excess Gibbs energy of mixing, which is typically provided by a thermodynamic model. |
| Partial molar excess Gibbs energy of mixing | $g^{ex}_{i(\lambda)}$ | The partial molar excess Gibbs energy of mixing represents the contribution to the chemical potential term due to non-ideal behavior. This is the difference between the real chemical potential of a substance and that from assuming ideal mixing behavior. |
| Phase | | A body of matter that is uniform in chemical composition and physical state. Phases are separated from one another by a physical discontinuity. A phase is not to be confused with a state of matter. For example, there are three different phases of pure uranium in the solid state (orthogonal, tetragonal and body centerd cubic). Thermochimica considers two different types of phases: pure condensed phases and solution phases. "Mixtures" are numerically treated as solution phases. |
| Phase Assemblage | | A unique combination of phases predicted to be stable at thermodynamic equilibrium for a particular system. |
| Pure condensed phase | $\omega,\Omega$ | A condensed phase with invariant stoichiometry and may be interpreted mathematically as containing a single species with unit concentration. A pure condensed phase may be in either liquid or solid states. The symbol $\omega$ represents the phase index and $\Omega$ represents the number of stable pure condensed phases. |
| Solution phase | $\lambda,\Lambda$ | A phase containing a mixture of multiple species. Traditionally, the term "solution" refers to a condensed phase. Since the mathematical representation of a gaseous mixture is identical to that of a solution phase, the programming is simplified in the current work by calling a gaseous "mixture" as a "solution." The symbol $\lambda$ represents the phase index and $\Lambda$ represents the number of stable solution phases. |
| Species | $i$ | A chemically distinct molecular entity. For example, $H_2O$ has a distinct chemical composition, but can be in gaseous, liquid or solid phases. This differs from the term constituent, which refers to a particular species in a particular phase. The phase index is represented by the variable *i*. |
| Standard molar Gibbs energy | $g^{o}_{i(\lambda)}$ | The standard molar Gibbs energy of a pure constituent is the Gibbs energy of that constituent with unit activity. This quantity is computed using values from the ChemSage data-file. |
| State | | A state of matter distinguishes the distinct form that matter may take on. This includes solid, liquid, gas, plasma, and the Bose-Einstein condensate. Thermochimica only considers the solid, liquid and gaseous states. The term "state" is not to be confused with the term "phase". |
| Stoichiometry | | This refers to the relative amounts of atoms per formula mass of a substance. |
| Surroundings | | The space remaining in the Universe outside of a system. |
| System | | A portion of the Universe with a perimeter defined by real or imaginary boundaries. |

## 2.2 CONDITIONS FOR THERMODYNAMIC EQUILIBRIUM

The foundation of computing thermodynamic equilibria is based on minimizing the integral Gibbs energy of a closed system at constant temperature and hydrostatic pressure. This equilibrium condition is a manifestation of the first and second laws of thermodynamics, which state that the energy of the Universe is constant and that the entropy of the Universe tends to a maximum [6]. Alternatively, one could minimize the Helmholtz energy at constant temperature and volume. Traditionally, the Gibbs energy function is used instead of the Helmholtz energy because hydrostatic pressure is more easily controlled in experiments than volume, particularly for systems involving condensed phases [3].

From a numerical point of view, the objective of computing thermodynamic equilibria is to determine a unique combination of phases and their composition that yields a global minimum in the integral Gibbs energy of an isothermal, isobaric system, which is generally given by

$$G/RT = \sum_{\lambda=1}^{\Lambda} n_\lambda \sum_{i=1}^{N_\lambda} x_{i(\lambda)} \tilde{\mu}_i + \sum_{\omega=1}^{\Omega} n_\omega \tilde{\mu}_\omega \tag{1}$$

where $R$ [J·mol$^{-1}$·K$^{-1}$] is the ideal gas constant, $T$ [K] is the absolute temperature, $N_\lambda$ denotes the number of constituents in solution phase $\lambda$, and $\Lambda$ and $\Omega$ represent the total number of solution phases and pure condensed phases in the system, respectively. The mole fraction of constituent $i$ in solution phase $\lambda$ is $x_{i(\lambda)}$ [unitless] and the number of moles of solution phase $\lambda$ and pure condensed phase $\omega$ are denoted by $n_\lambda$ and $n_\omega$ [mol] respectively. Finally, $\tilde{\mu}_i$ and $\tilde{\mu}_\omega$ [unitless] represent the dimensionless chemical potential of constituent $i$ in solution phase $\lambda$ and pure condensed phase $\omega$, respectively. The diacritic symbol ~ is used to emphasize that chemical potential terms used in this discussion are dimensionless.

The chemical potential of a constituent in an ideal solution phase incorporates the standard molar Gibbs energy of the pure species, $\tilde{g}^o_{i(\lambda)}$ [unitless], and the entropic contribution due to mixing as a function of its mole fraction, such that [8]

$$\tilde{\mu}_i = \tilde{g}^o_{i(\lambda)} + \ln x_{i(\lambda)} \tag{2}$$

The chemical potential of a non-ideal solution phase constituent adds a partial molar excess Gibbs energy of mixing term, $\tilde{g}^{Ex}_{i(\lambda)}$ [unitless], to more closely represent experimentally observed behavior

$$\tilde{\mu}_i = \tilde{g}^o_{i(\lambda)} + \tilde{g}^{Ex}_{i(\lambda)} + \ln x_{i(\lambda)} \tag{3}$$

Finally, the chemical potential of a pure condensed phase does not include a composition dependent term, since the stoichiometry is by definition invariant. The equilibrium condition is subject to several constraints, including conservation of mass and the Gibbs phase rule. The total mass (in gram-atoms) of element $j$ is

$$b_j = \sum_{\lambda=1}^{\Lambda} n_\lambda \sum_{i=1}^{N_\lambda} x_{i(\lambda)} a_{i,j} + \sum_{\omega=1}^{\Omega} n_\omega a_{\omega,j} \tag{4}$$

where $a_{i,j}$ and $a_{\omega,j}$ [g-at/mol] are the stoichiometric coefficients of element $j$ in constituent $i$ and pure condensed phase $\omega$ respectively. The Gibbs phase rule is generally given as

$$F = C - \Phi + 2 \tag{5}$$

where F represents the number of degrees of freedom, C is the number of system components (taken here as chemical elements) and $\Phi$ (i.e., $\Phi = \Lambda + \Omega$) represents the number of phases currently predicted to

be stable in the system. Maintaining constant temperature and pressure removes two degrees of freedom. Since the number of degrees of freedom must be non-negative, the phase rule dictates that the maximum number of coexisting phases at equilibrium cannot exceed the number of chemical elements in a closed isothermal-isobaric system.

Finally, Gibbs' criteria for equilibrium require that the Gibbs energy of a closed system be at a global minimum at constant temperature and pressure. An equivalent statement defines the chemical potential of any constituent in a stable phase by a linear function of the element potentials as [7,8]

$$\tilde{\mu}_i = \sum_{j=1}^{E} a_{i,j} \tilde{\Gamma}_j \tag{6}$$

In summary, the necessary conditions for thermodynamic equilibrium require that the chemical potentials of all stable solution phase constituents and pure condensed phases abide by the above equality, and that the conservation of mass and the Gibbs phase rule are satisfied. The sufficient condition for equilibrium – which ensures that the system is at a global minimum yielding a unique solution – is that the molar Gibbs energy of all metastable phases do not violate the following inequality

$$\sum_{i=1}^{N_\lambda} x_{i(\lambda)} \tilde{\mu}_{i(\lambda)} > \sum_{i=1}^{N_\lambda} x_{i(\lambda)} \sum_{j=1}^{E} a_{i,j} \tilde{\Gamma}_j \tag{7}$$

The interested reader is referred to the literature on specific details on determining initial estimates for Thermochimica [9], a general overview of numerical techniques used by the solver [3,4], and numerical methods in ensuring that the necessary [10] and sufficient conditions [11] have been satisfied.

# 3. CODE DEVELOPMENT

## 3.1 OVERVIEW

Thermochimica is written in Fortran and complies with the Fortran 90/2003 standard. A description of each subroutine is provided in the dOxygen documentation located in the Appendix. The documentation for each subroutine includes a description of the purpose of each subroutine and a description of all input/output parameters. A description of each local variable is provided for each subroutine and a description of each global variable is provided in each module. All of the programming in Thermochimica is original unless otherwise specified in the source code. These subroutines have been modified from Numerical Recipes [12] and have been referenced appropriately in the source code.

## 3.2 CHECKING INPUT VALUES

Thermochimica will test to ensure that the input variables are appropriate for consideration. For instance, the units for temperature, pressure and mass are compared with the list of acceptable units highlighted in §1.3. Temperature, pressure and mass are converted to units of Kelvin, atmospheres and atom fractions, respectively. The values of temperature, pressure and mass are tested to be real and within an acceptable range. The acceptable range for temperature is 295 K $\leq T \leq$ 6000 K and the acceptable range for hydrostatic pressure is $10^{-6}$ atm $\leq P \leq 10^{6}$ atm. An error is recorded (via the variable `INFOThermo`) and execution halts if either temperature or pressure is out of range or not real.

An element that has an atom fraction that is out of range will not be considered part of the system. An error will be recorded and execution will halt if the number of elements in the system is less than two or if any of the atom fractions are non-real or negative. The minimum allowable atom fraction of any element is defined as the division of machine precision ($10^{-15}$ for double precision variables) by the relative error tolerance for the mass balance equations. By default, the tolerance for the relative errors for the mass balance equations is $10^{-5}$, which yields a minimum atom fraction tolerance of $10^{-10}$. The motivation for applying this tolerance is in accordance with numerical errors associated with solving a system of linear equations representing the mass balance constraints. This tolerance is generally exceedingly small in comparison to experimental errors and uncertainties, and it is considered more than sufficient for investigating a wide array of engineering problems.

## 3.3 EXAMPLE FORTRAN APPLICATION PROGRAMMING INTERFACE

Figure 3.1 provides an example Fortran wrapper to call the ChemSage data-file parser (ParseCSDataFile.f90), Thermochimica (Thermochimica.f90), the destructor (ResetThermoAll.f90, optional) and the debugger (ThermoDebug.f90, optional). This example wrapper is provided as a Fortran program with the software and is labeled thermo.f90. The destructor deallocates all allocatable arrays used by both the parser and the solver. The debugger prints an error message to the screen in the event that an error has been encountered. The integer scalar variable `INFOThermo` is used to identify a successful exit or to record a specific error encountered in execution. One should verify a null value for `INFOThermo` after calling the parser and before calling Thermochimica. An error may be returned in the event that there is an error opening, reading or closing the data-file. This particular example considers the uranium-oxygen system at 500 K and 1 atm with 1 mol U and 2.01 mol O.

```
    !-----------------------------------------------------------------
    !
    ! Purpose:
    ! ========
    !
    ! The purpose of this program is to be a wrapper around
    ! Thermochimica to simulate the interaction with another code.
    ! The module "ModuleThermoIO" is intended to be used by the other
    ! code that calls Thermochimica for input/output operations.
    !
    !-----------------------------------------------------------------

program thermo

    USE ModuleThermoIO

    implicit none


    ! Initialize variables:
    dElementMass         = 0D0
    INFOThermo           = 0


    ! Declare variable units:
    cThermoInputUnits(1) = 'K'
    cThermoInputUnits(2) = 'atm'
    cThermoInputUnits(3) = 'moles'

    ! Declare thermodynamic data-file path and name:
    cThermoFileName      = '../../../data/Example4.dat'

    ! Define the temperature and pressure:
    dPressure            = 1D0
    dTemperature         = 500D0

    ! Define the mass of each chemical element:
    dElementMass(92)     = 1D0          ! Element ID 92 = U
    dElementMass(8)      = 2.01D0       ! Element ID 8  = O

    ! Parse the ChemSage data-file:
    call ParseCSDataFile(cThermoFileName)

    ! Call Thermochimica:
    if (INFOThermo == 0) call Thermochimica

    ! Destruct everything (optional):
    if (INFOThermo == 0) call ResetThermoAll

    ! Call the debugger (optional):
    call ThermoDebug

end program thermo
```

**Figure 3.1** – A sample Fortran wrapper is provided that calls the ChemSage data-file parser, Thermochimica, the destructor (optional) and the debugger (optional).

As described in the Appendix, the variable `dElementMass` is a double real vector dimensioned from (0:118), where each coefficient corresponds to an element on the periodic table. The zeroth coefficient corresponds to the electron, which is used in ionic phases, such as aqueous, plasma or the solid solution phases based on the compound energy formalism (yet to be implemented). The system shown in Figure 3.1 represents uranium and oxygen, which have atomic numbers 92 and 8 respectively. Therefore, the example system above has 1 mole of uranium and 2.01 moles of oxygen.

### 3.4 EXAMPLE C++ APPLICATION PROGRAMMING INTERFACE

A sample C++ API for Thermochimica is given in Figure 3.2 below.

```
#ifndef thermochimica_THERMOCHIMICA_API_hh
#define thermochimica_THERMOCHIMICA_API_hh

#ifndef CMAKE_CONFIGURED

/* FORTRAN WRAPPER FUNCTIONS */
#define THERMOCHIMICA_SET_PRESSURE FC_FUNC_(thermochimica_set_pressure,
THERMOCHIMICA_SET_PRESSURE)
#define THERMOCHIMICA_SET_TEMPERATURE
FC_FUNC_(thermochimica_set_temperature, THERMOCHIMICA_SET_TEMPERATURE)
#define THERMOCHIMICA_SET_ELEMENTMASS
FC_FUNC_(thermochimica_set_elementmass, THERMOCHIMICA_SET_ELEMENTMASS)
#define THERMOCHIMICA_SET_PRESSUREUNIT
FC_FUNC_(thermochimica_set_pressureunit, THERMOCHIMICA_SET_PRESSUREUNIT)
#define THERMOCHIMICA_SET_TEMPERATUREUNIT
FC_FUNC_(thermochimica_set_temperatureunit, THERMOCHIMICA_SET_TEMPERATUREUNIT)
#define THERMOCHIMICA_SET_ELEMENTMASSUNIT
FC_FUNC_(thermochimica_set_elementmassunit, THERMOCHIMICA_SET_ELEMENTMASSUNIT)
#define THERMOCHIMICA_SET_OUTPUT FC_FUNC_(thermochimica_set_output,
THERMOCHIMICA_SET_OUTPUT)
#define THERMOCHIMICA_PARSECSDATAFILE
FC_FUNC_(thermochimica_parsecsdatafile, THERMOCHIMICA_PARSECSDATAFILE)
#define THERMOCHIMICA_PARSECSDATAFILE
FC_FUNC_(thermochimica_parsecsdatafile, THERMOCHIMICA_PARSECSDATAFILE)
#define THERMOCHIMICA FC_FUNC_(thermochimica,THERMOCHIMICA)
#define THERMOCHIMICA_RESETTHERMO FC_FUNC_(thermochimica_resetthermo,
THERMOCHIMICA_RESETTHERMO)
#define THERMOCHIMICA_THERMODEBUG FC_FUNC_(thermodebug,
THERMOCHIMICA_THERMODEBUG)
#define THERMOCHIMICA_GET_NUMSPECIES
FC_FUNC_(thermochimica_get_numspecies, THERMOCHIMICA_GET_NUMSPECIES)
#define THERMOCHIMICA_GET_SPECIESMOLEFRACTIONS
FC_FUNC_(thermochimica_get_speciesmolefractions,
THERMOCHIMICA_GET_SPECIESMOLEFRACTIONS)
#define THERMOCHIMICA_GET_INFO FC_FUNC_(thermochimica_get_info,
THERMOCHIMICA_GET_INFO)

#else

#include "utils/FC.h"

#define THERMOCHIMICA_SET_PRESSURE FC_GLOBAL_(thermochimica_set_pressure,
THERMOCHIMICA_SET_PRESSURE)
#define THERMOCHIMICA_SET_TEMPERATURE
FC_GLOBAL_(thermochimica_set_temperature, THERMOCHIMICA_SET_TEMPERATURE)
```

```
#define THERMOCHIMICA_SET_ELEMENTMASS
FC_GLOBAL_(thermochimica_set_elementmass, THERMOCHIMICA_SET_ELEMENTMASS)
#define THERMOCHIMICA_SET_PRESSUREUNIT
FC_GLOBAL_(thermochimica_set_pressureunit, THERMOCHIMICA_SET_PRESSUREUNIT)
#define THERMOCHIMICA_SET_TEMPERATUREUNIT
FC_GLOBAL_(thermochimica_set_temperatureunit, THERMOCHIMICA_SET_TEMPERATUREUNIT)
#define THERMOCHIMICA_SET_ELEMENTMASSUNIT
FC_GLOBAL_(thermochimica_set_elementmassunit, THERMOCHIMICA_SET_ELEMENTMASSUNIT)
#define THERMOCHIMICA_SET_OUTPUT FC_GLOBAL_(thermochimica_set_output,
THERMOCHIMICA_SET_OUTPUT)
#define THERMOCHIMICA_PARSECSDATAFILE
FC_GLOBAL_(thermochimica_parsecsdatafile, THERMOCHIMICA_PARSECSDATAFILE)
#define THERMOCHIMICA FC_GLOBAL_(thermochimica, THERMOCHIMICA)
#define THERMOCHIMICA_RESETTHERMO FC_GLOBAL_(thermochimica_resetthermo,
THERMOCHIMICA_RESETTHERMO)
#define THERMOCHIMICA_THERMODEBUG FC_GLOBAL_(thermodebug,
THERMOCHIMICA_THERMODEBUG)
#define THERMOCHIMICA_GET_NUMSPECIES FC_GLOBAL_(thermochimica_get_numspecies,
THERMOCHIMICA_GET_NUMSPECIES)
#define THERMOCHIMICA_GET_SPECIESMOLEFRACTIONS
FC_GLOBAL_(thermochimica_get_speciesmolefractions,
THERMOCHIMICA_GET_SPECIESMOLEFRACTIONS)
#define THERMOCHIMICA_GET_INFO FC_GLOBAL_(thermochimica_get_info,
THERMOCHIMICA_GET_INFO)

#endif


extern "C"
{

void THERMOCHIMICA_SET_TEMPERATURE( double *dbl);
void THERMOCHIMICA_SET_PRESSURE( double *dbl);
void THERMOCHIMICA_SET_ELEMENTMASS( double *dbl);
void THERMOCHIMICA_SET_TEMPERATUREUNIT( int *length, const char&);
void THERMOCHIMICA_SET_PRESSUREUNIT( int *length, const char&);
void THERMOCHIMICA_SET_ELEMENTMASSUNIT( int *length, const char&);
void THERMOCHIMICA_SET_OUTPUT( int *length, const char&, int *lengthb, const
char&, int *lengthc, const char&, int *lengthd, const char&);
void THERMOCHIMICA_PARSECSDATAFILE( int *length, const char&);
void THERMOCHIMICA_RESETTHERMO();
void THERMOCHIMICA();
void THERMOCHIMICA_THERMODEBUG();

void THERMOCHIMICA_GET_NUMSPECIES( int *num);
void THERMOCHIMICA_GET_SPECIESMOLEFRACTIONS( double *dbl);
void THERMOCHIMICA_GET_INFO(int *INFO);

}

#endif
```

**Figure 3.2** – A sample C++ application programming interface.

## 3.5 USEFUL OUTPUT VARIABLES

There are several useful variables that are provided as output that may be used by another code. Table 3.1 summarizes many of these variables organized by their corresponding module. The variable naming convention adopted by Thermochimica precedes each variable name by a single lower case character that identifies the type of variable. This includes the prefixes `n-` to identify the number of something (e.g., the number of elements in the system), `i-` to identify an integer, `d-` to identify a double real, `c-` to identify a character and `l-` to identify a logical variable. There are a few exceptions to this naming convention, such as the coefficients `i` and `j` (that typically represent array coefficients) and the error code `INFOThermo`.

**Table 3.1** – Summary of useful variables provided as output from Thermochimica

| **Module Name** | **Variable Name (array dimension)** | **Brief Description** |
|---|---|---|
| ModuleThermo | `nElements` | An integer scalar representing the number of elements in the system. |
| | `nSpecies` | An integer scalar representing the number of species in the system. |
| | `nConPhases` | An integer scalar representing the number of coexisting pure condensed phases. |
| | `nSolnPhases` | An integer scalar representing the number of coexisting solution phases. |
| | `nSolnPhasesSys` | An integer scalar representing the total number of solution phases in system data-file. |
| | `iAssemblage (1:nElements)` | An integer vector representing the indices of the phases predicted to coexist at equilibrium. |
| | `iPhase (1:nSpecies)` | An integer vector representing the phase index of all species in the system. |
| | `dMolesPhase (1:nElements)` | A double real vector representing the number of moles of coexisting phases at equilibrium (corresponds directly to `iAssemblage`). |
| | `dMolesSpecies (1:nSpecies)` | A double real vector representing the number of moles of all species in the system. |
| | `dStdGibbsEnergy (1:nSpecies)` | A double real vector representing the standard Gibbs energies of all species in the system. |
| | `dChemicalPotential (1:nSpecies)` | A double real vector representing the chemical potentials of all species in the system. |
| | `dElementPotential (1:nElements)` | A double real vector representing the element potentials. |
| | `cElementName (1:nElements)` | A character vector representing the name of each chemical element in the system. |
| | `cSolnPhaseName (1:nSolnPhasesSys)` | A character vector representing the name of all solution phases in the system data-file. |
| | `cSpeciesName (1:nSpecies)` | A character vector representing the name of all species in the system data-file. |
| | `dMolFraction (1:nSpecies)` | A double real vector representing the mole fraction of each species in the system data-file. |
| ModuleThermoIO | `INFOThermo` | An integer scalar representing a successful exit or an error (used in a similar style as LAPACK). |
| ModuleGEMSolver | `dPartialExcessGibbs (1:nSpecies)` | A double real vector representing the partial molar excess Gibbs energy of mixing of each species in the system data-file. |

The integer vector `iAssemblage` stores the indices of all phases predicted to be stable at equilibrium. The dimension of this vector is defined as the number of elements in the system, thus the Gibbs phase rule is necessarily satisfied. The coefficients `1:nConPhases` in `iAssemblage` represent the pure condensed phases, the coefficients `nElements-nSolnPhases+1:nElements` represent solution phases and all other entries are zero. The values of the coefficients for pure condensed phases (which are positive) correspond directly to coefficients in `dStdGibbsEnergy` and `cSpeciesName`. The values of the coefficients for solution phases are stored as negative integers in `iAssemblage`. The absolute values of coefficients in `iAssemblage` corresponding to solution phases correspond directly to coefficients in `cSolnPhaseName` (see next section for an example). All phases in `iAssemblage` correspond directly to `dMolesPhase`. Also, the vector `iPhase` represents the phase index corresponding to each species in the system. Furthermore, the chemical elements represented in the character vector `cElementName` correspond directly to the element potentials in `dElementPotential`. Finally, the variable `INFOThermo` indicates a successful exit or an error. An example is provided in the next section.

### 3.6 EXAMPLE SYSTEM

Consider the U-O system from Figure 3.1, which contains 2 solution phases containing a total of 18 species and pure condensed phases. Table 3.2 summarizes pertinent variables that are provided by Thermochimica at thermodynamic equilibrium.

**Table 3.2** – Sample output data from the example given in Figure 3.1 in the format of Table 3.1. All double real variables are cropped to three significant figures.

| **Module Name** | **Variable Name (array dimension)** | **Value** |
|---|---|---|
| ModuleThermo | `nElements` | 2 |
| | `nSpecies` | 18 |
| | `nConPhases` | 1 |
| | `nSolnPhases` | 1 |
| | `nSolnPhasesSys` | 2 |
| | `iAssemblage (1:nElements)` | 16, -2 |
| | `iPhase (1:nSpecies)` | 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, -1, -1 |
| | `dMolesPhase (1:nElements)` | 1D-2, 0.96D0 |
| | `dMolesSpecies (1:nSpecies)` | 0D0, 0D0, 0D0, 0D0, 0D0, 0D0, 0D0, 319D0, 4.07D-9, 2.91D-42, 0D0, 0D0, 0D0, 0D0, 0D0, 0D0, 0D0, 0D0 |
| | `dMolFraction (1:nSpecies)` | 1.06D-38, 8.43D-31, 2.39D-64, 1.02D-119, 2.06D-89, 2.32D-55, 2.92D-39, 1D0, 1.28D-11, 9.11D-45, 0, 0, 0, 0, 0, 0, 0, 0 |
| | `dStdGibbsEnergy (1:nSpecies)` | 40.3, -25.1, 5.02, 97.0, -19.9, -145, -230, -271, -280, -123, -6.44, -6.13, -5.58, -860, -902, -1132, 240, 240 |
| | `dChemicalPotential (1:nSpecies)` | 0, 0, 0, 0, 0, 0, 0, -271, -318, -224, 0, 0, 0, 0, 0, 0, 0, 0 |
| | `dElementPotential (1:nElements)` | -176.9, -47.16 |
| | `cElementName (1:nElements)` | “U”, “O” |
| | `cSolnPhaseName (1:nSolnPhasesSys)` | “gas_ideal”, “fluorite” |

| | | |
|---|---|---|
| | cSpeciesName (1:nSpecies) | "O", "O2", "O3", "U", "UO", "UO2", "UO3", "UO2", "UO3", "UO", "U_alpha-solid_orthorh(s)", "U_beta-solid_tetrago(s2)", "U_gamma-solid_cubic(s3)", "U3O7_solid(s)", "U3O8_solid(s)", "U4O9_solid(s)", "U", "O" |
| ModuleThermoIO | INFOThermo | 0 |
| ModuleGEMSolver | dPartialExcessGibbs (1:nSpecies) | 0, 0, 0, 0, 0, 0, 0, 1.46D-23, -12.8, 1.63D-10, 0, 0, 0, 0, 0, 0, 0, 0 |

The first coefficient of the integer vector iAssemblage corresponds to the only pure condensed phase in the system and the second coefficient corresponds to the only solution phase in the system. Thus, the first coefficient in iAssemblage refers to the 16th species in the system, which corresponds to "U4O9_solid(s)" in the cSpeciesName vector. Similarly, the second coefficient in iAssemblage refers to the 2nd solution phase in the system, which corresponds to "fluorite" in the cSolnPhaseName vector.

One can compare the results from Table 3.2 to the equations established in §2. For example, one could compute the chemical potential of $UO_3$ in the fluorite phase with equation (3) using values from Table 3.2:

$$\begin{aligned} \tilde{\mu}_{UO_{3(fluroite)}} &= \tilde{g}^{o}_{UO_{3(fluroite)}} + \tilde{g}^{Ex}_{UO_{3(fluroite)}} + \ln x_{UO_{3(fluroite)}} \\ -318 &= -280 - 12.8 + \ln\left(1.28D - 11\right) \\ -318 &= -318 \end{aligned} \tag{8}$$

which can also be compared to equation (6),

$$\begin{aligned} \tilde{\mu}_{UO_{3(fluroite)}} &= \sum_{j=1}^{E} a_{UO_3,j}\tilde{\Gamma}_j \\ \tilde{\mu}_{UO_{3(fluroite)}} &= (1)(-176.9) + (3)(-47.2) \\ \tilde{\mu}_{UO_{3(fluroite)}} &= -318 \end{aligned} \tag{9}$$

Also, the chemical potential of $UO_3$ in the gaseous phase must be equivalent to $UO_3$ in the fluorite phase. The chemical potential of $UO_3$ in the gaseous phase is computed using equation (2)

$$\begin{aligned} \tilde{\mu}_{UO_{3(g)}} &= \tilde{g}^{o}_{UO_{3(g)}} + \ln a_{UO_{3(g)}} \\ -318 &= -230 + \ln\left(2.92D - 39\right) \\ -318 &= -318 \end{aligned} \tag{10}$$

To reduce memory requirements in a multi-physics code that makes use of Thermochimica, it is recommended that one stores dElementPotential instead of dChemicalPotential because the former requires significantly less memory than the latter, and one can compute dChemicalPotential for any solution phase constituent or pure condensed phase from dElementPotential.

## 3.7 ERRORS/DEBUGGING

The variable `INFOThermo` is used to identify a successful exit or an error that has been encountered by either the parser or Thermochimica. The subroutine ThermoDebug is available for debugging purposes, which prints an error message to the screen that corresponds to a particular value of `INFOThermo`. Table 3.3 summarizes the errors that may be encountered by Thermochimica.

**Table 3.3** – Summary of error codes from `INFOThermo`.

| **Integer value of** `INFOThermo` | **Description of error** |
|---|---|
| 0 | Successful exit. |
| 1 | Temperature is out of range or a NAN. |
| 2 | Pressure is out of range or a NAN. |
| 3 | Mass is out of range or a NAN. |
| 4 | The input variable cThermoInputUnits is not recognizable. |
| 5 | The number of elements in the system is out of range. |
| 6 | The specified data-file was not found. |
| 7 | There is an unknown error in reading the data-file. |
| 8 | The number of solution phases in the system exceeds the maximum allowable number. |
| 9 | A pure chemical species is needed to be present in the database for each element. |
| 10 | The Leveling subroutine failed to determine an appropriate phase assemblage for further consideration. |
| 11 | The PostLeveling subroutine failed. |
| 12 | The non-linear solver failed to converge. |
| 13 | The non-linear solver detected a NAN. |
| 14 | The non-linear solver determined that there are no solution phases present, but the system cannot be represented by only pure condensed phases. |
| 15 | Failed to deallocate allocatable arrays from the ModuleThermo, ModuleThermoIO, and/or ModuleGEMSolver modules. |
| 16 | The LAPACK driver routines were not able to invert the Jacobian matrix in the non-linear solver. |
| 17 | The solution phase type is not supported. |
| 18 | Failed to deallocate allocatable arrays used in the ModuleParseCS module. |
| 19 | Failed to deallocate allocatable arrays used in the CheckSystem subroutine. |
| 20 | Failed to deallocate allocatable arrays used in the LevelingSolver subroutine. |
| 21 | Failed to deallocate allocatable arrays used in the InitGEM subroutine. |
| 22 | Failed to deallocate allocatable arrays used in the CompMolSolnPhase subroutine. |
| 23 | Failed to deallocate allocatable arrays used in the GEMBroyden subroutine. |
| 24 | A NAN was detected in the CompStoichSolnPhase.f90 subroutine. |
| 25 | A NAN was detected in the CompMolFraction.f90 subroutine. |
| 26 | Failed to deallocate allocatable arrays used in the CompMolAllSolnPhases subroutine. |
| 27 | The CheckQKTOSolnPhase subroutine failed to converge. |
| [100,1000) | Error in reading line int{(`INFOThermo` – 100)} of the header section of the data-file. |
| [1000,10000) | Error in reading entry int{(`INFOThermo` – 1000)/100} of solution phase int{(`INFOThermo` – 1000 – 100 * int{(`INFOThermo` – 1000)/100}} of the data-block section of the data-file. |
| Other | An unknown error has occurred. |

# 4. BUILDING AND COMPILING THERMOCHIMICA

The directory structure of Thermochimica is illustrated in Figure 4.1 below. The *bin* directory stores executables, *data* stores thermodynamic data-files, *doc* stores dOxygen documentation files (in both html and Latex format), *obj* stores Fortran object files and *src* stores all source code. The *src* directory contains a *shared* subdirectory containing all shared source f90 files, and the *test* directory contains all unit and application tests. Tests that are performed on a daily and weekly basis are stored in the *daily* and *weekly* subdirectories, respectively. The main directory contains the Makefile, Doxyfile and modules that are built by Fortran. Additionally, two script files ("rundailytests" and "runweeklytests") are provided that run all daily and weekly tests.

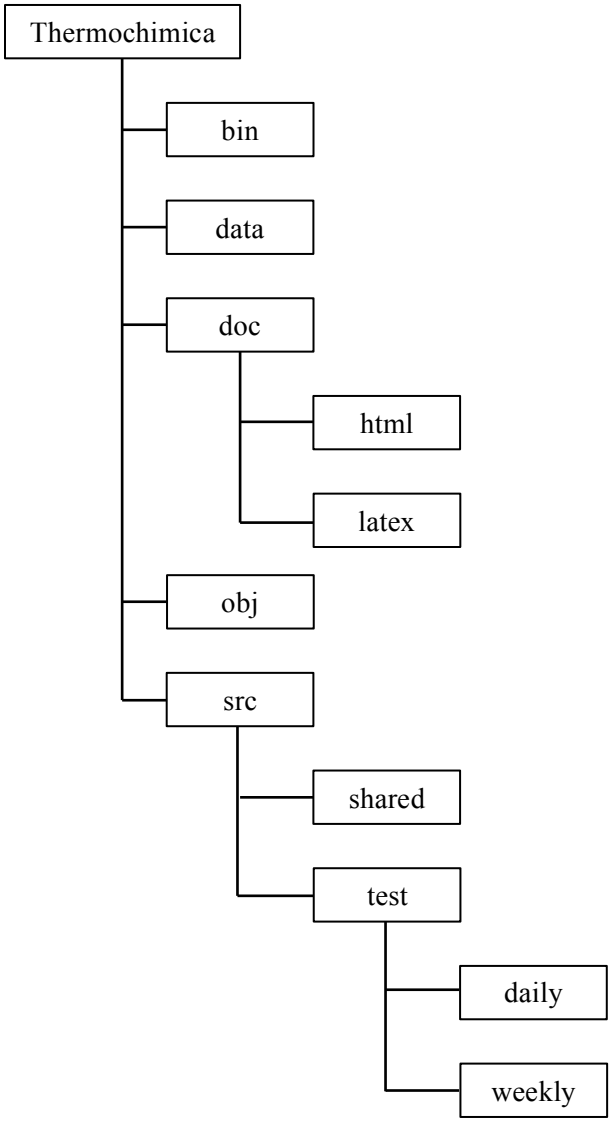


**Figure 4.1** – The data structure for Thermochimica is illustrated.

Thermochimica is compiled on a regular basis with both Intel and GNU compilers[5] on Mac OS-X and Linux operating systems. The only external library that Thermochimica is dependent on is LAPACK [13], which can be downloaded free of charge from the LAPACK web site[6]. The compiler can be specified with the *FC* variable in the Makefile, in addition to several compiler (i.e., *FCFLAGS*) and LAPACK (i.e., *LDFLAGS*) flags (if one chooses to changes them).

To compile Thermochimica, go to the Thermochimica directory in the Terminal and type `make`. By default, a sample executable called "thermo" is created. Table 4.1 summarizes the commands that can be used from the provided Makefile.

**Table 4.1** – Summary of all make commands

| Make command | Description |
|---|---|
| `make` | Compile Thermochimica with a default executable called "thermo". |
| `make test` | Compile all unit tests (executables located in the bin directory) |
| `make dailytest` | Compile all daily unit tests (executables located in the bin directory) |
| `make weeklytest` | Compile all weekly unit tests (executables located in the bin directory) |
| `make doc` | Compile dOxygen HTML and Latex documents |
| `make docHTML` | Compile dOxygen HTML documents only |
| `make doclatex` | Compile dOxygen Latex documents only |
| `make clean` | Clean object files |
| `make cleandoc` | Clean all dOxygen files |
| `make veryclean` | Clean all object files, dOxygen files, modules and executables |

To ensure that Thermochimica compiles without any issues and that it is producing correct results, the user is encouraged to run all daily and weekly unit and application tests. To run all daily tests, simply run the script file `./rundailytests` from the Terminal from the Thermochimica directory. Similarly, all weekly tests can be executed by running the script file `./runweeklytests`. Thermochimica will return the following for a successful test:

```
TestThermo01: PASS
```

and the following for a failure:

```
TestThermo01: FAIL <---
```

In the event that any tests fail, please contact M.H.A. Piro at piromh@ornl.gov or markuspiro@gmail.com with the data-file that was used and a detailed description of the circumstances of the error.

[5] The most recent version of these compilers that have been used are Intel Fortran 11.1 and gfortran 4.6.3.
[6] The LAPACK linear algebra library can be downloaded from http://www.netlib.org/lapack/

# 5. CONCLUSIONS

A new equilibrium thermochemistry library called Thermochimica has been described that is intended for direct integration into multi-physics codes. Specifically, this solver is intended to provide input to thermodynamic material properties and boundary conditions for continuum-scale and meso-scale simulations. The development of this software is specifically aimed at addressing concerns with integrating commercial thermodynamic software into multi-physics codes. First of all, the development of new software facilitates the distribution of software that could otherwise be complicated by licensing issues. Secondly, advancements in the numerical approach in computing thermodynamic equilibria enhance computational performance and robustness, which is especially needed when integrated into large multi-physics codes. Finally, the software has been intentionally designed from the beginning to handle extraordinarily large thermodynamic systems, thus eliminating any concern of software limitations regarding the number of chemical elements, species, pure condensed phases or solution phases.

The general application and use of Thermochimica have been reviewed in this document with an example Fortran application programming interface that demonstrates how it can be called from another code. Additionally, a sample problem is provided that demonstrates how one could make use of the data computed by Thermochimica. Finally, specific programming details of each variable and subroutine are described in a dOxygen report in the Appendix.

## 6. ACKNOLWEDGEMENTS

The primary author thanks W.T. Thompson and B.J. Lewis from the Royal Military College of Canada for initiating this work as part of his PhD dissertation. Technical discussions with B. Sundman, V. Protopopsecu, R. Sampath, K.T. Clarno and members of the AMP team are gratefully acknowledged. The development of the Advanced Multi-Physics (AMP) nuclear fuel performance code was funded by the Fuels Integrated Performance and Safety Code (IPSC) element of the Nuclear Energy Advanced Modeling and Simulations (NEAMS) program of the US Department of Energy Office of Nuclear Energy (DOE/NE), Advanced Modeling and Simulation Office (AMSO).

# 7. APPENDIX

The dOxygen generated documentation from Thermochimica 1.0 is provided, which is generated automatically from comments within the source code. A Latex report is generated in addition to an HTML web site. The HTML documentation is recommended for frequent use because of the convenience in navigating the documentation and source code. Page numbering and section labels were generated independently by dOxygen and do not correspond directly to this document. The general outline of the dOxygen report includes an introduction, data type index, file index, data type documentation and finally file documentation.

# Thermochimica

Generated by Doxygen 1.7.6.1

Tue Nov 27 2012 14:40:47

# Contents

# Chapter 1

# Introduction to Thermochimica

## 1.1 Introduction

The purpose of Thermochimica is to compute the quantities of species and phases at thermodynamic equilibrium and various other thermodynamic quantities. For a detailed discussion on numerical methods employed in this work and the principles of computational thermodynamics, refer to the following literature:

- M.H.A. Piro and B. Sundman, CALPHAD, to be published.
- M.H.A. Piro, S. Simunovic, T.M. Besmann, B.J. Lewis, W.T. Thompson, "The Thermochemistry Library Thermochimica," Computational Materials Science, 67 (2013) 266-272.
- M.H.A. Piro and S. Simunovic, "Performance Enhancing Algorithms for Computing Thermodynamic Equilibria," CALPHAD, 39 (2012) 104-110.
- M.H.A. Piro, "Computation of Thermodynamic Equilibria Pertinent to Nuclear Materials in Multi-Physics Codes," PhD Thesis, Royal Military College of Canada (2011).
- M.H.A. Piro, T.M. Besmann, S. Simunovic, B.J. Lewis, W.T. Thompson, "-Numerical Verification of Equilibrium Computations in Nuclear Fuel Performance Codes," Journal of Nuclear Materials, 414 (2011) 399-407.

The required input variables include:

- dTemperature: double real scalar, represents temperature;
- dPressure: double real scalar, represents absolute hydrostatic pressure;
- dElementMass: double real vector [0:118], represents mass of each chemical element;
- cThermoInputUnits: character vector [1:3], represents the units of the input variables;

- cThermoFileName: character scalar, represents data-file path and name.

Useful information that Thermochimica provides as output include:

- the identification of phases predicted to be stable at equilibrium (i.e., iAssemblage);
- the number of moles (i.e., dMolesPhase) and the mole fraction (i.e., dMolFraction) of each solution phase constituent;
- the number of moles of each pure condensed phase (i.e., dMolesPhase);
- the chemical potentials of all species and phases (i.e., dChemicalPotential); and
- the chemical potentials of the component elements (i.e., dElementPotential).

Thermochimica returns the variable INFOThermo (scalar integer) indictating a successful computation or an error that has been identified. A description of each possible value of INFOThermo is given in the ThermoDebug.f90 subroutine.

Any suggestions to improve documentation and/or programming are always welcome and appreciated. Please send all suggestions to Markus Piro at piromh@ornl.gov or markuspiro@gmail.com.

## 1.2 Overview

Thermochimica is written in Fortran using the F90/F03 standard. There are two calls required to make use of this library: 1) a call to the data-file parser (i.e., ParseCSDataFile.f90), and 2) a call to Thermochimica (i.e., Thermochimica.f90). See thermo.f90 for an example wrapper executable that calls the parser and Thermochimica with input variables described above. Finally, both the parser and Thermochimica are reset by calling ResetThermoAll.f90.

The main subroutines called by ParseCSDataFile.f90 are as follows:

| File name | Description |
|---|---|
| ParseCSDataFile.f90 | Open the specified ChemSage data-file and return an error if necessary. |
| ParseCSHeader.f90 | Parse the header section of a ChemSage data-file. |
| ParseCSDataBlock.f90 | Parse the data-block section of a ChemSage data-file. |
| ParseCSDataBlockGibbs.f90 | Parse the coefficients of the Gibbs energy equations in the datablock section of a ChemSage data-file. |
| ModuleParseCS.f90 | Module containing variables used by the parser. |

The main subroutines called by Thermochimica.f90 are as follows:

| File name | Description |
|---|---|
| CheckThermoInput.f90 | Check the input variables to Thermochimica and return an error if inappropriate. |
| InitThermo.f90 | Initialize Thermochimica (e.g., physical constants and numerical tolerances). |
| CheckSystem.f90 | Check for consistency between the data-file and the current system. |
| CompThermoData.f90 | Compute thermodynamic data (e.g., standard Gibbs energies, etc.). |
| CheckThermoData.f90 | Check the thermodynamic data to ensure it is appropriate. |
| LevelingSolver.f90 | The Leveling Solver estimates the equilibrium phase assemblage assuming that all solution phase constituents and pure condensed phase may be initially treated as pure condensed phasess. |
| PostLevelingSolver.f90 | The Post-Leveling Solver further improves upon the estimates from Leveling by including compositional dependent terms to solution phase constituents. |
| GEMSolver.f90 | The Gibbs Energy Minimization (GEM) Solver computes thermodynamic equilibrium including non-ideal mixing behaviour. |

## 1.3 Style

All of the associated subroutines employ the following variable naming convention:

| Prefix | Description |
|---|---|
| i | Integer variable |
| n | Number of something (e.g., nElements refers to the number of elements in the system) |
| d | Double real variable (i.e., real(8)) |
| c | Character variable |
| l | Logical variable |

## 1.4 Glossary

The following gives a brief description of thermodynamics nomenclature used in the soure code.

| Term | Description |
|---|---|
| Activity | A dimensionless quantity related to the chemical potential of a substance and is represented by $a_i$. The activity is equivalent to mole fraction for an ideal solution phase. |
| Activity Coefficient | The activity coefficient accounts for the departure of a substance from ideal behaviour and is represented by $\gamma_i$. This is related to the Partial molar excess Gibbs energy of mixing. |
| Aqueous phase | A particular solution phase where the solvent is water and many of the solutes are electrically charged ions. |
| Chemical potential | A measure of the effect on the Gibbs energy of the system by the introduction of a substance. The chemical potential is defined as $\mu_i = \frac{\partial G_{sys}}{\partial n_i}\vert_{T,P,n_{k\neq i}}$, which yields $\mu_i = g_i^\circ + RTln(a_i)$. |
| Closed system | A system that permits the exchange of heat and work with its surroundings at constant mass. |
| Constituent | A constituent of a solution phase refers to a particular species in a particular phase. |
| Element | A chemical element, which is not to be confused with a nuclear fuel element or an element of a vector/matrix. |
| Gibbs energy | A thermodynamic potential measuring the maximum amount of useful work obtainable from an isothermal-isobaric closed system. The Gibbs energy, represented as G, is defined as the difference between enthalpy and the product of temperature and entropy $G = H - TS$. |
| Ion | An atom or molecule where the number of electrons does not equal the number of protons. |
| Isobaric | A system at constant pressure. |
| Isothermal | A system at constant temperature. |
| Molality | Molality denotes the number of moles of a solute i per kilogram of solvent (not solution). |
| Mole | A quantity of mass measured as 6.02214179E23 atoms. Equivalent to gram-atom for a pure element. |
| Mole fraction | The fraction of moles of a particular species in a particular solution phase. |

| Term | Description |
|---|---|
| Partial molar excess Gibbs energy of mixing | The partial molar excess Gibbs energy of mixing, represented as $g_i^{ex}$, represents the contribution to the chemical potential term due to non-ideal behaviour. This is the difference between the real chemical potential of a substance and that if assuming ideal mixing behaviour. |
| Phase | A body of matter that is uniform in chemical composition and physical state. Phases are separated from one another by a physical discontinuity. A phase is not to be confused with a state of matter. For example, there are three different phases of uranium in a solid state (orthogonal, tetragonal and body centred cubic). |
| Phase assemblage | A unique combination of phases predicted to be stable at equilibrium. |
| Pure condensed phase | A condensed phase with invariant stoichiometry and may be interpreted mathematically as containing a single species with unit concentration. |
| Solution phase | A phase containing a mixture of multiple species. A solution phase can be in a gaseous, liquid or solid state. |
| Species | A chemically distinct molecular entity. For example, H2O has a distinct chemical composition, but can be in gaseous, liquid or solid phases. This differs from the term constituent, which refers to a particular species in a particular phase. |
| Standard molar Gibbs energy | The standard molar Gibbs energy of a pure species, represented as $g_i^{\circ}$, is the Gibbs energy of that species with unit activity. This quantity is computed using values from the ChemSage data-file. |
| State | A state of matter distinguishes the distinct form that matter may take on. This includes solid, liquid, gas and plasma...and for you physicists, the Bose-Einstein condensate. This is not to be confused with the term "phase". |
| Stoichiometry | This refers to the relative amounts of atoms per formula mass of a substance. |
| System | A portion of the Universe with a perimeter defined by real or imaginary boundaries. |

# Chapter 2

# Data Type Index

## 2.1 Data Types List

Here are the data types with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Data Type Documentation

## 4.1 ModuleGEMSolver Module Reference

### Public Attributes

- integer iterLast
- integer iterStep
- integer iterRevert
- integer iterGlobal
- integer iterLastCon
- integer iterLastSoln
- integer iterSwap
- integer iterLastMiscGapCheck
- integer iConPhaseLast
- integer iSolnPhaseLast
- integer iSolnSwap
- integer iPureConSwap
- integer, parameter iterGlobalMax = 2000
- integer, dimension(:,:), allocatable iterHistory
- real(8) dGEMFunctionNorm
- real(8) dGEMFunctionNormLast
- real(8), dimension(:), allocatable dSumMolFractionSoln
- real(8), dimension(:), allocatable dMolesPhaseLast
- real(8), dimension(:), allocatable dUpdateVar
- real(8), dimension(:), allocatable dDrivingForceSoln
- real(8), dimension(:), allocatable dPartialExcessGibbs
- real(8), dimension(:), allocatable dPartialExcessGibbsLast
- real(8), dimension(:,:), allocatable dEffStoichSolnPhase
- logical lDebugMode
- logical lRevertSystem
- logical lConverged
- logical, dimension(:), allocatable lSolnPhases
- logical, dimension(:), allocatable lMiscibility

### 4.1.1 Detailed Description

**Parameters**

| | |
|---|---|
| *iterLast* | The last global iteration that the phase assemblage was adjusted. |
| *iterHistory* | An integer matrix representing all of the indices of phases that contribute to the equilibrium phase assemblage at each stage in the iteration history. |
| *dGEMFunctionNorm* | A double real scalar representing the norm of the functional vector in the GEMSolver. |
| *dGEMFunctionNormLast* | A double real scalar representing the norm of the functional vector in the GEMSolver from the last iteration. |
| *dSumMolFractionSoln* | A double real vector representing the sum of mole fractions in each solution phase. |
| *dUpdateVar* | A double real vector representing the direction vector that updates the function |
| *dPartialExcessGibbs* | A double real vector representing the partial molar excess Gibbs energy of mixing of each species in the system. |
| *dEffStoichSolnPhase* | A double real matrix representing the effective stoichiometry of each solution phase. |
| *lDebugMode* | A logical variable used for debugging purposes. When it is TRUE, a number of print statements are applied. |
| *lRevertSystem* | A logical variable identifying whether the system should be reverted (TRUE) or not (FALSE). |
| *lConverged* | A logical variable identifying whether the system has converged (TRUE) or not (FALSE). |
| *lSolnPhases* | A logical vector indicating whether a particular solution phase is currently assumed to be stable (true) or not (false). |
| *lMiscibility* | A logical vector indicating whether a particular solution phase has a miscibility gap (true) or not (false). |

Definition at line 54 of file ModuleGEMSolver.f90.

### 4.1.2 Member Data Documentation

#### 4.1.2.1 real(8), dimension(:), allocatable ModuleGEMSolver::dDrivingForceSoln

Definition at line 67 of file ModuleGEMSolver.f90.

#### 4.1.2.2 real(8), dimension(:,:), allocatable ModuleGEMSolver::dEffStoichSolnPhase

Definition at line 69 of file ModuleGEMSolver.f90.

#### 4.1.2.3 real(8) ModuleGEMSolver::dGEMFunctionNorm

Definition at line 66 of file ModuleGEMSolver.f90.

#### 4.1.2.4 real(8) ModuleGEMSolver::dGEMFunctionNormLast

Definition at line 66 of file ModuleGEMSolver.f90.

#### 4.1.2.5 real(8), dimension(:), allocatable ModuleGEMSolver::dMolesPhaseLast

Definition at line 67 of file ModuleGEMSolver.f90.

#### 4.1.2.6 real(8), dimension(:), allocatable ModuleGEMSolver::dPartialExcessGibbs

Definition at line 68 of file ModuleGEMSolver.f90.

#### 4.1.2.7 real(8), dimension(:), allocatable ModuleGEMSolver::dPartialExcessGibbsLast

Definition at line 68 of file ModuleGEMSolver.f90.

#### 4.1.2.8 real(8), dimension(:), allocatable ModuleGEMSolver::dSumMolFractionSoln

Definition at line 67 of file ModuleGEMSolver.f90.

#### 4.1.2.9 real(8), dimension(:), allocatable ModuleGEMSolver::dUpdateVar

Definition at line 67 of file ModuleGEMSolver.f90.

#### 4.1.2.10 integer ModuleGEMSolver::iConPhaseLast

Definition at line 62 of file ModuleGEMSolver.f90.

#### 4.1.2.11 integer ModuleGEMSolver::iPureConSwap

Definition at line 62 of file ModuleGEMSolver.f90.

#### 4.1.2.12 integer ModuleGEMSolver::iSolnPhaseLast

Definition at line 62 of file ModuleGEMSolver.f90.

#### 4.1.2.13 integer ModuleGEMSolver::iSolnSwap

Definition at line 62 of file ModuleGEMSolver.f90.

#### 4.1.2.14 integer ModuleGEMSolver::iterGlobal

Definition at line 60 of file ModuleGEMSolver.f90.

#### 4.1.2.15 integer, parameter ModuleGEMSolver::iterGlobalMax = 2000

Definition at line 63 of file ModuleGEMSolver.f90.

#### 4.1.2.16 integer, dimension(:,:), allocatable ModuleGEMSolver::iterHistory

Definition at line 64 of file ModuleGEMSolver.f90.

#### 4.1.2.17 integer ModuleGEMSolver::iterLast

Definition at line 60 of file ModuleGEMSolver.f90.

#### 4.1.2.18 integer ModuleGEMSolver::iterLastCon

Definition at line 61 of file ModuleGEMSolver.f90.

#### 4.1.2.19 integer ModuleGEMSolver::iterLastMiscGapCheck

Definition at line 61 of file ModuleGEMSolver.f90.

#### 4.1.2.20 integer ModuleGEMSolver::iterLastSoln

Definition at line 61 of file ModuleGEMSolver.f90.

#### 4.1.2.21 integer ModuleGEMSolver::iterRevert

Definition at line 60 of file ModuleGEMSolver.f90.

#### 4.1.2.22 integer ModuleGEMSolver::iterStep

Definition at line 60 of file ModuleGEMSolver.f90.

#### 4.1.2.23 integer ModuleGEMSolver::iterSwap

Definition at line 61 of file ModuleGEMSolver.f90.

#### 4.1.2.24 logical ModuleGEMSolver::lConverged

Definition at line 71 of file ModuleGEMSolver.f90.

#### 4.1.2.25 logical ModuleGEMSolver::lDebugMode

Definition at line 71 of file ModuleGEMSolver.f90.

#### 4.1.2.26 logical, dimension(:), allocatable ModuleGEMSolver::lMiscibility

Definition at line 72 of file ModuleGEMSolver.f90.

#### 4.1.2.27 logical ModuleGEMSolver::lRevertSystem

Definition at line 71 of file ModuleGEMSolver.f90.

#### 4.1.2.28 logical, dimension(:), allocatable ModuleGEMSolver::lSolnPhases

Definition at line 72 of file ModuleGEMSolver.f90.

The documentation for this module was generated from the following file:

- src/shared/ModuleGEMSolver.f90

## 4.2 ModuleParseCS Module Reference

### Public Attributes

- integer nElementsCS
- integer nSpeciesCS
- integer nSolnPhasesSysCS
- integer INFO
- integer nMagneticTermsCS
- integer nParamCS
- integer, parameter nSolnPhasesSysMax = 42
- integer, parameter nGibbsCoeff = 13
- integer, parameter nMaxGibbsEqs = 6
- integer, parameter nParamMax = 4
- integer, dimension(:), allocatable nSpeciesPhaseCS

- integer, dimension(:), allocatable nGibbsEqSpecies
- integer, dimension(:), allocatable iPhaseCS
- integer, dimension(:), allocatable iParticlesPerMoleCS
- integer, dimension(:), allocatable nParamPhaseCS
- integer, dimension(:), allocatable iParamPassCS
- integer, dimension(:,:), allocatable iSpeciesAtomsCS
- integer, dimension(:,:), allocatable iRegularParamCS
- real(8), dimension(:), allocatable dAtomicMass
- real(8), dimension(:,:), allocatable dGibbsCoeffSpeciesTemp
- real(8), dimension(:,:), allocatable dRegularParamCS
- real(8), dimension(:,:), allocatable dGibbsMagneticCS
- character(3), dimension(:), allocatable cElementNameCS
- character(8), dimension(:), allocatable cSolnPhaseTypeCS
- character(25), dimension(:), allocatable cSolnPhaseNameCS
- character(25), dimension(:), allocatable cSpeciesNameCS

### 4.2.1 Detailed Description

**Parameters**

| | |
|---|---|
| *INFO* | A scalar integer that indicates a successful exit or identifies an error. |
| *nElements-CS* | Number of elements in the system. |
| *nSoln-PhasesSys-CS* | Number of solution phases in the system. |
| *nSoln-PhasesSys-Max* | Maximum number of solution phases in a system that can be considered. |
| *nSpecies-PhaseCS* | Number of species in a solution phase. |
| *nSpeciesCS* | Number of species in the system (combined solution species and pure separate phases). |
| *nGibbsEq-Species* | Number of Gibbs energy equations for a particular species. |
| *nGibbsCoeff* | Number of coefficients for a Gibbs energy equation. |
| *nMaxGibbs-Eqs* | Maximum number of Gibbs energy equations per species. |
| *nParamMax* | Maximum number of parameters in a sub-system of a non-ideal solution phase. The default is set to 4 (i.e., quaternary). |
| *iSpecies-Atoms* | Integer matrix representing the number of atoms of a particular elements in a species (i.e., stoichiometry matrix). |
| *iRegular-Param* | Integer matrix representing the component numbers and exponents in a regular solution phase. |
| *iPhase* | Integer vector containing the index of the phase that a particular species belongs to. iPhase = 0 for a pure separate phase, iPhase = -1 for a "dummy species", iPhase $>$ 0 for a solution species, where the number corresponds to the solution phase index. |

| | |
|---|---|
| *iParticles-PerMoleCS* | An integer vector containing the number of particles per mole of the constituent species formula mass. The default value is 1. |
| *cSystemTitle* | A character string representing the name of the system. |
| *cDummy* | A dummy character variable. |
| *cElement-Name* | The name of a chemical element. |
| *cSpecies-NameCS* | The name of a species (short hand). Note that this can be a solution species or pure condensed phase. |
| *cSolnPhase-Name* | The name of a solution phase. |
| *cSolnPhase-Type* | The type of a solution phase. |
| *dGibbs-Dummy* | An abritrary value for the molar standard Gibbs energy that is applied |
| *dAtomic-Mass* | Atomic mass of an element. |
| *dGibbs-Coeff-Species-Temp* | Temporary double array of coefficients for a Gibbs energy equation. |

Definition at line 47 of file ModuleParseCS.f90.

### 4.2.2 Member Data Documentation

#### 4.2.2.1 character(3), dimension(:), allocatable ModuleParseCS::cElementNameCS

Definition at line 64 of file ModuleParseCS.f90.

#### 4.2.2.2 character(25), dimension(:), allocatable ModuleParseCS::cSolnPhaseNameCS

Definition at line 66 of file ModuleParseCS.f90.

#### 4.2.2.3 character(8), dimension(:), allocatable ModuleParseCS::cSolnPhaseTypeCS

Definition at line 65 of file ModuleParseCS.f90.

#### 4.2.2.4 character(25), dimension(:), allocatable ModuleParseCS::cSpeciesNameCS

Definition at line 67 of file ModuleParseCS.f90.

#### 4.2.2.5 real(8), dimension(:), allocatable ModuleParseCS::dAtomicMass

Definition at line 61 of file ModuleParseCS.f90.

#### 4.2.2.6 real(8), dimension(:,:), allocatable ModuleParseCS::dGibbsCoeffSpeciesTemp

Definition at line 62 of file ModuleParseCS.f90.

#### 4.2.2.7 real(8), dimension(:,:), allocatable ModuleParseCS::dGibbsMagneticCS

Definition at line 62 of file ModuleParseCS.f90.

#### 4.2.2.8 real(8), dimension(:,:), allocatable ModuleParseCS::dRegularParamCS

Definition at line 62 of file ModuleParseCS.f90.

#### 4.2.2.9 integer ModuleParseCS::INFO

Definition at line 53 of file ModuleParseCS.f90.

#### 4.2.2.10 integer, dimension(:), allocatable ModuleParseCS::iParamPassCS

Definition at line 58 of file ModuleParseCS.f90.

#### 4.2.2.11 integer, dimension(:), allocatable ModuleParseCS::iParticlesPerMoleCS

Definition at line 57 of file ModuleParseCS.f90.

#### 4.2.2.12 integer, dimension(:), allocatable ModuleParseCS::iPhaseCS

Definition at line 57 of file ModuleParseCS.f90.

#### 4.2.2.13 integer, dimension(:,:), allocatable ModuleParseCS::iRegularParamCS

Definition at line 59 of file ModuleParseCS.f90.

#### 4.2.2.14 integer, dimension(:,:), allocatable ModuleParseCS::iSpeciesAtomsCS

Definition at line 59 of file ModuleParseCS.f90.

#### 4.2.2.15 integer ModuleParseCS::nElementsCS

Definition at line 53 of file ModuleParseCS.f90.

#### 4.2.2.16 integer, parameter ModuleParseCS::nGibbsCoeff = 13

Definition at line 56 of file ModuleParseCS.f90.

#### 4.2.2.17 integer, dimension(:), allocatable ModuleParseCS::nGibbsEqSpecies

Definition at line 57 of file ModuleParseCS.f90.

#### 4.2.2.18 integer ModuleParseCS::nMagneticTermsCS

Definition at line 54 of file ModuleParseCS.f90.

#### 4.2.2.19 integer, parameter ModuleParseCS::nMaxGibbsEqs = 6

Definition at line 56 of file ModuleParseCS.f90.

#### 4.2.2.20 integer ModuleParseCS::nParamCS

Definition at line 54 of file ModuleParseCS.f90.

#### 4.2.2.21 integer, parameter ModuleParseCS::nParamMax = 4

Definition at line 56 of file ModuleParseCS.f90.

#### 4.2.2.22 integer, dimension(:), allocatable ModuleParseCS::nParamPhaseCS

Definition at line 58 of file ModuleParseCS.f90.

#### 4.2.2.23 integer ModuleParseCS::nSolnPhasesSysCS

Definition at line 53 of file ModuleParseCS.f90.

#### 4.2.2.24 integer, parameter ModuleParseCS::nSolnPhasesSysMax = 42

Definition at line 55 of file ModuleParseCS.f90.

#### 4.2.2.25 integer ModuleParseCS::nSpeciesCS

Definition at line 53 of file ModuleParseCS.f90.

#### 4.2.2.26 integer, dimension(:), allocatable ModuleParseCS::nSpeciesPhaseCS

Definition at line 57 of file ModuleParseCS.f90.

The documentation for this module was generated from the following file:

- src/shared/Parser/ModuleParseCS.f90

## 4.3 ModuleSubMin Module Reference

### Public Attributes

- integer nVar
- integer iFirst
- integer iLast
- integer iSolnPhaseIndexOther
- real(8) dDrivingForce
- real(8) dDrivingForceLast
- real(8), parameter dSubMinTolerance = 1D-3
- real(8), parameter dMinMoleFraction = 1D-100
- real(8), parameter dTolEuclideanNorm = 1D-2
- real(8), parameter dTolDrivingForceChange = 1D-3
- real(8), dimension(:), allocatable dChemicalPotentialStar
- real(8), dimension(:), allocatable dRHS
- logical lSubMinConverged

### 4.3.1 Detailed Description

**Parameters**

| | |
|---|---|
| *nVar* | The number of species in the specified solution phase. |
| *iFirst* | The first species in the specified solution phase. |
| *iLast* | The last species in the specified solution phase. |
| *iterSubMax* | The maximum number of iterations in the subminimization routine. |
| *dDriving-Force* | The driving force of the specified solution phase. |
| *dSubMin-Tolerance* | A numerical tolerance of the maximum change in dMolFraction that is used to identify convergence. |
| *dChemical-PotentialStar* | The chemical potential of each solution phase constituent defined by the element potentials. |
| *dRHS* | A working vector that is used to represent the functional vector in the SubMinNewton subroutine and the direction vector that is computed. |
| *dHessian* | A double real array representing the Hessian matrix. |
| *lSubMin-Converged* | A logical scalar indicating a converged (i.e., TRUE) or non-converged solution (i.e., FALSE). |

| | |
|---|---|
| *dTolEuclideanNorm* | A double real scalar representing the tolerance of the Euclidean norm between the mole fraction vectors of two corresponding solution phases. |
| *dTolDrivingForceChange* | A double real scalar represending the tolerance for the change in the driving force. |

Definition at line 31 of file ModuleSubMin.f90.

### 4.3.2 Member Data Documentation

#### 4.3.2.1 real(8), dimension(:), allocatable ModuleSubMin::dChemicalPotentialStar

Definition at line 42 of file ModuleSubMin.f90.

#### 4.3.2.2 real(8) ModuleSubMin::dDrivingForce

Definition at line 39 of file ModuleSubMin.f90.

#### 4.3.2.3 real(8) ModuleSubMin::dDrivingForceLast

Definition at line 39 of file ModuleSubMin.f90.

#### 4.3.2.4 real(8), parameter ModuleSubMin::dMinMoleFraction = 1D-100

Definition at line 40 of file ModuleSubMin.f90.

#### 4.3.2.5 real(8), dimension(:), allocatable ModuleSubMin::dRHS

Definition at line 42 of file ModuleSubMin.f90.

#### 4.3.2.6 real(8), parameter ModuleSubMin::dSubMinTolerance = 1D-3

Definition at line 40 of file ModuleSubMin.f90.

#### 4.3.2.7 real(8), parameter ModuleSubMin::dTolDrivingForceChange = 1D-3

Definition at line 41 of file ModuleSubMin.f90.

#### 4.3.2.8 real(8), parameter ModuleSubMin::dTolEuclideanNorm = 1D-2

Definition at line 41 of file ModuleSubMin.f90.

#### 4.3.2.9 integer ModuleSubMin::iFirst

Definition at line 37 of file ModuleSubMin.f90.

#### 4.3.2.10 integer ModuleSubMin::iLast

Definition at line 37 of file ModuleSubMin.f90.

#### 4.3.2.11 integer ModuleSubMin::iSolnPhaseIndexOther

Definition at line 37 of file ModuleSubMin.f90.

#### 4.3.2.12 logical ModuleSubMin::lSubMinConverged

Definition at line 44 of file ModuleSubMin.f90.

#### 4.3.2.13 integer ModuleSubMin::nVar

Definition at line 37 of file ModuleSubMin.f90.

The documentation for this module was generated from the following file:

- src/shared/ModuleSubMin.f90

## 4.4 ModuleThermo Module Reference

### Public Attributes

- integer nElements
- integer nSpecies
- integer nParam
- integer nMaxParam
- integer nDummySpecies
- integer nConPhases
- integer nSolnPhases
- integer nSolnPhasesSys
- integer, parameter iTolNum = 15
- integer, parameter nElementsPT = 118
- integer, dimension(:), allocatable iSpeciesTotalAtoms
- integer, dimension(:), allocatable iPhase
- integer, dimension(:), allocatable nSpeciesPhase
- integer, dimension(:), allocatable iParticlesPerMole
- integer, dimension(:), allocatable iAssemblage
- integer, dimension(:), allocatable nParamPhase

- integer, dimension(:), allocatable iElementSystem
- integer, dimension(:), allocatable iSpeciesPass
- integer, dimension(:,:), allocatable iSpeciesAtoms
- integer, dimension(:,:), allocatable iRegularParam
- integer, dimension(:,:), allocatable iterHistoryLevel
- real(8) dIdealConstant
- real(8) dNormalizeSum
- real(8) dNormalizeInput
- real(8), dimension(itolnum) dTolerance
- real(8), dimension(:), allocatable dStdGibbsEnergy
- real(8), dimension(:), allocatable dGibbsSolnPhase
- real(8), dimension(:), allocatable dMolesSpecies
- real(8), dimension(:), allocatable dChemicalPotential
- real(8), dimension(:), allocatable dExcessGibbsParam
- real(8), dimension(:), allocatable dLevel
- real(8), dimension(:), allocatable dElementPotential
- real(8), dimension(:), allocatable dMolesPhase
- real(8), dimension(:), allocatable dMolesElement
- real(8), dimension(:), allocatable dMolFraction
- real(8), dimension(:,:), allocatable dAtomFractionSpecies
- character(3), dimension(:), allocatable cElementName
- character(25), dimension(:), allocatable cSpeciesName
- character(8), dimension(:), allocatable cSolnPhaseType
- character(25), dimension(:), allocatable cSolnPhaseName

### 4.4.1 Detailed Description

**Parameters**

| | |
|---|---|
| *nElements* | The number of elements in the system. |
| *nElementsPT* | The number of chemical elements on the periodic table. |
| *nSpecies* | The number of species in the system. |
| *nSpeciesPhase* | An integer vector representing the number of species in each solution phase. |
| *nParam* | The number of mixing parameters in the system. |
| *nParamPhase* | The number of mixing parameters in each solution phase. |
| *nMaxParam* | The maximum number of parameters that are allowed. |
| *nDummySpecies* | The number of dummy species that have been added to a ChemSage data-file from FactSage. |
| *nConPhases* | The number of pure condensed phases predicted to be stable at equilibrium. |
| *nSolnPhases* | The number of solution phases predicted to be stable at equilibrium. |
| *nSolnPhasesSys* | The number of solution phases in the system (not necessarily stable at equilibrium). |

| | |
|---|---|
| *iTolNum* | The number of numerical tolerances (used only for allocation purposes). |
| *iSpecies-TotalAtoms* | The total number of atoms per formula mass of a species. |
| *iPhase* | An integer vector representing the phase type (0: pure condensed phase; $>$ 0: solution phase index; -1: dummy species). |
| *iParticles-PerMole* | The number of particles per mole of constituent. |
| *iAssemblage* | Integer vector containing the indices of phases estimated to be part of the equilibrium phase assemblage. |
| *iSpecies-Pass* | An integer vector that is used soley to determine whether a particular species will be considered in the system. |
| *iSpecies-Atoms* | The number of atoms of a particular element for a particular species. |
| *iRegular-Param* | An integer matrix representing information pertient to regular solution models. The first coefficient represents the number of components in the sub-system and the other coefficients represent the indices of components in the sub-system. |
| *iterHistory-Level* | An integer matrix representing all of the indices of phases that contribute to the equilibrium phase assemblage at each stage in the iteration history during Leveling. |
| *dIdeal-Constant* | The ideal gas constant. |
| *dChemical-Potential* | A double real vector representing the chemical potential of each species. To be precise, this is defined as the difference between the standard molar Gibbs energy and the chemical potential defined by the element potentials (represented in dimensionless units and per formula mass). |
| *dElement-Potential* | A double real vector representing the element potentials. |
| *dExcess-GibbsParam* | A double real vector representing excess Gibbs energy of mixing parameters. |
| *dMoles-Element* | A double real vector representing the total number of moles of each element. |
| *dMoles-Phase* | A double real vector representing the moles of each phase. |
| *dMoles-Species* | A double real vector representing the number of moles each species in the system. |
| *dMolFraction* | A double real vector representing the mole fraction of each species in the system. |
| *dLevel* | A double real vector representing the adjustment applied to the element potentials. |
| *dAtom-Fraction-Species* | A double real matrix representing tha atom fraction of each element in each species. |
| *dTolerance* | A double real vector representing numerical tolerances (defined in InitThermo.f90). |
| *cElement-Name* | A character vector representing the name of each element in the system. |

| | |
|---|---|
| *cSpecies-Name* | A character vector representing the name of each species in short-form. |
| *cSolnPhase-Name* | A character vector representing the name of each solution phase. |
| *cSolnPhase-Type* | A character vector representing the type of each solution phase. |

Definition at line 72 of file ModuleThermo.f90.

### 4.4.2 Member Data Documentation

#### 4.4.2.1 character(3), dimension(:), allocatable ModuleThermo::cElementName

Definition at line 92 of file ModuleThermo.f90.

#### 4.4.2.2 character(25), dimension(:), allocatable ModuleThermo::cSolnPhaseName

Definition at line 95 of file ModuleThermo.f90.

#### 4.4.2.3 character(8), dimension(:), allocatable ModuleThermo::cSolnPhaseType

Definition at line 94 of file ModuleThermo.f90.

#### 4.4.2.4 character(25), dimension(:), allocatable ModuleThermo::cSpeciesName

Definition at line 93 of file ModuleThermo.f90.

#### 4.4.2.5 real(8), dimension(:,:), allocatable ModuleThermo::dAtomFractionSpecies

Definition at line 90 of file ModuleThermo.f90.

#### 4.4.2.6 real(8), dimension(:), allocatable ModuleThermo::dChemicalPotential

Definition at line 88 of file ModuleThermo.f90.

#### 4.4.2.7 real(8), dimension(:), allocatable ModuleThermo::dElementPotential

Definition at line 89 of file ModuleThermo.f90.

#### 4.4.2.8 real(8), dimension(:), allocatable ModuleThermo::dExcessGibbsParam

Definition at line 88 of file ModuleThermo.f90.

#### 4.4.2.9 real(8), dimension(:), allocatable ModuleThermo::dGibbsSolnPhase

Definition at line 87 of file ModuleThermo.f90.

#### 4.4.2.10 real(8) ModuleThermo::dIdealConstant

Definition at line 85 of file ModuleThermo.f90.

#### 4.4.2.11 real(8), dimension(:), allocatable ModuleThermo::dLevel

Definition at line 88 of file ModuleThermo.f90.

#### 4.4.2.12 real(8), dimension(:), allocatable ModuleThermo::dMolesElement

Definition at line 89 of file ModuleThermo.f90.

#### 4.4.2.13 real(8), dimension(:), allocatable ModuleThermo::dMolesPhase

Definition at line 89 of file ModuleThermo.f90.

#### 4.4.2.14 real(8), dimension(:), allocatable ModuleThermo::dMolesSpecies

Definition at line 87 of file ModuleThermo.f90.

#### 4.4.2.15 real(8), dimension(:), allocatable ModuleThermo::dMolFraction

Definition at line 89 of file ModuleThermo.f90.

#### 4.4.2.16 real(8) ModuleThermo::dNormalizeInput

Definition at line 85 of file ModuleThermo.f90.

#### 4.4.2.17 real(8) ModuleThermo::dNormalizeSum

Definition at line 85 of file ModuleThermo.f90.

#### 4.4.2.18 real(8), dimension(:), allocatable ModuleThermo::dStdGibbsEnergy

Definition at line 87 of file ModuleThermo.f90.

#### 4.4.2.19 real(8), dimension(itolnum) ModuleThermo::dTolerance

Definition at line 86 of file ModuleThermo.f90.

#### 4.4.2.20 integer, dimension(:), allocatable ModuleThermo::iAssemblage

Definition at line 82 of file ModuleThermo.f90.

#### 4.4.2.21 integer, dimension(:), allocatable ModuleThermo::iElementSystem

Definition at line 82 of file ModuleThermo.f90.

#### 4.4.2.22 integer, dimension(:), allocatable ModuleThermo::iParticlesPerMole

Definition at line 81 of file ModuleThermo.f90.

#### 4.4.2.23 integer, dimension(:), allocatable ModuleThermo::iPhase

Definition at line 81 of file ModuleThermo.f90.

#### 4.4.2.24 integer, dimension(:,:), allocatable ModuleThermo::iRegularParam

Definition at line 83 of file ModuleThermo.f90.

#### 4.4.2.25 integer, dimension(:,:), allocatable ModuleThermo::iSpeciesAtoms

Definition at line 83 of file ModuleThermo.f90.

#### 4.4.2.26 integer, dimension(:), allocatable ModuleThermo::iSpeciesPass

Definition at line 82 of file ModuleThermo.f90.

#### 4.4.2.27 integer, dimension(:), allocatable ModuleThermo::iSpeciesTotalAtoms

Definition at line 81 of file ModuleThermo.f90.

#### 4.4.2.28 integer, dimension(:,:), allocatable ModuleThermo::iterHistoryLevel

Definition at line 83 of file ModuleThermo.f90.

#### 4.4.2.29 integer, parameter ModuleThermo::iTolNum = 15

Definition at line 80 of file ModuleThermo.f90.

#### 4.4.2.30 integer ModuleThermo::nConPhases

Definition at line 79 of file ModuleThermo.f90.

#### 4.4.2.31 integer ModuleThermo::nDummySpecies

Definition at line 78 of file ModuleThermo.f90.

#### 4.4.2.32 integer ModuleThermo::nElements

Definition at line 78 of file ModuleThermo.f90.

#### 4.4.2.33 integer, parameter ModuleThermo::nElementsPT = 118

Definition at line 80 of file ModuleThermo.f90.

#### 4.4.2.34 integer ModuleThermo::nMaxParam

Definition at line 78 of file ModuleThermo.f90.

#### 4.4.2.35 integer ModuleThermo::nParam

Definition at line 78 of file ModuleThermo.f90.

#### 4.4.2.36 integer, dimension(:), allocatable ModuleThermo::nParamPhase

Definition at line 82 of file ModuleThermo.f90.

#### 4.4.2.37 integer ModuleThermo::nSolnPhases

Definition at line 79 of file ModuleThermo.f90.

#### 4.4.2.38 integer ModuleThermo::nSolnPhasesSys

Definition at line 79 of file ModuleThermo.f90.

#### 4.4.2.39 integer ModuleThermo::nSpecies

Definition at line 78 of file ModuleThermo.f90.

#### 4.4.2.40 integer, dimension(:), allocatable ModuleThermo::nSpeciesPhase

Definition at line 81 of file ModuleThermo.f90.

The documentation for this module was generated from the following file:

- src/shared/ModuleThermo.f90

## 4.5 ModuleThermoIO Module Reference

### Public Attributes

- integer iCounter
- real(8) dTemperature
- real(8) dPressure
- real(8), dimension(0:118) dElementMass
- character(15), dimension(3) cThermoInputUnits
- character(120) cThermoFileName
- integer INFOThermo
- integer nSolnPhasesOut
- integer nPureConPhaseOut
- integer nSpeciesOut
- real(8), dimension(:), allocatable dSolnPhaseMolesOut
- real(8), dimension(:), allocatable dPureConPhaseMolesOut
- real(8), dimension(:), allocatable dSpeciesMoleFractionOut
- character(25), dimension(:), allocatable cSolnPhaseNameOut
- character(25), dimension(:), allocatable cPureConPhaseNameOut
- character(25), dimension(:), allocatable cSpeciesNameOut
- character(25), dimension(:), allocatable cSpeciesPhaseOut

### 4.5.1 Detailed Description

**Parameters**

| | |
|---|---|
| *INFO-Thermo* | An integer scalar identifying whether the program exits successfully or if it encounters an error. Details are provided in ThermoDebug.f90. |
| *d-Temperature* | A double real scalar representing the absolute temperature. |
| *dPressure* | A double real scalar representing the absolute hydrostatic pressure. |
| *dElement-Mass* | A double real vector representing the mass of each chemical element for all the elements on the periodic table. |

| | |
|---|---|
| *cThermoFileName* | Name of a ChemSage data-file (e.g., 'UO2fuelthermo.dat'). NOTE: this has a maximum of 120 characters, which includes the path and the file extension. |
| *cThermoInputUnits* | A character vector containing the input units for Thermochimica. The following lists acceptable character strings:<br>• 1: Temperature units ['K', 'C', 'F', 'R'];<br>• 2: Absolute hydrostatic pressure units ['atm', 'psi', 'bar', 'Pa', 'kPa'];<br>• 3: Mass of each chemical element ['mass fraction', 'kilograms', 'grams', 'pounds', 'mole fraction', 'atom fraction', 'atoms', 'moles']. |

Definition at line 42 of file ModuleThermoIO.f90.

### 4.5.2 Member Data Documentation

#### 4.5.2.1 character(25), dimension(:), allocatable ModuleThermoIO::cPureConPhaseNameOut

Definition at line 58 of file ModuleThermoIO.f90.

#### 4.5.2.2 character(25), dimension(:), allocatable ModuleThermoIO::cSolnPhaseNameOut

Definition at line 58 of file ModuleThermoIO.f90.

#### 4.5.2.3 character(25), dimension(:), allocatable ModuleThermoIO::cSpeciesNameOut

Definition at line 58 of file ModuleThermoIO.f90.

#### 4.5.2.4 character(25), dimension(:), allocatable ModuleThermoIO::cSpeciesPhaseOut

Definition at line 58 of file ModuleThermoIO.f90.

#### 4.5.2.5 character(120) ModuleThermoIO::cThermoFileName

Definition at line 53 of file ModuleThermoIO.f90.

#### 4.5.2.6 character(15), dimension(3) ModuleThermoIO::cThermoInputUnits

Definition at line 52 of file ModuleThermoIO.f90.

#### 4.5.2.7 real(8), dimension(0:118) ModuleThermoIO::dElementMass

Definition at line 51 of file ModuleThermoIO.f90.

#### 4.5.2.8 real(8) ModuleThermoIO::dPressure

Definition at line 50 of file ModuleThermoIO.f90.

#### 4.5.2.9 real(8), dimension(:), allocatable ModuleThermoIO::dPureConPhaseMolesOut

Definition at line 57 of file ModuleThermoIO.f90.

#### 4.5.2.10 real(8), dimension(:), allocatable ModuleThermoIO::dSolnPhaseMolesOut

Definition at line 57 of file ModuleThermoIO.f90.

#### 4.5.2.11 real(8), dimension(:), allocatable ModuleThermoIO::dSpeciesMoleFractionOut

Definition at line 57 of file ModuleThermoIO.f90.

#### 4.5.2.12 real(8) ModuleThermoIO::dTemperature

Definition at line 50 of file ModuleThermoIO.f90.

#### 4.5.2.13 integer ModuleThermoIO::iCounter

Definition at line 49 of file ModuleThermoIO.f90.

#### 4.5.2.14 integer ModuleThermoIO::INFOThermo

Definition at line 56 of file ModuleThermoIO.f90.

#### 4.5.2.15 integer ModuleThermoIO::nPureConPhaseOut

Definition at line 56 of file ModuleThermoIO.f90.

#### 4.5.2.16 integer ModuleThermoIO::nSolnPhasesOut

Definition at line 56 of file ModuleThermoIO.f90.

#### 4.5.2.17 integer ModuleThermoIO::nSpeciesOut

Definition at line 56 of file ModuleThermoIO.f90.

The documentation for this module was generated from the following file:

- src/shared/ModuleThermoIO.f90

# Chapter 5

# File Documentation

## 5.1 src/shared/ArrowSolver.f90 File Reference

Solve a system of simultaneous linear equations with a symmetric arrow matrix.

### Functions/Subroutines

- subroutine ArrowSolver (n, INFO, dDiagonal, dSymmetricVal, dFunction)

### 5.1.1 Detailed Description

Solve a system of simultaneous linear equations with a symmetric arrow matrix.

**Author**

M.H.A. Piro

**Date**

September 19, 2012

Definition in file ArrowSolver.f90.

### 5.1.2 Function/Subroutine Documentation

#### 5.1.2.1 subroutine ArrowSolver ( integer, intent(in) *n,* integer *INFO,* real(8), dimension(n-1), intent(in) *dDiagonal,* real(8), intent(in) *dSymmetricVal,* real(8), dimension(n), intent(inout) *dFunction* )

The purpose of this subroutine is to solve a real system of linear equations Ax = b where A is an N-by-N symmetric arrow matrix with constant values on the arrow head and x and

b are N vectors. This subroutine exploits the structure of linear equations of this specific problem. To be precise, the A matrix (if it were constructed) would be 0 everywhere except for the diagonal [except A(n,n) = 0], the bottom row and the extreme right column. The extreme right column is a transpose of the bottom row and all coefficients of this row are a constant value. Needless to say that this solver will not work for any other type of system of linear equations.

This subroutine effectively performs Gaussian elimination on only the bottom row. - Computational expense is further reduced for this specific problem when performing back subsituation by exploiting the sparsity of the A matrix. This subroutine returns an error via the INFO variable if an error has been detected. The possible values for INFO are summarized below:

INFO 0 Successful exit. -1 dSymmetricVal is zero or a NAN. -2 dDiagonal contains a zero or a NAN. i The row corresponding to i (where i $>$ 0) in dFunction contains a NAN.

**Parameters**

| | | |
|---|---|---|
| `in` | *n* | An integer scalar representing the number of linear equations. |
| `out` | *INFO* | An integer scalar indicating a successful exit or an error. |
| `in` | *dDiagonal* | A double real vector (dimension n-1) representing the diagonal of the A matrix. |
| `in` | *dSymmetricVal* | A double real scalar representing the constant value lying on the arrow head. |
| `in,out` | *dFunction* | A double real vector (dimension n) representing the functional vector on input and on output it represents the solution vector. |

Definition at line 73 of file ArrowSolver.f90.

## 5.2 src/shared/Broyden.f90 File Reference

Perform a unit iteration using Broyden's method.

### Functions/Subroutines

- subroutine Broyden (nVar, y, s, f, dInvBroyden)

### 5.2.1 Detailed Description

Perform a unit iteration using Broyden's method.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

Definition in file Broyden.f90.

### 5.2.2 Function/Subroutine Documentation

#### 5.2.2.1 subroutine Broyden ( integer *nVar,* real(8), dimension(nvar) *y,* real(8), dimension(nvar) *s,* real(8), dimension(nvar) *f,* real(8), dimension(nvar,nvar) *dInvBroyden* )

The purpose of this subroutine is to perform a unit iteration in solving a system of non-linear equations using the "good" Broyden method. This subroutine does not perform an iteration process, but is intended to be used within a main iteration cycle. An estimate of the inverse of the Broyden matrix is required as input and an improved estimate is provided as output in addition to a new direction vector s.

One of the advantages of using Broyden's method in comparison to the more popular BFGS method is that Broyden's method permits nonsymmetric updates to the inverse Broyden matrix, whereas the BFGS method maintains symmetry. The true Jacobian for some numerical problems may not always be symmetric.

**Parameters**

| | | |
|---|---|---|
| in | *nVar* | The number of unknown variables (and of course the number of non-linear equations) |
| in,out | *s* | The change of the unknown variable vector (nVar) |
| in,out | *y* | The change of the function vector (nVar) |
| in | *f* | The function vector (nVar) |
| in,out | *dInvBroyden* | The inverse of the Broyden matrix (nVar x nVar) |

Definition at line 45 of file Broyden.f90.

## 5.3 src/shared/CheckConvergence.f90 File Reference

Check convergence in the non-linear solver.

### Functions/Subroutines

- subroutine CheckConvergence

### 5.3.1 Detailed Description

Check convergence in the non-linear solver.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

GEMSolver.f90

Definition in file CheckConvergence.f90.

### 5.3.2 Function/Subroutine Documentation

#### 5.3.2.1 subroutine CheckConvergence ( )

The purpose of this subroutine is to check if convergence has been achieved. The conditions for thermodynamic equilibrium are (refer to above references for more details):

1. None of the phases in the estimated phase assemblage are "dummy species", which were introduced by the ChemSage data-file,
2. The standard Gibbs energy of a pure condensed phase is not below the Gibbs Plane. If so, this phase should be added to the assemblage.
3. The number of moles of all species and phases are positive and non-zero.
4. The sum of mole fractions of all species in a solution phase that is not in currently assumed to be stable is less than unity.
5. The relative errors of the mass balance equations are within tolerance.

Each criterion listed above is sequentially tested and the system is considered converged when all are satisfied. Control is returned to the PGESolver subroutine when any of the criterions has not been satisfied. Note that the order of testing is done in a fashion that progressively increases computational expense. For example, testing the mass balance constraints is the most computationally expensive task, which is why it is performed last.

Note that the Gibbs Phase Rule is necessarily satisfied because iAssemblage is dimensioned by nElements. Therefore, it is impossible for nPhases $>$ nElements and the Gibbs Phase Rule is implicitly satisfied at all times.

Definition at line 87 of file CheckConvergence.f90.

## 5.4 src/shared/CheckMiscibilityGap.f90 File Reference

Check for a miscibility gap.

## Functions/Subroutines

- subroutine CheckMiscibilityGap (iSolnPhaseIndex, lAddPhase)

### 5.4.1 Detailed Description

Check for a miscibility gap.

**Author**

M.H.A. Piro

**Date**

Aug. 30, 2012

**See also**

Subminimization.f90
CheckSolnPhaseAdd.f90
CheckConvergence.f90

Definition in file CheckMiscibilityGap.f90.

### 5.4.2 Function/Subroutine Documentation

#### 5.4.2.1 subroutine CheckMiscibilityGap ( integer *iSolnPhaseIndex*, logical *lAddPhase* )

The purpose of this subroutine is to check whether a particular non-ideal solution phase containing a miscibility gap should be added to the system. The subminimization routine determines whether the driving force associated with a local minima is positive/negative. There may be multiple local minima for a particular solution phase and different local minima may be discovered depending on the initial estimates of the mole fractions of this phase. The approach taken here performs subminimization multiple times where the initial estimates for each case begin at the extremums of the domain space. Specifically, the mole fractions of all constituents are set to an arbitrarily small value (e.g., 1D-3) except for one constituent where the sum of the mole fractions equals unity. Each constituent in this phase is systematically initialized as being dominant.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnPhaseIndex* | An integer scalar representing the absolute index of the solution phase that is being considered. |
| out | *lAddPhase* | A logical scalar indicating whether the phase should be added (i.e., TRUE) |

Definition at line 52 of file CheckMiscibilityGap.f90.

## 5.5 src/shared/CheckPhaseAssemblage/AddPureConPhase.f90 - File Reference

Add a pure condensed phase to the system.

### Functions/Subroutines

- subroutine AddPureConPhase (iPhaseChange, lSwapLater, lPhasePass)

### 5.5.1 Detailed Description

Add a pure condensed phase to the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPureConPhaseAdd.f90
CheckIterHistory.f90
CheckPhaseChange.f90

Definition in file AddPureConPhase.f90.

### 5.5.2 Function/Subroutine Documentation

#### 5.5.2.1 subroutine AddPureConPhase ( integer *iPhaseChange,* logical *lSwapLater,* logical *lPhasePass* )

The purpose of this subroutine is to add a pure condensed phase to the estimated phase assemblage. The new phase assemblage is tested to ensure that it is appropriate for furhter consideration.

**Parameters**

| | | |
|---|---|---|
| in | *iPhase-Change* | An integer scalar representing the pure condensed phase that is to be added to the system. |
| out | *lSwapLater* | A logical scalar indicating whether the phase should be swapped with another phase later on in another subroutine. |
| out | *lPhasePass* | A local scalar indicating whether the new estimated phase assemblage passed (.TRUE.) or failed (.FALSE.). |

Definition at line 51 of file AddPureConPhase.f90.

## 5.6 src/shared/CheckPhaseAssemblage/AddSolnPhase.f90 File Reference

Add a solution phase to the system.

### Functions/Subroutines

- subroutine AddSolnPhase (iPhaseChange, lSwapLater, lPhasePass)

### 5.6.1 Detailed Description

Add a solution phase to the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckSolnPhaseAdd.f90
CompMolSolnPhase.f90

Definition in file AddSolnPhase.f90.

### 5.6.2 Function/Subroutine Documentation

#### 5.6.2.1 subroutine AddSolnPhase ( integer *iPhaseChange,* logical *lSwapLater,* logical *lPhasePass* )

The purpose of this subroutine is to add a solution phase to the estimated phase assemblage. The new phase assemblage is tested to ensure that it is appropriate and a logical variable lPhasePass is returned.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseChange* | An integer scalar representing the absolute index of a solution phase to be added to the system. |
| out | *lSwapLater* | A logical variable indicating whether a phase should be swapped for another phase when this particular phase cannot be added directly to the system. |
| out | *lPhasePass* | A logical variable indicating whether the new estimated phase assemblage passed (.TRUE.) or failed (.FALSE.). |

Definition at line 62 of file AddSolnPhase.f90.

## 5.7 src/shared/CheckPhaseAssemblage/CheckAddMiscriblePhase.f90 File Reference

Check the.

### Functions/Subroutines

- subroutine CheckAddMiscriblePhaseIndex (iPhaseAddAbs)

### 5.7.1 Detailed Description

Check the.

**Author**

M.H.A. Piro

**Date**

Oct. 21, 2012

**See also**

AddSolnPhase.f90
SwapSolnForPureConPhase.f90

Definition in file CheckAddMiscriblePhase.f90.

### 5.7.2 Function/Subroutine Documentation

#### 5.7.2.1 subroutine CheckAddMiscriblePhaseIndex ( integer *iPhaseAddAbs* )

The purpose of this subroutine is to check if the phase that is being added is miscible and if one of the other corresponding phases is currently not predicted to be stable. If so, this subroutine checks if the absolute index of the phase that is to be added is greater than the other phase, in which case it swaps the two phases.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseAddAbs* | Absolute index of the miscible solution phase that is to be added to the system. |

Definition at line 41 of file CheckAddMiscriblePhase.f90.

## 5.8 src/shared/CheckPhaseAssemblage/CheckIterHistory.f90 File - Reference

Check the iteration history to see if a particular phase assemblage has previously been considered.

### Functions/Subroutines

- subroutine CheckIterHistory (iAssemblageTest, iterBack, lSwapLater)

### 5.8.1 Detailed Description

Check the iteration history to see if a particular phase assemblage has previously been considered.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

AddPureConPhase.f90

Definition in file CheckIterHistory.f90.

### 5.8.2 Function/Subroutine Documentation

#### 5.8.2.1 subroutine CheckIterHistory ( integer, dimension(nelements) *iAssemblageTest,* integer *iterBack,* logical *lSwapLater* )

The purpose of this subroutine is to check whether a particular phase assemblage has been considered in the iteration history. An additional test is performed that checks the order that a phase is added to, removed from, or swapped from the system. For example, a system may be comprised of phases A, B, C and D at iteration w, phase D is removed at iteration x, and phase B is prematurely removed at iteration y. At iteration z, one determines that phase B should be added to the system, which would result in an identical phase assemblage at iteration x. This test recognizes that the phase assemblages considered at iteration x and z are identical, but the changes to the system that results in these assemblages are different.

The logical variable lSwapLater is returned indicating whether this phase assemblage can be considered or not.

**Parameters**

| | | |
|---|---|---|
| in | *iAssemblageTest* | An integer vector representing the phase assemblage to be tested. |
| in | *iterBack* | An integer scalar indicating how many iteratoins to go back in history. |
| out | *lSwapLater* | A logical variable indicating whether a phase should be swapped later on. |

Definition at line 67 of file CheckIterHistory.f90.

## 5.9 src/shared/CheckPhaseAssemblage/CheckPhaseAssemblage.f90 File Reference

Check whether the estimated phase assemblage needs to be modified.

### Functions/Subroutines

- subroutine CheckPhaseAssemblage

### 5.9.1 Detailed Description

Check whether the estimated phase assemblage needs to be modified.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

GEMSolver.f90
CheckPureConPhaseRem.f90
CheckPureConPhaseAdd.f90
CheckSolnPhaseRem.f90
CheckSolnPhaseAdd.f90
CheckStagnation.f90
RevertSystem.f90

Definition in file CheckPhaseAssemblage.f90.

## 5.9.2 Function/Subroutine Documentation

### 5.9.2.1 subroutine CheckPhaseAssemblage ( )

The purpose of this subroutine is to check whether a phase should be added to, removed from, or swap another phase currently in the estimated phase assemblage and to perform any necessary action. The conditions for adding or removing a phase from the assemblage follow:

1. Remove a pure condensed phase when the number of moles of that phase is negative,

2. Remove a solution phase when the number of moles of that phase is less than a specified tolernace,

3. Add a pure condensed phase when the driving force is negative,

4. Add a solution phase when the sum of all mole fractions within the phase is greater than unity. An equivalent statement is that the driving force of this phase is negative.

To prevent contradicting the Gibbs Phase Rule (i.e., the maximum number of phases is equal to the number of elements when temperature and pressure are constant), a phase replaces another phase when the number of phases already in the system equals the number of elements. The phase assemblage is not checked every iteration to give the numerical solution a chance to converge.

The main subroutines used by this solver are summarized below:

| File name | Description |
|---|---|
| RevertSystem.f90 | Revert the system to a particular phase assemblage corresponding to a partiuclar iteration. |
| CheckPureConPhaseRem.f90 | Check if a pure condensed phase should be removed. |
| CheckSolnPhaseRem.f90 | Check if a solution phase should be removed. |
| CheckStagnation.f90 | Check if the system is stagnant. Specifically, this subroutine counts the # of solution phases that have molar quantities that have changed by more than a specified amount. |
| CheckPureConPhaseAdd.f90 | Check if a pure condensed phase should be added to the system. |
| CheckSolnPhaseAdd.f90 | Check if a solution phase should be added to the system. |

Definition at line 148 of file CheckPhaseAssemblage.f90.

## 5.10 src/shared/CheckPhaseAssemblage/CheckPhaseChange.f90 File Reference

Check whether a particular phase change is appropriate for further consideration.

### Functions/Subroutines

- subroutine CheckPhaseChange (lPhasePass, INFO)

### 5.10.1 Detailed Description

Check whether a particular phase change is appropriate for further consideration.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

GEMNewton.f90

Definition in file CheckPhaseChange.f90.

### 5.10.2 Function/Subroutine Documentation

#### 5.10.2.1 subroutine CheckPhaseChange ( logical *lPhasePass,* integer *INFO* )

The purpose of this subroutine is to check whether a particular phase assemblage is a valid candidate. It is possible for a particular combination of phases to yield non-real values when evaluating the Jacobian. For example, suppose there is 1 mol of uranium in the system and the only uranium containing phase is removed from the assemblage. Clearly, there must be at least one phase containing uranium for the system to be defined.

**Parameters**

| | | |
|---|---|---|
| out | *lPhasePass* | A logical variable indicating whether the candidate phase assemblage is appropriate (i.e., .TRUE.) or not (i.e., .FALSE.). |
| out | *INFO* | An integer scalar used to identify a successful exit or an error by the GEMNewton.f90 subroutine. |

Definition at line 42 of file CheckPhaseChange.f90.

## 5.11 src/shared/CheckPhaseAssemblage/CheckPureConPhaseAdd.f90 File Reference

Check whether a pure condensed phase should be added to the system.

### Functions/Subroutines

- subroutine CheckPureConPhaseAdd (iMaxDrivingForce, dMaxDrivingForce)
- subroutine CheckPureConPhaseSwap (iPhaseChange, lSwapLater, lPhasePass)

### 5.11.1 Detailed Description

Check whether a pure condensed phase should be added to the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPhaseAssemblage.f90

Definition in file CheckPureConPhaseAdd.f90.

### 5.11.2 Function/Subroutine Documentation

#### 5.11.2.1 subroutine CheckPureConPhaseAdd ( integer *iMaxDrivingForce*, real(8) *dMaxDrivingForce* )

The purpose of this subroutine is to check whether a pure condensed phase should be added to the system. The condition for a pure condensed phase to be added is when the driving force is less than a specified tolerance (e.g., $\sim$ -1D-5). The driving force is defined as the difference between the standard molar Gibbs energy of this phase and the corresponding value computed by the element potentials. Refer to the following literature for a more thorough discussion:

H.L. Lukas, S.G. Fries and B. Sundman, "Computational Thermodynamics: The Calphad Method," Cambridge University Press, New York, 2007.

A pure condensed phase can be added to the system when the current number of phases is less than the number of elements in the system; however, it must replace an existing phase in the event that the number of phases is equal to the number of elements to prevent contradicting Gibbs' Phase Rule. If a particular phase cannot be added directly to the system, then it will try to replace an existing phase.

Definition at line 67 of file CheckPureConPhaseAdd.f90.

#### 5.11.2.2 subroutine CheckPureConPhaseSwap ( integer *iPhaseChange,* logical *lSwapLater,* logical *lPhasePass* )

Definition at line 146 of file CheckPureConPhaseAdd.f90.

## 5.12 src/shared/CheckPhaseAssemblage/CheckPureConPhase-Rem.f90 File Reference

Check whether a pure condensed phase should be removed.

### Functions/Subroutines

- subroutine CheckPureConPhaseRem

### 5.12.1 Detailed Description

Check whether a pure condensed phase should be removed.

**Author**

M.H.A. Piro

**Date**

May 23, 2012

**See also**

CheckPhaseAssemblage.f90
RemPureConPhase.f90
RemPureConAddSolnPhase.f90
SwapPureConPhase.f90
ShuffleAssemblage.f90

Definition in file CheckPureConPhaseRem.f90.

### 5.12.2 Function/Subroutine Documentation

#### 5.12.2.1 subroutine CheckPureConPhaseRem ( )

The purpose of this subroutine is to check whether a pure condensed phase should be removed from the system. The condition for a pure condensed phase to be removed from the system is when the number of moles of that phase is below a specified tolerance and the change to the number of moles of the phase is below an arbitrarily small value.

Definition at line 58 of file CheckPureConPhaseRem.f90.

## 5.13 src/shared/CheckPhaseAssemblage/CheckSolnPhaseAdd.f90 File Reference

Check if a solution phase should be added to the system.

### Functions/Subroutines

- subroutine CheckSolnPhaseAdd
- subroutine CheckSolnPhaseSwap (i, lPhasePass)

### 5.13.1 Detailed Description

Check if a solution phase should be added to the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPhaseAssemblage.f90
SortPick.f90
AddSolnPhase.f90
ShuffleAssemblage.f90
SwapSolnForPureConPhase.f90
SwapSolnPhase.f90
CompMolFraction.f90

Definition in file CheckSolnPhaseAdd.f90.

### 5.13.2 Function/Subroutine Documentation

#### 5.13.2.1 subroutine CheckSolnPhaseAdd ( )

The purpose of this subroutine is to check whether a solution phase should be added to the current estimated assemblage of stable phases in the system. The condition for a solution phase to be added to the system is when the sum of hypothetical mole fractions of all constituents within a solution phase exceeds unity (within tolerance). A solution phase can be added to the system when the total number of phases is less than the number of elements in the system; however, it must replace an existing phase when the number of phases equals the number of elements (to prevent contradicting Gibbs' Phase Rule).

Definition at line 79 of file CheckSolnPhaseAdd.f90.

#### 5.13.2.2 subroutine CheckSolnPhaseSwap ( integer *i,* logical *lPhasePass* )

Definition at line 190 of file CheckSolnPhaseAdd.f90.

## 5.14 src/shared/CheckPhaseAssemblage/CheckSolnPhaseRem.f90 File Reference

Check whether a solution phase needs to be removed from the system.

### Functions/Subroutines

- subroutine CheckSolnPhaseRem

### 5.14.1 Detailed Description

Check whether a solution phase needs to be removed from the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPhaseAssemblage.f90
RemSolnPhase.f90
RemSolnAddPureConPhase.f90
CompDrivingForce.f90

Definition in file CheckSolnPhaseRem.f90.

### 5.14.2 Function/Subroutine Documentation

#### 5.14.2.1 subroutine CheckSolnPhaseRem ( )

The purpose of this subroutine is to check whether a solution phase should be removed from the system. A solution phase should be removed from the sytem if the number of moles of that phase is less than a certain tolernace (specified in InitThermo.f90). It may be possible that this phase may not be removed directly from the system because it would result in a singularity. Provisions are in place to check if a pure condensed phase should take the place of this solution phase. If this does not work, then a check is performed to see if a different solution phase should take the place of this phase. If this does not work and the number of moles of this solution phase is sufficiently small, then pure condensed phases are systematically removed from the system in a last attempt to converge.

Definition at line 71 of file CheckSolnPhaseRem.f90.

## 5.15 src/shared/CheckPhaseAssemblage/CheckStagnation.f90 File Reference

Check if the system is stagnant.

### Functions/Subroutines

- subroutine CheckStagnation (dMolesPhaseChange, dMaxChange, nPhasesCheck)

### 5.15.1 Detailed Description

Check if the system is stagnant.

**Author**

M.H.A. Piro

**Date**

July 4, 2012

**See also**

CheckPhaseAssemblage.f90

Definition in file CheckStagnation.f90.

### 5.15.2 Function/Subroutine Documentation

#### 5.15.2.1 subroutine CheckStagnation ( real(8) *dMolesPhaseChange,* real(8) *dMaxChange,* integer *nPhasesCheck* )

The purpose of this subroutine is to check whether the system has become stagnant by determining the number of solution phases that have molar quantities that changed by a specified value. For example, this subroutine will return the number of solution phases that have changed by more than 5%.

**Parameters**

| | | |
|---|---|---|
| in | *dMoles-Phase-Change* | A double real scalar representing the relative change of the number of moles of a solution phase (e.g., = 0.05). |
| out | *nPhases-Check* | An integer scalar representing the number of solution phases that have molar quantities that have changed more than the specified amount. |
| out | *dMax-Change* | A double real scalar representing the maximum relative change of dMolesPhase. |

Definition at line 46 of file CheckStagnation.f90.

## 5.16 src/shared/CheckPhaseAssemblage/CompDrivingForce.f90 File Reference

Compute the driving force of all pure condensed phases.

### Functions/Subroutines

- subroutine CompDrivingForce (iMaxDrivingForce, dMaxDrivingForce)

### 5.16.1 Detailed Description

Compute the driving force of all pure condensed phases.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

Definition in file CompDrivingForce.f90.

### 5.16.2 Function/Subroutine Documentation

#### 5.16.2.1 subroutine CompDrivingForce ( integer *iMaxDrivingForce*, real(8) *dMaxDrivingForce* )

The purpose of this subroutine is to compute the driving force of all pure condensed phases in the database. The driving force is defined as the difference between the standard molar Gibbs energy of a pure condensed phase and the corresponding value computed from the element potentials. This value is used to determine whether a pure condensed phase should be added to the system. For a more thorough explanation of the chemical significance of the driving force, refer to the following literature:

H.L. Lukas, S.G. Fries, B. Sundman, Computational Thermodynamics - The Calphad Method, Cambridge University Press, New York, 2007.

**Parameters**

| | | |
|---|---|---|
| out | *iMaxDrivingForce* | An integer scalar representing the index of the pure condensed phase with the maximum driving force. A value of zero is returned by default. |
| out | *dMaxDrivingForce* | A double real scalar representing the maximum driving force of all pure condensed phases. A value of zero is returned by default. |

Definition at line 41 of file CompDrivingForce.f90.

## 5.17 src/shared/CheckPhaseAssemblage/RemPureConAddSolnPhase.f90 File Reference

Simultaneously remove a pure condensed phase and add a solution phase.

### Functions/Subroutines

- subroutine RemPureConAddSolnPhase (lPhasePass)

  *param[out] lPhasePass A logical variable indicating whether the new estimated phase assemblage passed (.TRUE.) or failed (.FALSE.).*

### 5.17.1 Detailed Description

Simultaneously remove a pure condensed phase and add a solution phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPureConPhaseRem.f90
AddSolnPhase.f90

Definition in file RemPureConAddSolnPhase.f90.

### 5.17.2 Function/Subroutine Documentation

#### 5.17.2.1 subroutine RemPureConAddSolnPhase ( logical *lPhasePass* )

param[out] lPhasePass A logical variable indicating whether the new estimated phase assemblage passed (.TRUE.) or failed (.FALSE.).

The purpose of this subroutine is to remove a particular pure condensed phase and add a solution phase.

Definition at line 41 of file RemPureConAddSolnPhase.f90.

## 5.18 src/shared/CheckPhaseAssemblage/RemPureConPhase.f90 - File Reference

Remove a pure condensed phase from the system.

### Functions/Subroutines

- subroutine RemPureConPhase (iPhaseChange, lSwapLater, lPhasePass)

### 5.18.1 Detailed Description

Remove a pure condensed phase from the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPureConPhaseRem.f90
CheckPhaseChange.f90

Definition in file RemPureConPhase.f90.

### 5.18.2 Function/Subroutine Documentation

#### 5.18.2.1 subroutine RemPureConPhase ( integer *iPhaseChange,* logical *lSwapLater,* logical *lPhasePass* )

The purpose of this subroutine is to remove a pure condensed phase from the estimated phase assemblage.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseChange* | An integer scalar representing the index of the pure condensed phase to be removed from the system. |
| out | *lSwapLater* | A logical scalar indicating whether this phase should be swapped later. |
| out | *lPhasePass* | A logical scalar indicating whether the new phase assemblage has passed. |

Definition at line 45 of file RemPureConPhase.f90.

## 5.19 src/shared/CheckPhaseAssemblage/RemSolnAddPureConPhase.f90 File Reference

Simultaneously remove a particular solution phase and add a particular pure condensed phase.

### Functions/Subroutines

- subroutine RemSolnAddPureConPhase (iPhaseAdd, iPhaseRem, lPhasePass)

### 5.19.1 Detailed Description

Simultaneously remove a particular solution phase and add a particular pure condensed phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckSolnPhaseRem.f90
RemSolnPhase.f90
CheckSysOnlyPureConPhases.f90
CheckConvergence.f90
CheckPhaseChange.f90

Definition in file RemSolnAddPureConPhase.f90.

### 5.19.2 Function/Subroutine Documentation

#### 5.19.2.1 subroutine RemSolnAddPureConPhase ( integer *iPhaseAdd*, integer *iPhaseRem*, logical *lPhasePass* )

The purpose of this subroutine is to remove a particular solution phase and add a particular pure condensed phase.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseAdd* | An integer scalar representing the index of pure condensed phase to be added to the system. |
| in | *iPhaseRem* | An integer scalar representing the ndex of solution phase to be removed from the system. |
| out | *lPhasePass* | A logical variable indicating whether the new estimated phase assemblage passed (.TRUE.) or failed (.FALSE.). |

Definition at line 51 of file RemSolnAddPureConPhase.f90.

## 5.20 src/shared/CheckPhaseAssemblage/RemSolnPhase.f90 File - Reference

Remove a solution phase from the system.

## Functions/Subroutines

- subroutine RemSolnPhase (iPhaseChange, lPhasePass)
- subroutine CheckRemMiscible Phase (iPhaseRemoveAbs)

### 5.20.1 Detailed Description

Remove a solution phase from the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckSolnPhaseRem.f90
CheckPhaseChange.f90

Definition in file RemSolnPhase.f90.

### 5.20.2 Function/Subroutine Documentation

#### 5.20.2.1 subroutine CheckRemMiscible Phase ( integer *iPhaseRemoveAbs* )

Definition at line 186 of file RemSolnPhase.f90.

#### 5.20.2.2 subroutine RemSolnPhase ( integer *iPhaseChange*, logical *lPhasePass* )

The purpose of this subroutine is to remove a solution phase from the estimated phase assemblage. The CheckPhaseChange.f90 subroutine is called to ensure that the new phase assemblage is appropriate for further consideration.

**Parameters**

| | | |
|---|---|---|
| in | *iPhase-Change* | An integer scalar representing the solution phase index corresponding to the iAssemblage and dMolesPhase vectors. |
| out | *lPhasePass* | A logical variable indicating whether the new phase assemblage has passed. |

Definition at line 54 of file RemSolnPhase.f90.

## 5.21 src/shared/CheckPhaseAssemblage/RevertSystem.f90 File - Reference

Revert the system to a previously considered phase assemblage.

### Functions/Subroutines

- subroutine RevertSystem (iterSpecific)

### 5.21.1 Detailed Description

Revert the system to a previously considered phase assemblage.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

Definition in file RevertSystem.f90.

### 5.21.2 Function/Subroutine Documentation

#### 5.21.2.1 subroutine RevertSystem ( integer *iterSpecific* )

The purpose of this subroutine is to revert the system to the last successfully tested phase assemblage.

**Parameters**

| | | |
|---|---|---|
| in | *iterSpecific* | An integer scalar representing a specific global iteration to revert to. |

Definition at line 48 of file RevertSystem.f90.

## 5.22 src/shared/CheckPhaseAssemblage/ShuffleAssemblage.f90 - File Reference

Shuffle the phase assemblage in the order that is most favorable for phase exchange.

## Functions/Subroutines

- subroutine ShuffleAssemblage (iNewPhase, iPhaseTypeOut)

### 5.22.1 Detailed Description

Shuffle the phase assemblage in the order that is most favorable for phase exchange.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

SortPick.f90
CheckPureConPhaseAdd.f90
CheckPureConPhaseRem.f90
CheckSolnPhaseAdd.f90
GetNewAssemblage.f90

Definition in file ShuffleAssemblage.f90.

### 5.22.2 Function/Subroutine Documentation

#### 5.22.2.1 subroutine ShuffleAssemblage ( integer *iNewPhase,* integer *iPhaseTypeOut* )

The purpose of this subroutine is to shuffle the current estimated phase assemblage such that the order of phases in iAssemblage are in oder of atomic similarity to a new phase that is to be added to the system. The Gibbs Phase Rule dictates that the maximum number of phases that can coexist cannot exceed the number of system components (taken here as chemical elements). Therefore, if the number of phases currently expected to be stable is equal to the number of elements, another phase must be withdrawn from the system to accomodate this new phase.

The principle of this technique is to quantify the atomic similarity between the new phase and all other phases in the system. The phase with the most similar atomic constituency is the most likely best candidate. The principle of this technique is to compute the Euclidean norm vector in nElements dimensional space, where one point is the stoichiometry of the new phase to be introduced and each other point represents the other phases in the current phase assemblage. The iAssemblage vector is reorganized in descending order of the Euclidean Norm.

The principle of this technique are discussed in greater detail in the following literature:

- M.H.A. Piro and S. Simunovic, "Performance Enhancing Algorithms for Computing Thermodynamic Equilibria," CALPHAD, 39 (2012) 104-110.

**Parameters**

| | | |
|---|---|---|
| in | *iNewPhase* | Index of the new phase to be introduced. If it is positive, it is a pure condensed phase. If it is negative, it is a solution phase. |
| in | *iPhaseType-Out* | An integer idenitifying the type of phase with the smallest Euclidean Norm. This can be used to determine whether a pure condensed phase should be swapped first or a solution phase. |

iPhaseTypeOut = 0: pure condensed phase; iPhaseTypeOut = 1: solution phase.

Definition at line 98 of file ShuffleAssemblage.f90.

## 5.23 src/shared/CheckPhaseAssemblage/Subminimization.f90 File Reference

Determine whether a particular non-ideal solution phase should be added to the system by performing a subminimization routine.

### Functions/Subroutines

- subroutine Subminimization (iSolnPhaseIndex, lPhasePass)
- subroutine SubMinInit (iSolnPhaseIndex, iterSubMax)
- subroutine SubMinChemicalPotential (iSolnPhaseIndex)
- subroutine SubMinDrivingForce
- subroutine SubMinNewton (iSolnPhaseIndex)
- subroutine SubMinLineSearch (iSolnPhaseIndex)
- subroutine SubMinCheckDuplicate (lDuplicate)

### 5.23.1 Detailed Description

Determine whether a particular non-ideal solution phase should be added to the system by performing a subminimization routine.

**Author**

M.H.A. Piro

**Date**

Aug. 21, 2012

**See also**

CheckSolnPhaseAdd.f90

Definition in file Subminimization.f90.

## 5.23.2 Function/Subroutine Documentation

### 5.23.2.1 subroutine SubMinCheckDuplicate ( logical *lDuplicate* )

Definition at line 592 of file Subminimization.f90.

### 5.23.2.2 subroutine SubMinChemicalPotential ( integer *iSolnPhaseIndex* )

Definition at line 301 of file Subminimization.f90.

### 5.23.2.3 subroutine SubMinDrivingForce ( )

Definition at line 352 of file Subminimization.f90.

### 5.23.2.4 subroutine Subminimization ( integer *iSolnPhaseIndex,* logical *lPhasePass* )

The purpose of this subroutine is to determine whether a particular non-ideal solution phase should be added to the system by performing a subminimization. The criteria for adding any type of phase (regardless of whether it is a pure stoichiometric phase, ideal or non-ideal solution phase) is based on whether the driving force is positive (it should not be added) or negative (it should be added to the system). The driving force is defined as the difference between the molar Gibbs energy of a particular phase and the corresponding value defined by the element potentials. A graphical interpretation of the driving force is the difference between the molar Gibbs energy of a particular phase (represented by a line for a stoichiometric phase or a point on a curve for a solution phase) and the corresponding value on the Gibbs hyperplane.

For further information regarding the term "driving force", refer to:

H.L. Lukas, S.G. Fries and B. Sundman, "Computational Thermodynamics: The Calphad Method," Cambridge University Press, New York (2007).

The premise of the subminimization method is to determine a unique combination of mole fractions for a particular non-ideal solution phase that minimizes the driving force of that phase. As an additional constraint, the sum of the mole fractions of this phase must equal unity. In the subminimization approach, a new Lagrangian function is defined as:

$L_\lambda = \sum_{i=1}^{N_\lambda} x^n_{i(\lambda)} \left( \mu^n_{i(\lambda)} - \sum_{j=1}^{E} a_{i,j} \Gamma^m_j \right) - \pi^n_\lambda \left( \sum_{i=1}^{N_\lambda} x^n_{i(\lambda)} - 1 \right)$

which solves for $N_\lambda + 1$ variables corresponding to $x^n_{i(\lambda)}$ and $\pi^n_\lambda$. For more information regarding this approach, refer to the following literature:

C.E. Harvie, J.P. Greenberg and J.H. Weare, "A Chemical Equilibrium Algorithm for Highly Non-Ideal Multiphase Systems: Free Energy Minimization," Geochimica et - Cosmochimica Acta, V. 51 (1987) 1045-1057.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnPhase-Index* | An integer scalar representing the absolute index of the solution phase that is being considered. |
| out | *lPhasePass* | A logical scalar indicating whether the phase should be added (i.e., TRUE) |

Definition at line 103 of file Subminimization.f90.

#### 5.23.2.5 subroutine SubMinInit ( integer *iSolnPhaseIndex,* integer *iterSubMax* )

Definition at line 200 of file Subminimization.f90.

#### 5.23.2.6 subroutine SubMinLineSearch ( integer *iSolnPhaseIndex* )

Definition at line 477 of file Subminimization.f90.

#### 5.23.2.7 subroutine SubMinNewton ( integer *iSolnPhaseIndex* )

Definition at line 409 of file Subminimization.f90.

## 5.24 src/shared/CheckPhaseAssemblage/SwapPureConForSoln-Phase.f90 File Reference

Swap a pure condensed phase for a solution phase.

### Functions/Subroutines

- subroutine SwapPureConForSolnPhase (iPhaseChange, lPhasePass)

### 5.24.1 Detailed Description

Swap a pure condensed phase for a solution phase.

**Author**

M.H.A. Piro

**Date**

May 2, 2012

**See also**

CheckPureConPhaseAdd.f90
AddPureConPhase.f90
SwapPureConPhase.f90
CheckPhaseChange.f90

Definition in file SwapPureConForSolnPhase.f90.

### 5.24.2 Function/Subroutine Documentation

#### 5.24.2.1 subroutine SwapPureConForSolnPhase ( integer *iPhaseChange*, logical *lPhasePass* )

The purpose of this subroutine is to add a partiuclar pure condnesed phase and remove a solution phase. The particular solution phase that will be removed will be determined in this subroutine.

**Parameters**

| | | |
|---|---|---|
| in | *iPhase-Change* | An integer scalar representing the absolute index of a pure condensed phase that is to be added to the system. |
| out | *lPhasePass* | A logical variable indicating whether the new phase assemblage has passed. |

Definition at line 54 of file SwapPureConForSolnPhase.f90.

## 5.25 src/shared/CheckPhaseAssemblage/SwapPureConPhase.f90 File Reference

Swap a pure condensed phase for another pure condensed phase.

### Functions/Subroutines

- subroutine SwapPureConPhase (iPhaseChange, lSwapLater, lPhasePass)

### 5.25.1 Detailed Description

Swap a pure condensed phase for another pure condensed phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckPureConPhaseAdd.f90
AddPureConPhase.f90
SwapPureConForSolnPhase.f90
CheckPhaseChange.f90

Definition in file SwapPureConPhase.f90.

### 5.25.2 Function/Subroutine Documentation

#### 5.25.2.1 subroutine SwapPureConPhase ( integer *iPhaseChange,* logical *lSwapLater,* logical *lPhasePass* )

The purpose of this subroutine is to swap one pure condensed phase for another pure condensed phase in the estimated phase assemblage .

**Parameters**

| | | |
|---|---|---|
| in | *iPhase-Change* | An integer scalar representing the absolute phase index of a pure condensed phase. |
| out | *lPhasePass* | A logical variable indicating whether the phase assemblage has passed or failed. |
| out | *lSwapLater* | A logical variable indicating whether the phase should be swapped later. |

Definition at line 47 of file SwapPureConPhase.f90.

## 5.26 src/shared/CheckPhaseAssemblage/SwapSolnForPureCon-Phase.f90 File Reference

Swap a particular solution phase for a pure condensed phase.

### Functions/Subroutines

- subroutine SwapSolnForPureConPhase (iPhaseChange, lPhasePass)

### 5.26.1 Detailed Description

Swap a particular solution phase for a pure condensed phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckSolnPhaseAdd.f90
CompMolSolnPhase.f90
CheckPhaseChange.f90

Definition in file SwapSolnForPureConPhase.f90.

### 5.26.2 Function/Subroutine Documentation

#### 5.26.2.1 subroutine SwapSolnForPureConPhase ( integer *iPhaseChange*, logical *lPhasePass* )

The purpose of this subroutine is to add a particular solution phase and remove a pure condnesed phase. This subroutine will determine which pure condensed phase is to be removed from the system. The iteration history is checked to ensure that this particular phase assemblage has not been previously considered.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseChange* | An integer scalar representing the index of a solution phase that is to be introduced to the system. |
| out | *lPhasePass* | A logical variable indicating whether a particular phase assemblage is appopriate for testing (TRUE) or not (FALSE). |

Definition at line 53 of file SwapSolnForPureConPhase.f90.

## 5.27 src/shared/CheckPhaseAssemblage/SwapSolnPhase.f90 File Reference

Swap a particular solution phase for another solution phase.

### Functions/Subroutines

- subroutine SwapSolnPhase (iPhaseChange, lPhasePass)

### 5.27.1 Detailed Description

Swap a particular solution phase for another solution phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CheckSolnPhaseAdd.f90
SwapSolnForPureConPhase.f90
CompMolSolnPhase.f90
CheckPhaseChange.f90

Definition in file SwapSolnPhase.f90.

### 5.27.2 Function/Subroutine Documentation

#### 5.27.2.1 subroutine SwapSolnPhase ( integer *iPhaseChange*, logical *lPhasePass* )

The purpose of this subroutine is to attempt to swap a particular solution phase for another solution phase in the current estimated phase assemblage. The logical variable lPhasePass returns a value of FALSE if the phase in question cannot swap for any other solution phase in the system.

**Parameters**

| | | |
|---|---|---|
| in | *iPhase-Change* | An integer scalar representing the index of a solution phase that is to be introduced to the system. |
| out | *lPhasePass* | A logical variable indicatin whether the phase assemblage is appropriate for consdieration (TRUE) or not (FALSE). |

Definition at line 59 of file SwapSolnPhase.f90.

## 5.28 src/shared/CheckPhaseAssemblage/SwapSolnPhaseSpecific.f90 File Reference

Swap one specific solution phase for another specific solution phase.

## Functions/Subroutines

- subroutine SwapSolnPhaseSpecific (iPhaseAdd, iPhaseRem, lPhasePass)

### 5.28.1 Detailed Description

Swap one specific solution phase for another specific solution phase.

**Author**

M.H.A. Piro

**Date**

May 8, 2012

**See also**

CheckSolnPhaseRem.f90
SwapSolnPhase.f90
CompMolSolnPhase.f90
CompStoichSolnPhase.f90
CompChemicalPotential.f90
CheckPhaseChange.f90

Definition in file SwapSolnPhaseSpecific.f90.

### 5.28.2 Function/Subroutine Documentation

#### 5.28.2.1 subroutine SwapSolnPhaseSpecific ( integer *iPhaseAdd*, integer *iPhaseRem*, logical *lPhasePass* )

The purpose of this subroutine is to attempt to swap a particular solution phase for another specific solution phase in the current estimated phase assemblage. The logical variable lPhasePass returns a value of FALSE if the phase in question cannot swap for any other solution phase in the system.

**Parameters**

| | | |
|---|---|---|
| in | *iPhaseAdd* | An integer scalar representing the absolute index of a solution phase that is to be added to the system. |
| in | *iPhaseRem* | An integer scalar representing the index of a solutoin phase that is to be removed from the system. |
| out | *lPhasePass* | A logical variable indicatin whether the phase assemblage is appropriate for consdieration (TRUE) or not (FALSE). |

Definition at line 57 of file SwapSolnPhaseSpecific.f90.

## 5.29 src/shared/CheckQKTOSolnPhase.f90 File Reference

Check if a QKTO solution phase should be added to the system.

### Functions/Subroutines

- subroutine CheckQKTOSolnPhase (iSolnIndex)

### 5.29.1 Detailed Description

Check if a QKTO solution phase should be added to the system.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

CompMolFraction.f90
PolyRegular.f90
KohlerInterpolate.f90
Broyden.f90

Definition in file CheckQKTOSolnPhase.f90.

### 5.29.2 Function/Subroutine Documentation

#### 5.29.2.1 subroutine CheckQKTOSolnPhase ( integer *iSolnIndex* )

The purpose of this subroutine is to determine whether a non-ideal solution phase based on the regular solution model with Kohler interpolation (QKTO) should be added to the system. This caculation can be relative computationally expensive and is only performed when absolutely necessary. The premise of this calculation is to compute a unique combination of hypothetitcal mole fractions for this particular QKTO phase that would be required for this phase to be in equilibrium with the system. An iterative procedure is required because the chemical potential of a constituent in a non-ideal solution phase is a non-linear function involving the standard molar Gibbs energy, the ideal mixing term, partial excess terms and its mole fraction.

Broyden's method is used to iteratively approximate the mole fraction of all constituents in the solution phase under consideration. The natural logarithm of the mole fraction is used instead of the mole fraction because the mole fraction varies by many orders of magnitude.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnIndex* | An integer scalar representing the absolute solution phase index. |

Definition at line 73 of file CheckQKTOSolnPhase.f90.

## 5.30 src/shared/CheckSysOnlyPureConPhases.f90 File Reference

Check the system when only pure condensed phases are expected to be stable.

### Functions/Subroutines

- subroutine CheckSysOnlyPureConPhases

### 5.30.1 Detailed Description

Check the system when only pure condensed phases are expected to be stable.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

CompMolFraction.f90
CheckQKTOSolnPhase.f90
CheckConvergence.f90

Definition in file CheckSysOnlyPureConPhases.f90.

### 5.30.2 Function/Subroutine Documentation

#### 5.30.2.1 subroutine CheckSysOnlyPureConPhases ( )

Definition at line 60 of file CheckSysOnlyPureConPhases.f90.

## 5.31 src/shared/CheckSystem.f90 File Reference

Check for consistency between the system and the data-file.

## Functions/Subroutines

- subroutine CheckSystem

### 5.31.1 Detailed Description

Check for consistency between the system and the data-file.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90
GetElementName.f90

Definition in file CheckSystem.f90.

### 5.31.2 Function/Subroutine Documentation

#### 5.31.2.1 subroutine CheckSystem ( )

The purpose of this subroutine is to ensure that the selection of system components in the parsed ChemSage data-file and the data provided to Thermochimica are consistent. System components are always taken to be chemical elements in Thermochimica, or "elements" for sake of brevity. An element will only be considered if thermodynamic data is provided by the data-file and if the mass of that particular element is provided. If the mass of a particular element is provided but there isn't any data for it (via the data-file), that element will not be considered. Similarly, if thermodynamic data (via the data-file) is provided for a particular element, but the mass of that element is not available, that element will not be considered.

This subroutine will select all species and phases that are relevant to this system from the data parsed from the ChemSage data-file.

The variable cInputThermo(3) can accept the following values:

| Units | Description |
|---|---|
| "mass fraction" | Mass fraction in dimensionless units (e.g., gram/gram, kilogram/kilogram, pound/pound, wt%). |
| "mole fraction" | Mole fraction in dimensionless units (e.g., mole/mole, mol%). |
| "atom fraction" | Atom fraction in dimensionless units (e.g., atom/atom, at%). |
| "kilograms" | All quantities are in kilograms. |
| "grams" | All quantities are in grams. |
| "pounds" | All quantities are in pounds. |
| "moles" | All quantities are in moles. |
| "gram-atoms" | All quantities are in gram-atoms (same as moles for the pure elements); |
| "atoms" | All quantities are in atoms. |

Definition at line 114 of file CheckSystem.f90.

## 5.32 src/shared/CheckThermoData.f90 File Reference

Check the thermodynamic database for pure species.

### Functions/Subroutines

- subroutine CheckThermoData

### 5.32.1 Detailed Description

Check the thermodynamic database for pure species.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90

Definition in file CheckThermoData.f90.

### 5.32.2 Function/Subroutine Documentation

#### 5.32.2.1 subroutine CheckThermoData ( )

The purpose of this subroutine is to ensure that the thermodynamic database is appropriate for use by Thermochimica. Specifically, at least one pure species of each element must be present in the database. A nonzero value of the integer scalar INFOThermo is returned if the thermodynamic database is inappropriate. This subroutine is also used as a placeholder for additional checks in future development. The following list gives a description of INFOThermo that is relevent to this subroutine.

0 - Successful exit; 9 - A pure chemical species does not exist for at least one element.

Definition at line 60 of file CheckThermoData.f90.

## 5.33 src/shared/CheckThermoInput.f90 File Reference

Check the input quantities and character units.

### Functions/Subroutines

- subroutine CheckThermoInput

### 5.33.1 Detailed Description

Check the input quantities and character units.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90

Definition in file CheckThermoInput.f90.

### 5.33.2 Function/Subroutine Documentation

#### 5.33.2.1 subroutine CheckThermoInput ( )

The purpose of this subroutine is to apply a unit conversion to the input variables (if necessary) and to ensure that the input variables are appropriate. The working units for

temperature, absolute hydrostaic pressure and mass are Kelvin [K], atmospheres [atm] and moles [mol], respectively. These variables are tested to ensure that they are within an acceptable range and that they are real. An integer scalar INFOThermo is returned in a similar style as LAPACK to identify an error.

A description of each value of INFOThermo that could be returned from this subroutine is given below:

- 0 - Successful exit,
- 1 - Temperature is out of range or a NAN,
- 2 - Hydrostatic pressure is out of range or a NAN,
- 3 - The mass of any element is out of range or a NAN, and
- 4 - The character string representing the input units is unrecognizable.

The variable cThermoInputUnits can take on the following values:

| cThermoInputUnits | Description |
|---|---|
| "K" | Temperature in Kelvin |
| "C" | Temperature in Celsius |
| "F" | Temperature in Fahrenheit |
| "R" | Temperature in Rankine |
| "atm" | Pressure in atmospheres |
| "psi" | Pressure in pounds per square inch |
| "bar" | Pressure in bars |
| "Pa" | Pressure in Pascals |
| "kPa" | Pressure in kiloPascals |

Definition at line 103 of file CheckThermoInput.f90.

## 5.34 src/shared/CompChemicalPotential.f90 File Reference

Compute the chemical potentials of all solution phase constituents.

### Functions/Subroutines

- subroutine CompChemicalPotential (lCompEverything)

### 5.34.1 Detailed Description

Compute the chemical potentials of all solution phase constituents.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

CompMolFraction.f90

Definition in file CompChemicalPotential.f90.

### 5.34.2 Function/Subroutine Documentation

#### 5.34.2.1 subroutine CompChemicalPotential ( logical *lCompEverything* )

The purpose of this subroutine is to compute the chemical potentials of all solution phase constituents expected to be stable at equilibrium.

**Parameters**

| | | |
|---|---|---|
| in | *lComp-Everything* | A logical scalar indicating whether everything should be computed, or only what is necessary. For the most part, it is only necessary to compute chemical potentials of solution species that are expected to be stable. |

Definition at line 45 of file CompChemicalPotential.f90.

## 5.35 src/shared/CompExcessQKTO.f90 File Reference

Compute the partial molar excess Gibbs energy of mixing of solution phase constituents in a QKTO solution phase.

### Functions/Subroutines

- subroutine CompExcessQKTO (iSolnIndex)

### 5.35.1 Detailed Description

Compute the partial molar excess Gibbs energy of mixing of solution phase constituents in a QKTO solution phase.

**Author**

M.H.A. Piro

**See also**

Subminimization.f90

**Date**

June 13, 2012

Definition in file CompExcessQKTO.f90.

### 5.35.2 Function/Subroutine Documentation

#### 5.35.2.1 subroutine CompExcessQKTO ( integer *iSolnIndex* )

The purpose of this subroutine is to compute the partial molar excess Gibbs energy of mixing (dPartialExcessGibbs) of all constituents in a non-ideal solution phase designated as 'QKTO' (Quasi-chemical Kohlter-TOop). The PolyRegular subroutine computes the excess Gibbs energy of mixing of a regular solution sub-system (see PolyRegular for a definition) and the KohlerInterpolate subroutine performs a Kohler interpolation of a sub-system to a phase.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnIndex* | Absolute index of a solution phase |

Definition at line 48 of file CompExcessQKTO.f90.

## 5.36 src/shared/CompFunctionNorm.f90 File Reference

Compute the functional norm for the line search.

### Functions/Subroutines

- subroutine CompFunctionNorm

### 5.36.1 Detailed Description

Compute the functional norm for the line search.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

GEMLineSearch.f90
CompStoichSolnPhase.f90

Definition in file CompFunctionNorm.f90.

### 5.36.2 Function/Subroutine Documentation

#### 5.36.2.1 subroutine CompFunctionNorm ( )

The purpose of this subroutine is to compute the functional norm for the line search algorithm to determine whether the system is converging sufficinetly or diverging. The functional vector is not directly computed because it is not needed. Thus, the functional norm is computed directly. This term incorporates the residuals of the mass balance equations, the average residual between the chemical potential of each species and the corresponding value computed from the element potentials.

Definition at line 41 of file CompFunctionNorm.f90.

## 5.37 src/shared/CompGibbsMagnetic.f90 File Reference

Compute magnetic contributions to the Gibbs energy terms.

### Functions/Subroutines

- subroutine CompGibbsMagnetic (i, j)

### 5.37.1 Detailed Description

Compute magnetic contributions to the Gibbs energy terms.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

CompThermoData.f90

**Parameters**

| | | |
|---|---|---|
| in | *i* | Gibbs energy equation coefficient |
| in | *j* | Species index |
| in,out | *dChemical-Potential* | A double real vector representing the chemical potential. |

Definition in file CompGibbsMagnetic.f90.

### 5.37.2 Function/Subroutine Documentation

#### 5.37.2.1 subroutine CompGibbsMagnetic ( integer *i*, integer *j* )

The purpose of this subroutine is to compute the magnetic contribution to the standard molar Gibbs energy of a pure species. This contribution is given by $\Delta G_{mag} = RTln(B_o+1)g(\tau)$, where $B_o$ is the average magnetic moment per atom, $\tau$ is the critical temperature (i.e., the Curie temperature for ferromagnetic materials or the Neel temperature for antiferromagnetic materials) and $g$ is a function of $\tau$.

The following reference explains the magnetic contribution to the Gibbs energy term that is used in this subroutine:

- A.T. Dinsdale, "SGTE Data for Pure Elements," CALPHAD, 15, 4 (1991) 317-425.

**Parameters**

| | | |
|---|---|---|
| in | *i* | An integer scalar corresponding to the Gibbs energy index. |
| in | *j* | An integer scalar representing the species index. |

Definition at line 50 of file CompGibbsMagnetic.f90.

## 5.38 src/shared/CompMolAllSolnPhases.f90 File Reference

Compute the number of moles of all stable pure condensed and solution phases.

### Functions/Subroutines

- subroutine CompMolAllSolnPhases

### 5.38.1 Detailed Description

Compute the number of moles of all stable pure condensed and solution phases.

**Author**

M.H.A. Piro

**Date**

June 20, 2012

**See also**

AddSolnPhase.f90
SwapSolnPhase.f90
SwapSolnForPureConPhase.f90
CompMolSolnPhase.f90

Definition in file CompMolAllSolnPhases.f90.

### 5.38.2 Function/Subroutine Documentation

#### 5.38.2.1 subroutine CompMolAllSolnPhases ( )

The purpose of this subroutine is to estimate the number of moles of each solution and pure condensed phase in the system. The stoichiometry of each solution phase and (obviously) pure condensed phase is fixed and this subroutine determines a particular combination of molar quantities that minimizes the mass balance residuals using a - Linear Leasr Squares (LLS) technique.

This subroutine addresses a particular issue that occasionally arises when a pure condensed phase is driven out of the system (when it should), but it also drives a solution phase to almost be removed when it shouldn't. At this point, the system may have sufficiently diverged that it may be impossible for the system to recover. The element potentials may be close to the equilibrium values, but the molar quantities of solution speices are way off. This subroutine therefore fixes the chemical potentials and determines the best combination of molar quantities of the phases.

Definition at line 51 of file CompMolAllSolnPhases.f90.

## 5.39 src/shared/CompMolFraction.f90 File Reference

Compute the mole fraction of all solution phase constituents of a particular solution phase.

### Functions/Subroutines

- subroutine CompMolFraction (k)

### 5.39.1 Detailed Description

Compute the mole fraction of all solution phase constituents of a particular solution phase.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

Definition in file CompMolFraction.f90.

### 5.39.2 Function/Subroutine Documentation

#### 5.39.2.1 subroutine CompMolFraction ( integer *k* )

**See also**

InitGEMSolver.f90
CompChemicalPotential.f90
CheckSolnPhaseAdd.f90
AddSolnPhase.f90
RemPureConAddSolnPhase.f90
SwapSolnPhase.f90
SwapSolnForPureConPhase.f90

The purpose of this subroutine is to compute mole fractions of all solution phase constituents, the sum of mole fractions of all constituents in a solution phase (i.e., dSumMolFractionSoln) and the effective stoichiometry of each solution phase (i.e., dEffStoichSolnPhase). The relationship between the mole fraction of a solution phase constituent and its chemical potential depends on the type of solution phase. The solution phase types that are supported follow:

**Parameters**

| | | |
|---|---|---|
| in | *k* | Absolute solution phase index. |

Definition at line 92 of file CompMolFraction.f90.

## 5.40 src/shared/CompMolSolnPhase.f90 File Reference

Compute the number of moles of all stable solution phases.

### Functions/Subroutines

- subroutine CompMolSolnPhase

### 5.40.1 Detailed Description

Compute the number of moles of all stable solution phases.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

AddSolnPhase.f90
SwapSolnPhase.f90
SwapSolnForPureConPhase.f90

Definition in file CompMolSolnPhase.f90.

### 5.40.2 Function/Subroutine Documentation

#### 5.40.2.1 subroutine CompMolSolnPhase ( )

The purpose of this subroutine is to estimate the number of moles of each solution phase in the system when a new solution phase is added. This can be very important to consider because an initial estimate of zero moles would mean that the solution phase constituents would not contribute to the Jacobian matrix. A more serious issue is if the system is initially estimated to be only comprised of pure stoichiometric phases and then a solution phase is added, then the upper left quadrant of the Jacobain matrix would be zero.

The number of moles of all pure condensed phases are fixed and the number of moles of solution phases are computed using a Linear Least Squares (LLS) approach. The linear minimization in this subroutine involves the relative errors of the mass balances instead of the residuals. Normalizing each coefficient of matrix A and vector B by the number of moles of the corresponding element provides greater generality when the number of moles of different elements varies by many orders of magnitude.

Definition at line 67 of file CompMolSolnPhase.f90.

## 5.41 src/shared/CompStoichSolnPhase.f90 File Reference

Compute the stoichiometry of a solution phase.

### Functions/Subroutines

- subroutine CompStoichSolnPhase (k)

### 5.41.1 Detailed Description

Compute the stoichiometry of a solution phase.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

Definition in file CompStoichSolnPhase.f90.

### 5.41.2 Function/Subroutine Documentation

#### 5.41.2.1 subroutine CompStoichSolnPhase ( integer *k* )

The purpose of this subroutine is to compute the effective stoichiometry of a particular solution phase.

**Parameters**

| | | |
|---|---|---|
| in | *k* | An integer scalar representing the absolute solution phase index. |

Definition at line 47 of file CompStoichSolnPhase.f90.

## 5.42 src/shared/CompThermoData.f90 File Reference

Compute thermodynamic data.

### Functions/Subroutines

- subroutine CompThermoData

### 5.42.1 Detailed Description

Compute thermodynamic data.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90
CompGibbsMagnetic.f90

Definition in file CompThermoData.f90.

### 5.42.2 Function/Subroutine Documentation

#### 5.42.2.1 subroutine CompThermoData ( )

The purpose of this subroutine is to compute thermodynamic data for all substances in the system. The specific variables that are computed include:

- the standard molar Gibbs energy for each pure substance using the specified temperature and pressure,
- the excess molar Gibbs energy of mixing for each sub-system,
- the total number of atoms per formula mass for each compound,
- the atomic fraction of each element in a particular compound, and
- construction of the Hessian matrix, which will be used later by the GEMNewton subroutine.

The coefficients for the standard molar Gibbs energy equations for pure species originate from a ChemSage data-file that was parsed from the ParseCSDataFile program. The format for the coefficients follow:

$g_i^\circ = A + BT + CT\,ln(T) + DT^2 + ET^3 + F/T + (GT^U + HT^V + IT^W + JT^X + KT^Y + LT^Z)$

Note that the terms in paranteses in the above equation are additional terms. Some of the exponents used in the additional terms may be 99. This corresponds to the natural logarithm.

For some very strange reason, the B coefficient in the above equation is modified by $\sim$0.10945 J/mol for only gaseous species when FactSage generates a ChemSage data-file. Another very peculiar observation is that if a database is constructed in FactSage using the new ChemSage data-file, FactSage will automatically remove this quantity from the B coefficient. The B coefficient for gaseous species is corrected in this subroutine.

Definition at line 86 of file CompThermoData.f90.

## 5.43 src/shared/GEMDebug.f90 File Reference

### Functions/Subroutines

- subroutine GEMDebug (iDebug)

### 5.43.1 Function/Subroutine Documentation

#### 5.43.1.1 subroutine GEMDebug ( integer *iDebug* )

Definition at line 2 of file GEMDebug.f90.

## 5.44 src/shared/GEMLineSearch.f90 File Reference

Perform a line search for the GEMSolver.f90.

### Functions/Subroutines

- subroutine GEMLineSearch
- subroutine InitGEMLineSearch (dStepLength, dMolesSpeciesLast, dElementPotentialLast)
- subroutine UpdateSystemVariables (dStepLength, dMolesSpeciesLast, dElementPotentialLast)

### 5.44.1 Detailed Description

Perform a line search for the GEMSolver.f90.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

GEMSolver.f90
GEMNewton.f90
CompChemicalPotential.f90
CompFunctionNorm.f90

Definition in file GEMLineSearch.f90.

### 5.44.2 Function/Subroutine Documentation

#### 5.44.2.1 subroutine GEMLineSearch ( )

The purpose of this subroutine is to perform a line search using the direction vector computed by the Newton/Broyden solver. The system is updated using an appropriate step length that satisfies the Wolfe conditions. Specifically, values of dChemicalPotential and dMolesPhase are updated. It is possible for the system of equations to be ill- behaved and yield inappropriate results. An initial step-length is computed by normalizing the largest change of the system variables by a pre-defined value. The maximum change to the element potentials is 1 and the maximum change to the number of moles of a solution phase is twice of the previous value. For more information, refer to Chapter 6 of the above reference.

Definition at line 72 of file GEMLineSearch.f90.

#### 5.44.2.2 subroutine InitGEMLineSearch ( real(8) *dStepLength,* real(8), dimension(nspecies) *dMolesSpeciesLast,* real(8), dimension(nelements) *dElementPotentialLast* )

Definition at line 224 of file GEMLineSearch.f90.

#### 5.44.2.3 subroutine UpdateSystemVariables ( real(8) *dStepLength,* real(8), dimension(nspecies) *dMolesSpeciesLast,* real(8), dimension(nelements) *dElementPotentialLast* )

Definition at line 403 of file GEMLineSearch.f90.

## 5.45 src/shared/GEMNewton.f90 File Reference

Compute the direction vector for the GEMSolver using Newton's method.

### Functions/Subroutines

- subroutine GEMNewton (INFO)

  *The purpose of this subroutine is to compute the direction vector for the Gibbs energy minimization (GEM) solver using Newton's method. The Hessian matrix and its corresponding constraint vector are first constructed and then the direction vector representing the system parameters is solved with the DGESV driver routine from LAPACK. The updated element potentials, adjustments to the number of moles of solution phases and the number of moles of pure condensed phases are applied in the GEMLineSearch.f90 subroutine.*

### 5.45.1 Detailed Description

Compute the direction vector for the GEMSolver using Newton's method.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

GEMSolver.f90
GEMLineSearch.f90

Definition in file GEMNewton.f90.

## 5.45.2 Function/Subroutine Documentation

### 5.45.2.1 subroutine GEMNewton ( integer *INFO* )

The purpose of this subroutine is to compute the direction vector for the Gibbs energy minimization (GEM) solver using Newton's method. The Hessian matrix and its corresponding constraint vector are first constructed and then the direction vector representing the system parameters is solved with the DGESV driver routine from LAPACK. The updated element potentials, adjustments to the number of moles of solution phases and the number of moles of pure condensed phases are applied in the GEMLineSearch.f90 subroutine.

For further information regarding this methodology, refer to the following material:

- W.B. White, S.M. Johnson, G.B. Dantzig, "Chemical Equilibrium in Complex Mixtures," Journal of Chemical Physics, V. 28, N. 5, 1958.
- G. Eriksson, "Thermodynamic Studies of High Temperature Equilibria," Acta - Chemica Scandinavica, 25, 1971.
- G. Eriksson, E. Rosen, "General Equations for the Calculation of Equilibria in Multiphase Systems," Chemica Scripta, 4, 1973.

**Parameters**

| | | |
|---|---|---|
| out | *INFO* | An integer scalar used by LAPACK indicating a successful exit or an error. |

Definition at line 73 of file GEMNewton.f90.

## 5.46 src/shared/GEMSolver.f90 File Reference

Gibbs Energy Minimization solver.

### Functions/Subroutines

- subroutine GEMSolver

### 5.46.1 Detailed Description

Gibbs Energy Minimization solver.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

Thermochimica.f90
InitGEMSolver.f90
GEMNewton.f90
GEMLineSearch.f90
CheckPhaseAssemblage.f90
CheckConvergence.f90

Definition in file GEMSolver.f90.

### 5.46.2 Function/Subroutine Documentation

#### 5.46.2.1 subroutine GEMSolver ( )

The purpose of this subroutine is to compute the quantities of species and phases at thermodynamic equilibrium using the Gibbs Energy Minimization (GEM) method. This subroutine uses values of dMolesPhase, dChemicalPotential and iAssemblage from the Leveling and PostLeveling subroutines as initial estimates for computation.

The main subroutines used by this solver are summarized below:

| File name | Description |
|---|---|
| InitGEMSolver.f90 | Initialize the GEMSolver by establishing the initial phase assemblage and composition. |
| CheckSysOnlyPureConPhases.f90 | Check the system if there are only pure condensed phases. The system may already be converged. |
| GEMNewton.f90 | Compute the direction vector using Newton's method. |
| GEMLineSearch.f90 | Perform a line search along the direction vector. |
| CheckPhaseAssemblage.f90 | Check if the phase assemblage needs to be adjusted. |
| CheckConvergence.f90 | Check if the system has converged. |

Definition at line 85 of file GEMSolver.f90.

## 5.47 src/shared/GetElementName.f90 File Reference

Get the name of each chemical element.

### Functions/Subroutines

- subroutine GetElementName (cElementNamePT)

### 5.47.1 Detailed Description

Get the name of each chemical element.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

CheckSystem.f90

Definition in file GetElementName.f90.

### 5.47.2 Function/Subroutine Documentation

#### 5.47.2.1 subroutine GetElementName ( character(3), dimension(0:nelementspt) *cElementNamePT* )

The purpose of this subroutine is to return the name of each chemical element on the periodic table.

Definition at line 27 of file GetElementName.f90.

## 5.48 src/shared/GetFirstAssemblage.f90 File Reference

Determine the first phase assemblage for testing.

### Functions/Subroutines

- subroutine GetFirstAssemblage

### 5.48.1 Detailed Description

Determine the first phase assemblage for testing.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

LevelingSolver.f90

Definition in file GetFirstAssemblage.f90.

### 5.48.2 Function/Subroutine Documentation

#### 5.48.2.1 subroutine GetFirstAssemblage ( )

The purpose of this subroutine is to estimate the very first phase assemblage for the Leveling subroutine. Only the pure species will be considered here, since any possible combination of the pure species will yield a positive number of moles. In more mathematical terms, the stoichiometry matrix is a diagonal matrix when the assemblage is comprised of only pure species.

Definition at line 56 of file GetFirstAssemblage.f90.

## 5.49 src/shared/GetNewAssemblage.f90 File Reference

Determine the next phase assemblage to be considered in Leveling.

### Functions/Subroutines

- subroutine GetNewAssemblage (iter)

### 5.49.1 Detailed Description

Determine the next phase assemblage to be considered in Leveling.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

ShuffleAssemblage.f90
LevelingSolver.f90
PostLevelingSolver.f90

Definition in file GetNewAssemblage.f90.

### 5.49.2 Function/Subroutine Documentation

#### 5.49.2.1 subroutine GetNewAssemblage ( integer *iter* )

The purpose of this subroutine is to provide a new estimated phase assemblage to be tested in the Leveling subroutine. The phase with the most negative relative Gibbs energy will be introduced into the previous estimated phase assemblage. In order to avoid violating the Gibbs Phase Rule, this new phase will replace a phase from the previous assemblage. The conditions for considering a new phase assemblage are:

1. The number of moles of each phase must be non-negative and real,
2. The phase assemblage has not been previously tested (this is only perfomed after x iterations),
3. The numerical adjustments to the Gibbs Plane are real. This last check does not add any additional expense because this would have to be computed anyways.

There is an important, yet subtle, verification performed in the third (3) condition. - The Phase Rule dictates that the maximum number of phases that can coexist in an isobaric-isothermal closed system cannot exceed the number of system components (elements). An additional condition to the Phase Rule, which is normally implied in thermodynamics texts but not explicity stated, is that only one pure separate phase of one component can exist at equilibrium. For example, U(BCC) and U(FCC) cannot coexist. If two pure separate phases of the same component (same X) are included in the phase assemblage, then the Gibbs Plane is not uniquely defined (A is not a unique matrix) resulting in non-real adjustments to the Gibbs Plane. By ensuring that the adjustments applied to the Gibbs Plane are real not only avoids obvious numerical problems, but also guarentees that the Phase Rule is explicitly satisfied.

Definition at line 78 of file GetNewAssemblage.f90.

## 5.50 src/shared/InitGEMSolver.f90 File Reference

Initialize the GEMSolver.f90 subroutine.

### Functions/Subroutines

- subroutine InitGEMSolver
- subroutine InitGemCheckSoInPhase

### 5.50.1 Detailed Description

Initialize the GEMSolver.f90 subroutine.

**Author**

M.H.A. Piro

**Date**

Apr. 25, 2012

**See also**

GEMSolver.f90

Definition in file InitGEMSolver.f90.

### 5.50.2 Function/Subroutine Documentation

#### 5.50.2.1 subroutine InitGemCheckSoInPhase ( )

Definition at line 303 of file InitGEMSolver.f90.

#### 5.50.2.2 subroutine InitGEMSolver ( )

The purpose of this subroutine is to initialize the GEMSolver.f90 subroutine. Specifically, this subroutine determines which pure condensed phases and solution phases are initially estimated to contribute to the equilibrium phase assemblage. Initial estimates of the quantity of each phase was determined by the LevelingSolver.f90 subroutine. Also, many allocatable arrays are allocated in this subroutine.

Definition at line 52 of file InitGEMSolver.f90.

## 5.51 src/shared/InitThermo.f90 File Reference

Initialize Thermochimica.

### Functions/Subroutines

- subroutine InitThermo

### 5.51.1 Detailed Description

Initialize Thermochimica.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**Returns**

dNormalizeInput

**See also**

Thermochimica.f90

Definition in file InitThermo.f90.

### 5.51.2 Function/Subroutine Documentation

#### 5.51.2.1 subroutine InitThermo ( )

The purpose of this subroutine is to initialize Thermochimica. Various physical constants and numerical constants are defined.

Definition at line 41 of file InitThermo.f90.

## 5.52 src/shared/KohlerInterpolate.f90 File Reference

Perform a Kohler interpolation for excess mixing terms.

### Functions/Subroutines

- subroutine KohlerInterpolate (iSolnIndex, iParam, xT, dGParam, dPartialGParam)

### 5.52.1 Detailed Description

Perform a Kohler interpolation for excess mixing terms.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

PolyRegular.f90

Definition in file KohlerInterpolate.f90.

### 5.52.2 Function/Subroutine Documentation

#### 5.52.2.1 subroutine KohlerInterpolate ( integer *iSolnIndex,* integer *iParam,* real(8) *xT,* real(8) *dGParam,* real(8), dimension(nmaxparam) *dPartialGParam* )

The purpose of this subroutine is to perform a Kohler interpolation of binary/ternary/quaternary model parameters (provided by PolyRegular.f90) in multi-component phases and return the partial molar excess Gibbs energy of mixing of a species in a non-ideal solution phase (QKTO).

For more information regarding the Kohler interpolation method and the derivation of the equations used in this subroutine, refer to the following paper:

- A.D. Pelton and C.W. Bale, "Computational Techniques for the Treatment of -Thermodynamic Data in Multicomponent Systems and the Calculation of Phase Equilibria," CALPHAD, V. 1, N. 3 (1977) 253-273.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnIndex* | An integer scalar representing the index of a solution phase. |
| in | *iParam* | An integer scalar representing the mixing parameter index. |
| in | *xT* | Sum of mole fractions of actual species in solution phase |
| in | *dGParam* | Excess Gibbs energy of sub-system |
| in | *dPartialG-Param* | Partial excess Gibbs energy of species in sub-system |

Definition at line 59 of file KohlerInterpolate.f90.

## 5.53 src/shared/LevelingSolver.f90 File Reference

A linear solver that estimates thermodynamic equilibrium.

### Functions/Subroutines

- subroutine LevelingSolver

### 5.53.1 Detailed Description

A linear solver that estimates thermodynamic equilibrium.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90
GetFirstAssemblage.f90
GetNewAssemblage.f90
PostLevelingSolver.f90

Definition in file LevelingSolver.f90.

### 5.53.2 Function/Subroutine Documentation

#### 5.53.2.1 subroutine LevelingSolver ( )

The purpose of this subroutine is to provide initial estimates of the equilibrium phase assemblage for the GEMSolver using the "Leveling" technique of Eriksson and Thompson.

The fundamental principle of the leveling technique is to temporarily assume that all species may be treated as pure stoichiometric phases. In more mathematical terms, the logarithmic term in the chemical potential function of a solution species becomes zero and the Gibbs energy of the system becomes a linear function. The Leveling subroutine can therefore be interpreted as a linear optimizer that provides initial estimates for the GEMSolver subroutine.

There are several advantages in employing the Leveling technique:

1. Close estimates are provided for dElementPotential and dChemicalPotential,
2. An estimate is provided for the phase assemblage, which is often fairly close to the equilibrium assemblage (i.e., which phases are expected to be most stable),
3. The estimated phase assemblage is determined with relatively few iterationsm, and
4. Estimates are provided for the mole fractions of all constituents in all solution phases.

For further information regarding the "Leveling" method, refer to the following literature:

- G. Eriksson and W.T. Thompson, "A Procedure to Estimate Equilibrium Concentrations in Multicomponent Systems and Related Applications," CALPHAD, V. 13, N. 4, pp. 389-400 (1989).
- M.H.A. Piro, "Computation of Thermodynamic Equilibria Pertinent to Nuclear Materials in Multi-Physics Codes," PhD Dissertation, Royal Military College of Canada, 2011.
- M.H.A. Piro, M.J. Welland, B.J. Lewis, W.T. Thompson and D.R. Olander, "-Development of a Self-Standing Numerical Tool to Compute Chemical Equilibria in Nuclear Materials," Top Fuel Conference, Paris, France (2009).

Definition at line 98 of file LevelingSolver.f90.

## 5.54 src/shared/ModuleGEMSolver.f90 File Reference

Fortran module for input/output of the non-linear solver.

### Data Types

- module ModuleGEMSolver

### 5.54.1 Detailed Description

Fortran module for input/output of the non-linear solver.

**Author**

M.H.A. Piro

Definition in file ModuleGEMSolver.f90.

## 5.55 src/shared/ModuleSubMin.f90 File Reference

Fortran module for the Subminimization routine.

### Data Types

- module ModuleSubMin

### 5.55.1 Detailed Description

Fortran module for the Subminimization routine. The purpose of this module is to

**Author**

M.H.A. Piro

Definition in file ModuleSubMin.f90.

## 5.56 src/shared/ModuleThermo.f90 File Reference

Fortran module for internal use of Thermochimica.

### Data Types

- module ModuleThermo

### 5.56.1 Detailed Description

Fortran module for internal use of Thermochimica. The purpose of this module is to provide the means to share information amoungst the various subroutines used by - Thermochimica

**Author**

M.H.A. Piro

Definition in file ModuleThermo.f90.

## 5.57 src/shared/ModuleThermoIO.f90 File Reference

Fortran module for input/output of Thermochimica.

### Data Types

- module ModuleThermoIO

### 5.57.1 Detailed Description

Fortran module for input/output of Thermochimica. The purpose of this module is to provide the means to share information between Thermochimica and any software that calls it.

**Author**

M.H.A. Piro

**See also**

ThermoDebug.f90

Definition in file ModuleThermoIO.f90.

## 5.58 src/shared/Parser/ModuleParseCS.f90 File Reference

A Fortran module used to store data for the ChemSage parser.

### Data Types

- module ModuleParseCS

### 5.58.1 Detailed Description

A Fortran module used to store data for the ChemSage parser.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

Definition in file ModuleParseCS.f90.

## 5.59 src/shared/Parser/ParseCSDataBlock.f90 File Reference

Parse the data block section of a ChemSage data-file.

### Functions/Subroutines

- subroutine ParseCSDataBlock

### 5.59.1 Detailed Description

Parse the data block section of a ChemSage data-file.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

ParseCSDataFile.f90
ParseCSDataBlockGibbs.f90

Definition in file ParseCSDataBlock.f90.

### 5.59.2 Function/Subroutine Documentation

#### 5.59.2.1 subroutine ParseCSDataBlock ( )

The purpose of this subroutine is to parse the "data block" section of a ChemSage data-file. The "data block" section of this data-file contains the thermodynamic data of all species and phases in the system, including the stoichiometry coefficients of all species and coefficients of the standard Gibbs energy equations. Details about the INFO error codes are given in ParseCSDataFile.f90.

Definition at line 56 of file ParseCSDataBlock.f90.

## 5.60 src/shared/Parser/ParseCSDataBlockGibbs.f90 File Reference

Parse the coefficients of the Gibbs energy equations in the datablock section of a ChemSage data-file.

## Functions/Subroutines

- subroutine ParseCSDataBlockGibbs (i, j, iCounterGibbsEqn)

### 5.60.1 Detailed Description

Parse the coefficients of the Gibbs energy equations in the datablock section of a ChemSage data-file.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

ParseCSDataBlock.f90

Definition in file ParseCSDataBlockGibbs.f90.

### 5.60.2 Function/Subroutine Documentation

#### 5.60.2.1 subroutine ParseCSDataBlockGibbs ( integer *i*, integer *j*, integer *iCounterGibbsEqn* )

The purpose of this subroutine is to parse the coefficients of the Gibbs energy equations in the "data block" section of a ChemSage data-file and store the coefficients for computation in another program. Details about the INFO error codes are given in - ParseCSDataFile.f90. Details about entry numbers are explained in the "ChemApp Programmer's manual".

Chemical species may include solution species, pure separate phases and what I call "dummy species". Dummy species do not have any physical significance and are only included in ChemSage datafiles for numerical purposes. Dummy species are always pure species (i.e., U, O or Pu; as opposed to UO2 and Pu5O8) and are included when the database does not include any pure species. Dummy species are identified in - ChemSage data-files as species that follow pure separate phases that do not end their character string with ")". All pure separate phases will end their character string with "(s)".

**Parameters**

| | | |
|---|---|---|
| in | *i* | An integer scalar representing the phase index. |
| in | *j* | An integer scalar representing the species index. |
| in | *iCounter-GibbsEqn* | An integer scalar representing the index of the Gibbs energy equation. |

Definition at line 67 of file ParseCSDataBlockGibbs.f90.

## 5.61 src/shared/Parser/ParseCSDataFile.f90 File Reference

Parse a ChemSage data-file.

### Functions/Subroutines

- subroutine ParseCSDataFile (cFileName)

### 5.61.1 Detailed Description

Parse a ChemSage data-file.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

ParseCSHeader.f90
ParseCSDataBlock.f90
ModuleParseCS.f90

Definition in file ParseCSDataFile.f90.

### 5.61.2 Function/Subroutine Documentation

#### 5.61.2.1 subroutine ParseCSDataFile ( character(*) *cFileName* )

The purpose of this subroutine is to parse a ChemSage data-file into a format that can be used by Thermochimica. The intended purpose of this program is to be called once in AMP and to store the parsed information to memory to minimize the number of times (to one) that a data-file has to be read from the hard disk.

First, the "header section" of the data-file is parsed to gather important information to allocate memory. Second, the "data block" section is parsed to store thermodynamic data of all species and phases in the system. A description of all the pertinent variables can be found in the ParseCSModule module.

This subroutine is looking for a specific file specified by the variable cFileName and it will return a non-zero value for INFO if there is an error. The error code returned by this program is represented by the integer scalar INFO and is described below:

| INFO Value | Description |
|---|---|
| 0 | Successful exit |
| 1 | The specified data-file was not found in the local directory |
| 2 | The number of elements in the system exceeds the maximum allowable number |
| 3 | The number of solution phases in the system exceeds the maximum allowable number |
| 4 | The solution phase type is not supported |
| 5 | The number of Gibbs energy equations has exceeded the amount allocated to memory |
| 1x | Error reading line x of the header section (e.g., INFO = 13 means that line 3 of the header section failed). The line numbering convention from the ChemApp programmer's library is used here |
| 1xab | Error reading entry x of the data block section, where ab refers to the number of the solution phase that is causing the problem (e.g., INFO = 1234 means that entry 2 of the data block section had an error with solution phase 34). The entry numbering convention from the ChemApp programmer's library is used here. |

This program is not capable of parsing all types of ChemSage data-files. The following is a summary of known limitations:

- Only IDMX and QKTO solution phases can be considered.
- Pure separate phases can be considered.
- Aqueous phases cannot be considered.
- Only Gibbs energy equations of the '4' type can be considered.
- The maximum number of solution phases is 42. The only reason why this is constrained is because this is the maximum number of solution phases that FactSage can handle.
- The maximum number of elements is 118 (limited by the periodic table)
- The maximum number of components in a sub-system for a particular interaction parameter is set to 4 (i.e., quaternary). Note that this limitation is for a sub-system, not for a solution phase.

**Parameters**

| | | |
|---|---|---|
| in | *cFileName* | A character string representing the path and name of the ChemSage data-file. |

Definition at line 113 of file ParseCSDataFile.f90.

## 5.62 src/shared/Parser/ParseCSHeader.f90 File Reference

Parse the header section of a ChemSage data-file.

### Functions/Subroutines

- subroutine ParseCSHeader

### 5.62.1 Detailed Description

Parse the header section of a ChemSage data-file.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

ParseCSDataFile.f90

Definition in file ParseCSHeader.f90.

### 5.62.2 Function/Subroutine Documentation

#### 5.62.2.1 subroutine ParseCSHeader ( )

The purpose of this subroutine is to parse the "header section" of a ChemSage data-file. The "Header Section" of this datafile contains important information for allocating memory, such as the number of elements in the system, number of solution phases in the system, the number of chemical species etc. Details about the INFO error codes are given in ParseCSDataFile.f90. Details about line numbers are explained in the "-ChemApp Programmer's manual".

Definition at line 72 of file ParseCSHeader.f90.

## 5.63 src/shared/PolyRegular.f90 File Reference

Compute the partial molar excess Gibbs energy of a polynomial regular solution model.

### Functions/Subroutines

- subroutine PolyRegular (iSolnIndex, iParam, xT, dGParam, dPartialGParam)

### 5.63.1 Detailed Description

Compute the partial molar excess Gibbs energy of a polynomial regular solution model.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

KohlerInterpolate.f90

Definition in file PolyRegular.f90.

### 5.63.2 Function/Subroutine Documentation

#### 5.63.2.1 subroutine PolyRegular ( integer *iSolnIndex*, integer *iParam*, real(8) *xT*, real(8) *dGParam*, real(8), dimension(nmaxparam) *dPartialGParam* )

The purpose of this subroutine is to compute the partial molar excess Gibbs energy of mixing of species in a regular sub-system. Note that the effective quantity (i.e., "y") is represented relative to the particular parameter of interest and differs from that of the phase as a whole.

**Parameters**

| | | |
|---|---|---|
| in | *iSolnIndex* | Integer scalar of the solution index. |
| in | *iParam* | Integer scalar of the parameter index. |
| out | *xT* | Sum of mole fractions of actual species in solution phase |
| out | *dGParam* | Excess Gibbs energy of sub-system |
| out | *dPartialGParam* | Partial excess Gibbs energy of a species in the sub-system |

Definition at line 55 of file PolyRegular.f90.

## 5.64 src/shared/PostLevelingSolver.f90 File Reference

Improve initial estimates from LevelingSolver.f90.

### Functions/Subroutines

- subroutine PostLevelingSolver

### 5.64.1 Detailed Description

Improve initial estimates from LevelingSolver.f90.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**See also**

Thermochimica.f90
LevelingSolver.f90
GetNewAssemblage.f90

Definition in file PostLevelingSolver.f90.

### 5.64.2 Function/Subroutine Documentation

#### 5.64.2.1 subroutine PostLevelingSolver ( )

The purpose of this subroutine is to improve the initial estimates provided by the -Leveling.f90 subroutine. The premise of the Leveling algorithm is to temporarily treat all species and phases as pure stoichiometric phases, which is mathematically equivalent to assigning a unit activity to all species and phases considered in the assemblage. Similar to the Leveling subroutine, the PostLeveling subroutine only considers nElements species in the system; however, the activity of those species is permitted to depart from unity for solution phases. The activity is taken to be equal to the mole fraction and is computed using the estimated number of moles of that species. In some situations, in particular when the number of moles of the elements varies by many orders of magnitude, the PostLeveling subroutine can provide significantly better initial estimates for optimization.

Definition at line 51 of file PostLevelingSolver.f90.

## 5.65 src/shared/ResetThermo.f90 File Reference

Deallocate allocatable variables used by the ModuleThermo.f90, ModulePGESolver.f90 modules.

### Functions/Subroutines

- subroutine ResetThermo

### 5.65.1 Detailed Description

Deallocate allocatable variables used by the ModuleThermo.f90, ModulePGESolver.f90 modules.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

ModuleThermo.f90

Definition in file ResetThermo.f90.

### 5.65.2 Function/Subroutine Documentation

#### 5.65.2.1 subroutine ResetThermo ( )

The purpose of this subroutine is to attempt to gracefully exit Thermochimica. - Allocatable arrays are deallocated and memory is stored for output to external packages.

Definition at line 36 of file ResetThermo.f90.

## 5.66 src/shared/ResetThermoAll.f90 File Reference

Deallocate all allocatable variables.

### Functions/Subroutines

- subroutine ResetThermoAll

### 5.66.1 Detailed Description

Deallocate all allocatable variables.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

ResetThermo.f90
ResetThermoParser.f90

Definition in file ResetThermoAll.f90.

### 5.66.2 Function/Subroutine Documentation

#### 5.66.2.1 subroutine ResetThermoAll ( )

The purpose of this subroutine is to fully destruct all allocatable variables associated with Thermochimica. This includes variables used internally by Thermochimica and by the parser that parses data-files as input.

Definition at line 30 of file ResetThermoAll.f90.

## 5.67 src/shared/ResetThermoParser.f90 File Reference

Deallocate allocatable variables used by the ModuleParseCS.f90 module.

### Functions/Subroutines

- subroutine ResetThermoParser

### 5.67.1 Detailed Description

Deallocate allocatable variables used by the ModuleParseCS.f90 module.

**Author**

M.H.A. Piro

**Date**

Apr. 26, 2012

**See also**

ModuleParseCS.f90

Definition in file ResetThermoParser.f90.

### 5.67.2 Function/Subroutine Documentation

#### 5.67.2.1 subroutine ResetThermoParser ( )

The purpose of this subroutine is to deallocate all allocatable arrays from the ModuleParseCS.f90 module, which are allocated in the ParseCS∗.f90 subroutines and then used in several subroutines by Thermochimica. A value of INFOThermo = 18 is returned if an error has occured during deallocation.

Definition at line 38 of file ResetThermoParser.f90.

## 5.68 src/shared/SortPick.f90 File Reference

Sort a double real vector (this vector is unchanged, the indices of this vector is sorted).

### Functions/Subroutines

- subroutine SortPick (n, dVec, iVec)

### 5.68.1 Detailed Description

Sort a double real vector (this vector is unchanged, the indices of this vector is sorted).

**Author**

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery (modified by M.H.A. Piro)

**See also**

CheckSolnPhaseAdd.f90

Definition in file SortPick.f90.

### 5.68.2 Function/Subroutine Documentation

#### 5.68.2.1 subroutine SortPick ( integer, intent(in) *n,* real(8), dimension(n), intent(in) *dVec,* integer, dimension(n), intent(out) *iVec* )

The purpose of this subroutine is to take a double real vector as input and return an integer vector representing the descending order of the real vector (unchanged). Although this sorting routine is by no means efficient for large problems, "it is meant to be invoked only for the most trivial sorting jobs, say, N $<$ 20."

**Parameters**

| | | |
|---|---|---|
| in | *n* | An integer scalar represening the dimension of dVec. |
| in | *dVec* | A double real vector that is to be sorted in descending order. |
| out | *iVec* | An integer vector representing the index of coefficients of dVec in descending order. |

Definition at line 49 of file SortPick.f90.

## 5.69 src/shared/Thermochimica.f90 File Reference

The main thermochemical solver.

### Functions/Subroutines

- subroutine Thermochimica

### 5.69.1 Detailed Description

The main thermochemical solver.

**Author**

M.H.A. Piro

**Date**

July 17, 2012

**See also**

CheckThermoInput.f90
InitThermo.f90
CheckSystem.f90
CompThermoData.f90
CheckThermoData.f90

LevelingSolver.f90
PostLevelingSolver.f90
GEMSolver.f90

Definition in file Thermochimica.f90.

### 5.69.2 Function/Subroutine Documentation

#### 5.69.2.1 subroutine Thermochimica ( )

Definition at line 429 of file Thermochimica.f90.

## 5.70 src/shared/ThermoDebug.f90 File Reference

Thermochimica debugger.

### Functions/Subroutines

- subroutine ThermoDEBUG

### 5.70.1 Detailed Description

Thermochimica debugger.

**Author**

M.H.A. Piro

**Date**

Apr. 24, 2012

**Parameters**

| | | |
|---|---|---|
| in | *INFO-Thermo* | An integer scalar representing a successful exit or an error. |

Definition in file ThermoDebug.f90.

### 5.70.2 Function/Subroutine Documentation

#### 5.70.2.1 subroutine ThermoDEBUG ( )

The purpose of this subroutine is to print an error message corresponding to a particular value of INFOThermo if a problem is encountered. This subroutine is intended to only be used for debugging purposes.

Definition at line 29 of file ThermoDebug.f90.

## 5.71 src/shared/ThermoOutput.f90 File Reference

This subroutine determines which values are to be provided as output.

### Functions/Subroutines

- subroutine ThermoOutput
- logical function IsSolnPhaseInSys (cSolnPhaseNameOut)
- integer function GetSolnPhaseIndex (cSolnPhaseNameOut)
- logical function IsPureConPhaseInSys (cPureConPhaseNameOut)

### 5.71.1 Detailed Description

This subroutine determines which values are to be provided as output.

**Author**

M.H.A. Piro

**Date**

May 8, 2012

**See also**

Thermochimica.f90

Definition in file ThermoOutput.f90.

### 5.71.2 Function/Subroutine Documentation

#### 5.71.2.1 integer function GetSolnPhaseIndex ( character(∗), intent(in) *cSolnPhaseNameOut* )

Definition at line 221 of file ThermoOutput.f90.

#### 5.71.2.2 logical function IsPureConPhaseInSys ( character(∗), intent(in) *cPureConPhaseNameOut* )

Definition at line 265 of file ThermoOutput.f90.

#### 5.71.2.3 logical function IsSoInPhaseInSys ( character(∗), intent(in) *cSoInPhaseNameOut* )

Definition at line 176 of file ThermoOutput.f90.

#### 5.71.2.4 subroutine ThermoOutput ( )

The purpose of this subroutine is to store particular information that is requested as output. This is done to avoid unnecessary use of memory. Note that it is possible that information is requested that is not in the system. This subroutine first checks to see if the requested information is even in the system and then it checks to see if the particular species or phase is stable at equilibrium.

Definition at line 49 of file ThermoOutput.f90.

## REFERENCES


1. C.W. Bale, P. Chartrand, S.A. Degterov, G. Eriksson, K. Hack, R. Ben Mahfoud, J. Melançon, A.D. Pelton and S. Peterson, "FactSage Thermochemical Software and Databases," CALPHAD, V. 26, N. 2 (2002) 189-228.

2. D. van Heesch, "Doxygen Manual," www.doxygen.org/, last visited October 10, 2012.

3. M.H.A. Piro, "Computation of Thermodynamic Equilibria Pertinent to Nuclear Materials in Multi-Physics Codes," PhD Thesis, Royal Military College of Canada, Kingston, ON, Canada (2011).

4. M.H.A. Piro, S. Simunovic, T.M. Besmann, B.J. Lewis and W.T. Thompson, "The Thermochemistry Library Thermochimica," Journal of Computational Materials Science, 67 (2013) 266-272.

5. International Union of Pure and Applied Chemistry (IUPAC), "IUPAC Gold Book", http://goldbook.iupac.org, last visited August 2, 2012.

6. M.W. Zemansky, R.H. Dittman, "Heat and Thermodynamics," 6th Ed., McGraw-Hill, New York (1981).

7. M. Hillert, "The Compound Energy Formalism", Journal of Alloys and Compounds, 320 (2001) 161-176.

8. M.W. Zemansky, R.H. Dittman, "Heat and Thermodynamics," 6th Ed., McGraw-Hill, New York (1981).

9. M.H.A. Piro and S. Simunovic, "Performance Enhancing Algorithms for Computing Thermodynamic Equilibria," CALPHAD, 39 (2012) 104-110.

10. M.H.A. Piro, T.M. Besmann, S. Simunovic, B.J. Lewis and W.T. Thompson, "Numerical Verification of Equilibrium Thermodynamic Computations in Nuclear Fuel Performance Codes," Journal of Nuclear Materials, 414 (2011) 399-407.

11. M.H.A. Piro and B. Sundman, to be published.

12. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes – The Art of Scientific Computing," Cambridge University Press, New York (1986).

13. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, "LAPACK User's Guide," Society for Industrial and Applied Mathematics, Philadelphia (1992).