

Thermochimica User Manual

M.H.A Piro, S. Simunovic, T.M. Besmann, M. Poschmann

November 30, 2022

1 Introduction

1.1 Overview

The software library Thermochimica [1] has been developed to determine a unique combination of phases and their compositions at thermochemical equilibrium. The development of Thermochimica has been 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, and 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, which is maintained on an open GitHub repository <https://github.com/ORNL-CEES/thermochimica>.

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. For stand-alone operations, Thermochimica can be operated in three modes:

1. Writing and compiling Fortran driver files (like those in the test directory).
2. Using input scripts (like those in the inputs directory).
3. Using the GUIs (can be run via scripts in the scripts directory).

The main purpose of the software is to be directly coupled to a multi-physics code to provide material properties and boundary conditions for various physical phenomena.

Temperature, hydrostatic pressure, the mass of each chemical element, and the units of the foregoing variables 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 [2] or through the

auxiliary code CSFAP.

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

- determination of which phases are stable at equilibrium,
- the mass of solution species and phases at equilibrium,
- mole fractions of solution phase species,
- site fractions of constituents in multi-sublattice phases,
- pair fractions of nearest neighbours in phase represented by the modified quasi-chemical model,
- thermochemical activities (which are related to partial pressures for gaseous species),
- chemical potentials of solution species, phases, and system components,
- integral Gibbs energy (referenced relative to standard state),
- heat capacity, entropy, and enthalpy of the equilibrium system.

Additional variables that are not listed above could be made available with a modest level of programming effort. All input/output operations are performed via Fortran modules, specifically ModuleThermo and ModuleThermoIO. An example is given in §3.6 of how one can make use of the aforementioned variables for the carbon-oxygen system. Details for all variables used by all modules are documented in a dOxygen report, which is provided in the Appendix. dOxygen [3] 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 Capabilities, Limitations, and Assumptions

All calculations performed by Thermochemica 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 refueling¹. 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 fuel². 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 Thermochemica and the data-file parser are summarized below³:

- Systems comprised of various combinations of pure stoichiometric phases, and ideal and non-ideal solution phases,

¹The simulation of transient behaviour 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.

²For the case of simulating nuclear fuel behaviour 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.

³Thermochemica 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 [4].

- Non-ideal regular solution phases based on the “QKTO”, “RKMP”, “SUBL”, and “SUBG” identification tags,
- 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
- 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.
- Some non-ideal models available in FactSage are not currently available in Thermochemica,
- Real gases cannot be considered, and
- Aqueous solution phases cannot be used in the current version⁴.

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 internally 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 Thermochemica are numerically correct; however, the results produced by the software depend on the thermodynamic data and models specified by the user. The user should be aware of limitations in using certain thermodynamic datasets and models in addition to the limitation of performing *equilibrium* calculations.

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 [4] and details of the numerical methods employed by Thermochemica are given by Piro *et al.* [1, 5, 6].

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

2.1 Glossary

Table 1 summarizes many thermodynamic terms that are used in this document and in comments located in the source code.

Table 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 stoichiometric phase that is stable.
Activity Coefficient	γ_i	The activity coefficient accounts for the departure of a substance from ideal behaviour 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	μ_i	A measure of the effect on the change of the Gibbs energy of the system by the introduction of substance i . The chemical potential is formally defined as $\mu_i = (\partial G/\partial n_i)_{T,P,n_{k \neq i}}$.
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 species 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” [7].
Constituent		A constituent of a solution phase refers to a particular species in a particular phase and/or a particular sublattice. For example, “H ₂ O” is a species, whereas “H ₂ O _(g) ” refers to H ₂ O in the gaseous phase and “H ₂ O _(l) ” refers to H ₂ O in a liquid phase.
Element		A chemical element as it appears on the Periodic Table of the Elements.
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 [7].
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 $6.02214179 \times 10^{23}$ 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 behaviour 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	$g_{i(\lambda)}^{ex}$	The partial molar excess Gibbs energy of mixing 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 from 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 pure uranium in the solid state (i.e., orthogonal, tetragonal and body centred cubic). Thermochemica considers two different types of phases: stoichiometric 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	ω, Ω	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 ω represents the phase index and Ω represents the number of stable pure condensed phases.
Solution Phase	λ, Λ	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 λ represents the phase index and Λ 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 and/or sublattice. The phase index is represented by the variable i .
Standard Molar Gibbs Energy	$g_{i(\lambda)}^o$	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. Thermochemica 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 [8]. 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 [4].

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

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

where R [J·mol⁻¹·K⁻¹] is the ideal gas constant, T [K] is the absolute temperature, N_{λ} denotes the number of constituents in solution phase λ , and Λ , and Ω represent the total number of solution phases and pure condensed phases in the system, respectively. The mole fraction of constituent i in solution phase λ is $x_{i(\lambda)}$ [unitless] and the number of moles of solution phase λ and pure condensed phase ω are denoted by n_{λ} and n_{ω} [mol] respectively. Finally, $\tilde{\mu}_i$ and $\tilde{\mu}_{\omega}$ [unitless] represent the dimensionless chemical potential of constituent i in solution phase λ and pure condensed phase ω , respectively. The diacritic symbol \sim 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, $\tilde{g}_{i(\lambda)}^o$ [unitless], and the entropic contribution due to mixing as a function of its mole fraction, such that [8]

$$\tilde{\mu}_i = \tilde{g}_{i(\lambda)}^o + \ln(x_{i(\lambda)}) \quad (2)$$

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

$$\tilde{\mu}_i = \tilde{g}_{i(\lambda)}^o + \tilde{g}_{i(\lambda)}^{ex} + \ln(x_{i(\lambda)}) \quad (3)$$

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

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

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

$$F = C - \Phi + 2 + \Xi \quad (5)$$

where F represents the number of degrees of freedom, C is the number of system components⁵, Φ (i.e., $\Phi = \Lambda + \Omega$) represents the number of phases currently predicted to be stable in the system, and Ξ represents any additional constraints (e.g., ionic charge). 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 [8, 9]

$$\tilde{\mu}_i = \sum_{j=1}^C \nu_{i,j} \tilde{\Gamma}_j \quad (6)$$

The above linear equality corresponds to species and phases that are stable at thermodynamic equilibrium. Non-stable phases are defined by the “driving force” or “tangent plane distance function”, which is defined as [5]:

⁵Note that a system component may represent a pure chemical element, a fixed combination of chemical elements, or an electron (i.e., for ionic phases).

$$\pi_\lambda = \min_x \sum_{i=1}^{N_\lambda} x_{i(\lambda)} \left(\mu_{i(\lambda)} - \sum_{j=1}^E \nu_{i,j} \Gamma_j \right) \quad (7)$$

which is positive for meta-stable phases and a negative value is used to indicate that the system is not at thermodynamic equilibrium.

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 Gibbs’ criteria, 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 driving force is positive and non-zero for all phases believed to not be stable.

The interested reader is referred to the literature on specific details on determining initial estimates for Thermochemica [10], a general overview of numerical techniques used by the solver [4, 1], and numerical methods in ensuring that the necessary [11] and sufficient conditions [5] have been satisfied.

3 Code Development

3.1 Overview

Thermochemica 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 Thermochemica 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

Thermochemica 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 internally 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 $295 \text{ K} \leq T \leq 6000 \text{ K}$ and the acceptable range for hydrostatic pressure is $10^{-6} \text{ atm} \leq P \leq 10^6 \text{ atm}$. 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

Listing 1 provides an example Fortran wrapper to call the data-file parser for ChemSage-style .dat files (`ParseCSDDataFile.f90`), Thermochemica (`Thermochemica.f90`), the destructor (`ResetThermoAll.f90`, optional)

and the debugger (ThermoDebug.f90, optional). This example wrapper is provided as a Fortran program with the software in the *test* directory 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 Thermochemica. 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 carbon-oxygen system at 1000 K and 1 atm with 2 mol C and 1 mol O.

```
!
!
! Purpose:
! =====
!
! The purpose of this program is to provide an example Application
! Programming Interface (API) for Thermochemica to simulate the interaction
! with another code. The module "ModuleThermoIO" is intended to be used by
! another code that calls Thermochemica for input/output operations.
!
```

```
program thermo
```

```
    USE ModuleThermoIO
```

```
    implicit none
```

```
    ! Specify units:
```

```
    cInputUnitTemperature = 'K'
    cInputUnitPressure    = 'atm'
    cInputUnitMass        = 'moles'
    cThermoFileName       = DATA_DIRECTORY // 'C-O.dat'
```

```
    ! Specify values:
```

```
    dPressure             = 1D0
    dTemperature          = 1000D0
    dElementMass(6)       = 2D0           ! C
    dElementMass(8)       = 1D0           ! O
```

```
    ! Specify output and debug modes:
```

```
    iPrintResultsMode     = 2
    lDebugMode            = .FALSE.
```

```
    ! Parse the ChemSage data-file:
```

```
    call ParseCSDataFile(cThermoFileName)
```

```
    ! Call Thermochemica:
```

```
    if (INFOThermo == 0)      call Thermochemica
```

```
    ! Perform post-processing of results:
```

```
    if (iPrintResultsMode > 0) call PrintResults
```

```
    ! Destruct everything:
```

```
    if (INFOThermo == 0)      call ResetThermoAll
```



```

! Call the debugger:
call ThermoDebug

end program thermo

```

Listing 1: A sample Fortran wrapper is provided that calls the ChemSage data-file parser, Thermochemica, the destructor (optional) and the debugger (optional).

As described in the Appendix, the variable `dElementMass` is a double real vector indexed 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 Listing 1 represents carbon and oxygen, which have atomic numbers 6 and 8 respectively. Therefore, the example system above has 2 mole of carbon and 1 moles of oxygen.

3.4 C++ Application Programming Interface

The C++ API provided with Thermochemica is given in Listing 2 below.

```

#include <string>
#define FORTRAN_CALL(name) name##_

extern "C"
{
    void FORTRAN_CALL(setthermofilename)(const char *, int *);
    void SetThermoFilename(const char *);
    void FORTRAN_CALL(setunittemperature)(const char *);
    void SetUnitTemperature(const char *);
    void FORTRAN_CALL(setunitpressure)(const char *);
    void SetUnitPressure(const char *);
    void FORTRAN_CALL(setunitmass)(const char *);
    void SetUnitMass(const char *);
    void FORTRAN_CALL(setunits)(const char *, const char *, const char *);
    void SetUnits(const char *, const char *, const char *);
    void FORTRAN_CALL(setstandardunits)();

    void FORTRAN_CALL(getmolfraction)(int *, double *, int *);
    void FORTRAN_CALL(getchemicalpotential)(int *, double *, int *);
    void FORTRAN_CALL(getelementpotential)(int *, double *, int *);

    void FORTRAN_CALL(getoutputchempot)(char *, double *, int *);
    void GetOutputChemPot(char *, double *, int *);
    void FORTRAN_CALL(getoutputsolnspecies)(const char *, int *, const char *,
        int *, double *, double *, int *);
    void GetOutputSolnSpecies(const char *, char *, double *, double *, int *);
    void FORTRAN_CALL(getoutputmolnspecies)(const char *, int *, double *, double
        *, int *);
    void GetOutputMolSpecies(const char *, double *, double *, int *);
    void FORTRAN_CALL(getoutputmolnspeciesphase)(const char *, int *, const char
        *, int *, double *, int *);
    void GetOutputMolSpeciesPhase(const char *, const char *, double *, int *);
    void FORTRAN_CALL(getelementmolesinphase)(const char *, int *, const char *,
        int *, double *, int *);
    void GetElementMolesInPhase(const char *, const char *, double *, int *);
    void FORTRAN_CALL(getelementmolefractioninphase)(const char *, int *, const
        char *, int *, double *, int *);
}

```

```

void GetElementMoleFractionInPhase(const char *, const char *, double *, int
    *);

void FORTRAN_CALL(getsolnphasemol)(const char *, double *, int *);
void GetSolnPhaseMol(const char *, double *, int *);
void FORTRAN_CALL(getpureconphasemol)(const char *, double *, int *);
void GetPureConPhaseMol(const char *, double *, int *);
void FORTRAN_CALL(getphaseindex)(const char *, int *, int *, int *);
void GetPhaseIndex(const char *, int *, int *);

void FORTRAN_CALL(getoutputsitefraction)(const char *, int *, int *, int *,
    double *, int *);
void GetOutputSiteFraction(const char *, int *, int *, double *, int *);
void FORTRAN_CALL(getsublsitemol)(const char *, int *, int *, int *, double
    *, int *);
void GetSublSiteMol(const char *, int *, int *, double *, int *);

void FORTRAN_CALL(setprintresultsmode)(int *);

void FORTRAN_CALL(setelementmass)(int *, double *);
void FORTRAN_CALL(presetelementmass)(int *, double *);
void FORTRAN_CALL(settemperaturepressure)(double *, double *);
void FORTRAN_CALL(checkinfothermo)(int *);

void FORTRAN_CALL(ssparsecsdatafile)();
void FORTRAN_CALL(thermochimica)();

void FORTRAN_CALL(solphaseparse)(int *, double *);

void FORTRAN_CALL(thermodebug)();

void FORTRAN_CALL(printresults)();
void FORTRAN_CALL(printstate)();

void FORTRAN_CALL(resetinfothermo)();
void FORTRAN_CALL(resetthermo)();
void FORTRAN_CALL(resetthermoall)();

// re-initialization-related functions
void FORTRAN_CALL(savereinitdata)();
void FORTRAN_CALL(getreinitdatasizes)(int *, int *);
void FORTRAN_CALL(getmolesphase)(double *);
void FORTRAN_CALL(getassemblage)(int *);
void FORTRAN_CALL(setreinitrequested)(int *);
void FORTRAN_CALL(resetreinit)();
void FORTRAN_CALL(getallelementpotential)(double *);
void FORTRAN_CALL(getelementfraction)(int *, double *);

unsigned int atomicNumber(const char *);
}

void ConvertToFortran(char *, std::size_t, const char *);

```

```

unsigned int checkTemperature(const std::string & temperature_unit);
unsigned int checkPressure(const std::string & pressure_unit);
unsigned int checkMass(const std::string & mass_unit);

std::string elementName(unsigned int atomic_number);

```

Listing 2: The C++ application programming interface provided with Thermochemica.

3.5 Useful Output Variables

There are several useful variables that are provided as output that may be used by another code. Table 2 summarizes many of these variables organized by their corresponding module. The Hungarian variable naming convention is adopted by Thermochemica, which 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 2: Summary of useful variables provided as output from Thermochemica.

Module Name	Variable Name (Array Dimension)	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.
	dMolFraction (1:nSpecies)	A double real vector representing the mole fraction of each species in the system data-file.
	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.
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 C-O system from Listing 1, which contains 1 solution phase containing a total of 16 species and pure condensed phases. Table 3 summarizes pertinent variables that are provided by Thermochemica at thermodynamic equilibrium.

Table 3: Sample output data from the example given in Listing 1 in the format of Table 2. All double real variables are cropped to three significant figures.

Module Name	Variable Name (Array Dimension)	Value
ModuleThermo	nElements	2
	nSpecies	16
	nConPhases	1
	nSolnPhases	1
	nSolnPhasesSys	1
	iAssemblage (1:nElements)	13, -1
	iPhase (1:nSpecies)	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, -1, -1
	dMolesPhase (1:nElements)	1.23, 0.77
	dMolesSpecies (1:nSpecies)	4.01×10^{-30} , 1.75×10^{-34} , 3.04×10^{-32} , 1.20×10^{-40} , 1.27×10^{-40} , 2.97×10^{-21} , 4.69×10^{-22} , 1.00×10^{-43} , 0.55, 4.66×10^{-20} , 0.23, 3.55×10^{-14} , 0, 0, 0, 0

	dMolFraction (1:nSpecies)	5.19×10^{-30} , 2.26×10^{-34} , 3.93×10^{-32} , 1.55×10^{-40} , 1.64×10^{-40} , 3.85×10^{-21} , 6.06×10^{-22} , 1.29×10^{-43} , 0.71, 6.02×10^{-20} , 0.29, 4.60×10^{-14} , 0.1, 0.1, 0.1, 0.1
	dStdGibbsEnergy (1:nSpecies)	65.91, 74.42, 67.75, 85.58, 84.00, 9.30, -26.55, -14.37, -38.88, 3.51, -75.70, -49.26, -1.52, -0.82, 100000, 100000
	dChemicalPotential (1:nSpecies)	-1.52, -3.05, -4.57, -6.09, -7.61, -37.70, -75.41, -113.12, -39.23, -40.75, -76.93, -79.98, 0, 0, 0, 0
	dElementPotential (1:nElements)	-37.70, -1.52
	cElementName (1:nElements)	"O", "C"
	cSolnPhaseName (1:nSolnPhasesSys)	"gas_ideal"
	cSpeciesName (1:nSpecies)	"C", "C2", "C3", "C4", "C5", "O", "O2", "O3", "CO", "C2O", "CO2", "C3O2", "C_graphite(s)", "C_Diamond(s2)", "C", "O"
ModuleThermoIO	INFOThermo	0
ModuleGEMSolver	dPartialExcessGibbs (1:nSpecies)	0, 0, 0, 0, 0, 0, 0, 0, 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 13th species in the system, which corresponds to "C_Graphite(s)" in the **cSpeciesName** vector. Similarly, the second coefficient in **iAssemblage** refers to the 1st solution phase in the system, which corresponds to "gas_ideal" in the **cSolnPhaseName** vector.

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

$$\begin{aligned}
\tilde{\mu}_{C_{graphite(s)}} &= \tilde{g}_{C_{graphite(s)}}^o + \tilde{g}_{C_{graphite(s)}}^{ex} + \ln(x_{C_{graphite(s)}}) \\
\tilde{\mu}_{C_{graphite(s)}} &= -1.52 - 0 + \ln(1) \\
\tilde{\mu}_{C_{graphite(s)}} &= -1.52
\end{aligned} \tag{8}$$

which can also be compared to equation (6),

$$\begin{aligned}
\tilde{\mu}_{C_{graphite(s)}} &= \sum_{j=1}^E a_{C_{graphite(s)},j} \tilde{\Gamma}_j \\
\tilde{\mu}_{C_{graphite(s)}} &= (1)(-1.52) \\
\tilde{\mu}_{C_{graphite(s)}} &= -1.52
\end{aligned} \tag{9}$$

Also, the chemical potential of C in the gaseous phase must be equivalent to C in the graphite phase. The chemical potential of C in the graphite phase is computed using Equation 2

$$\begin{aligned}
\tilde{\mu}_{C_{gas_ideal}} &= \tilde{g}_{C_{gas_ideal}}^o + \ln(a_{C_{gas_ideal}}) \\
\tilde{\mu}_{C_{gas_ideal}} &= 65.91 + \ln(5.19 \times 10^{-30}) \\
\tilde{\mu}_{C_{gas_ideal}} &= -1.52
\end{aligned} \tag{10}$$

To reduce memory requirements in a multi-physics code that makes use of Thermochemica, 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 and 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 4 summarizes the errors that may be encountered by `Thermochimica`.

Table 4: Summary of error codes from `INFOThermo`.

Value of <code>INFOThermo</code>	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 <code>cThermoInputUnits</code> 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 <code>ModuleThermo</code> , <code>ModuleThermoIO</code> , and/or <code>ModuleGEMSolver</code> 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 <code>ModuleParseCS</code> module.
19	Failed to deallocate allocatable arrays used in the <code>CheckSystem</code> subroutine.
20	Failed to deallocate allocatable arrays used in the <code>LevelingSolver</code> subroutine.
21	Failed to deallocate allocatable arrays used in the <code>InitGEM</code> subroutine.
22	Failed to deallocate allocatable arrays used in the <code>CompMolSolnPhase</code> subroutine.
23	Failed to deallocate allocatable arrays used in the <code>GEMBroyden</code> subroutine.
24	A NAN was detected in the <code>CompStoichSolnPhase</code> subroutine.
25	A NAN was detected in the <code>CompMolFraction</code> subroutine.
26	Failed to deallocate allocatable arrays used in the <code>CompMolAllSolnPhases</code> subroutine.
27	The <code>CheckQKTOSolnPhase</code> subroutine failed to converge.
28	LAPACK returned an error in the <code>SubMinNewton</code> subroutine.
29	There is an unknown error in the <code>Subminimization</code> subroutine.
30	LAPACK reported an error in <code>InitMolFractionMiscGap</code> .
31	The name of a chemical element in the parsed data-file does not correspond to a known element on the periodic table.
32	An unsupported number of mixing parameters is included in the data-file. <code>Thermochimica</code> is currently capable of handling: binary, ternary and quaternary terms.
33	The number of sublattices per phase is not supported.

35	A coefficient in the iTempVec vector in the CheckPureConPhaseRem subroutine is zero but must be positive.
36	An error occurred in interpreting the parsed data for a SUBL phase.
40	There is an element in a compound that is not in the dat file.
41	Error finding stoichiometry in terms of compounds.
43	An unsupported type of magnetic mixing parameter is included in the data-file. Thermochemica is currently capable of handling: binary terms.
44	An unsupported type of excess mixing term in the SUBI phase is included in the data-file. Please contact the developers and share this data-file.
50	Input file cannot be opened.
51	Reading input file failed.
52	Open (but no close) in input file.
53	Cannot read element number in input file.
54	A tag was not properly read in input parsing.
55	Not all input parameters were supplied in input file.
99	The input element masses are not representable in terms of the available species.
[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.

3.8 Phase Selection

Selecting which phases are included in a calculation can be useful for examining metastable states of a system, calculating specific properties of a phase (such as enthalpy of mixing), and for debugging. Thermochemica offers two methods for specifying which phases are active: exclusions and exceptions. Fundamentally, both modify a list of phases that are excluded from the calculation. Phases may be added to the list of exclusions directly by modifying `cPhasesExcluded`. A corresponding integer variable `nPhasesExcluded` should be set to the number of phases named in `cPhasesExcluded`. For example, to exclude phases named FCC and BCC from a calculation:

```
nPhasesExcluded      = 2
cPhasesExcluded(1) = 'FCC'
cPhasesExcluded(2) = 'BCC'
```

Alternatively, a list of phases can be provided in `cPhasesExcludedExcept`, and all phases not on that list will be added to the phase exclusion list. Similarly, `nPhasesExcludedExcept` should be set to the number of phases named in `cPhasesExcludedExcept`. Note both exclusions and exceptions may be used simultaneously, and that a phase named on both lists **will be excluded**. For example, to include only the phases named FCC and BCC in a calculation:

```
nPhasesExcludedExcept = 2
cPhasesExcludedExcept(1) = 'FCC'
cPhasesExcludedExcept(2) = 'BCC'
```

3.9 Fuzzy Stoichiometry

Occasionally, situations are encountered in which the stoichiometries of the species present in the assemblage result in an under-determined system of equations [13]. This renders it impossible to determine unique values for the element potentials, which can cause serious issues within Thermochemica's convergence routines. Such systems are most commonly encountered in calculations involving salt systems.

As a simple example, consider a calculation involving 1 mole of Li, 1 mole of Be, and 3 moles of F. Above about 400 °C, a single liquid phase should be predicted, containing 1 mole LiF and 1 mole BeF₂. Thus only

two species are present, while there are three elements in the system. While the chemical potentials of LiF and BeF₂ are well-defined, the potentials of Li, Be, and F are not. This becomes an issue when the GEM routine attempts to determine whether another species (perhaps F₂ gas) should be added to the assemblage. While the chemical potential of this species can be calculated, the driving force is not accurate because the element potential of F is not known reliably.

This difficulty can be alleviated in a few ways. Most simply, the input masses can be modified slightly to avoid the under-determined system (for example, set the mass of F to 3.00001 moles). Databases allowing for vacancies also avoid this problem (for example, consider a liquid phase with LiF, BeF₂, and vacancy + F species).

A third option, which avoids changing the system inputs or making substantial changes to the database, is to slightly modify the stoichiometries of the species involved. This can be enabled in Thermochemica by setting `lFuzzyStoich = .TRUE.` in the Thermochemica input. In this case, Thermochemica adds a random value in the range `-dFuzzMag` to `dFuzzMag` to each non-zero entry in the stoichiometry matrix before the calculation is run. These modified stoichiometries are referred to as “fuzzy”. The default value of `dFuzzMag` is 10^{-12} . The stoichiometry matrix is reset at the end of the calculation.

While this solution is imperfect, it does result in more reliable convergence for under-determined systems, and in particular molten salt calculations. It should be noted that using this option can result in small amounts of new phases being included in the assemblage which should not be present. It is recommended that users examine these in post-processing to determine if they are spurious.

3.10 Reinitializing Calculations

In many applications – particularly when coupling Thermochemica to other applications – numerous calls to Thermochemica must be made with similar input conditions (i.e. temperature, pressure, and concentrations of elements). For example, in a finite element simulation the conditions at one integration point are likely similar to those at neighboring integration points, as well as to the conditions at that integration point at the next timestep. As the resulting phase assemblages and species concentrations are likely to be similar, making use of the results of the preceding calculations in place of the leveling solver can yield a better starting point for the Gibbs Energy Minimization (GEM) procedure. This results in cost savings both by bypassing leveling and by reducing the number of GEM iterations required to converge the system.

To accommodate calculation reinitialization it is necessary to store a number of variables. If Thermochemica is coupled to another code, it is likely that these variables will have to be passed out to that code, stored until needed, and then passed back into Thermochemica. For example, in a finite element code that is using data from previous timesteps to reinitialize Thermochemica calculations, after each call at each integration point the reinitialization data must be stored so that it can be re-used during the next timestep when Thermochemica is called for that point again. Between those calls, Thermochemica will be called at many other integration points and thus the reinitialization data cannot simply be expected to be held within Thermochemica.

`iAssemblage`, `dChemicalPotential`, `dMolesPhase`, `dElementPotential`, and `dMolFraction`, as well as an integer array created just for reinitializations, `iElementsUsed`, must be stored to enable reinitialization. `iElementsUsed` lists which elements were present in any non-zero amount in the calculation for which data was saved. If this differs from the current collection of elements, the reinitialization attempt is aborted. The suffix `_Old` is added to each when they are saved, and the variables with this prefix are held in the module `ModuleReinit`. Additional flags to track the status of the reinitialization procedure are also available. These include `lReinitRequested`, which the user must set to `.TRUE.` if reinitializing is desired, `lReinitAvailable`, which is set to `.TRUE.` after reinitialization data is successfully loaded, and `lReinitLoaded`, which is set to `.TRUE.` after a reinitialization is successfully begun.

To make use of reinitialization, after a successful Thermochemica calculation `SaveReinitData` must be called. Then before the following calculation, `lReinitRequested` must be set to `.TRUE.`. If Thermochemica is coupled to another code, the developer must ensure that the `_Old` reinitialization variables are correctly supplied to Thermochemica. Useful subroutines for copying these variables to/from a coupled code are

Table 5: Summary of all make commands.

Make Command	Description
<code>make</code> (or <code>make all</code>)	Compile Thermochemica with all executables in top level of <code>test</code> directory.
<code>make test</code>	Compile Thermochemica and all unit tests (executables will be placed in the <code>bin</code> directory).
<code>make debug</code>	Compile Thermochemica and all unit tests with debug flags enabled (for use with <code>gdb</code> , for example).
<code>make libraries</code>	Compile Thermochemica Fortran and C libraries and install at location specified in <code>PREFIX</code> environment variable.
<code>make dailytest</code>	Compile all unit tests (executables placed in the <code>bin</code> directory).
<code>make doc</code>	Compile dOxygen HTML and \LaTeX documents.
<code>make docHTML</code>	Compile dOxygen HTML documents only.
<code>make doclatex</code>	Compile dOxygen \LaTeX documents only.
<code>make clean</code>	Clean object files.
<code>make cleandoc</code>	Clean all dOxygen files.
<code>make veryclean</code>	Clean all object files, dOxygen files, modules and executables.

available in `CouplingUtils.f90`.

4 Building and Compiling Thermochemica

The first thing that one should do is pull all files from the online repository, which includes source files, a makefile, documentation, etc. The directory structure of Thermochemica is illustrated in Figure 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 number of subdirectories containing all shared source f90 files, and the `test` directory contains all examples, unit tests, and application tests. Tests that are performed for code validation are stored in the `daily` subdirectory. The main directory contains the Makefile, Doxyfile and modules that are built by Fortran. Additionally, a script file (`run_tests`) is provided that runs all daily tests.

Thermochemica is compiled on a regular basis with GNU compilers⁶ on Mac OS-X and Linux operating systems. The only external library that Thermochemica is dependent on is LAPACK [14], which can be downloaded free of charge from the LAPACK web site⁷. 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 Thermochemica, go to the Thermochemica directory in the Terminal and type `make`. By default, a sample executable called `Thermo` is created. Table 5 summarizes the commands that can be used from the provided Makefile.

To ensure that Thermochemica 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 `./run_tests` from the Terminal from the Thermochemica directory. Thermochemica will return the following for a successful test:

```
TestThermo01: PASS
```

and the following for a failure:

⁶The most recent version of this compiler used is gfortran 11.3.0.

⁷The LAPACK linear algebra library can be downloaded from netlib.org.

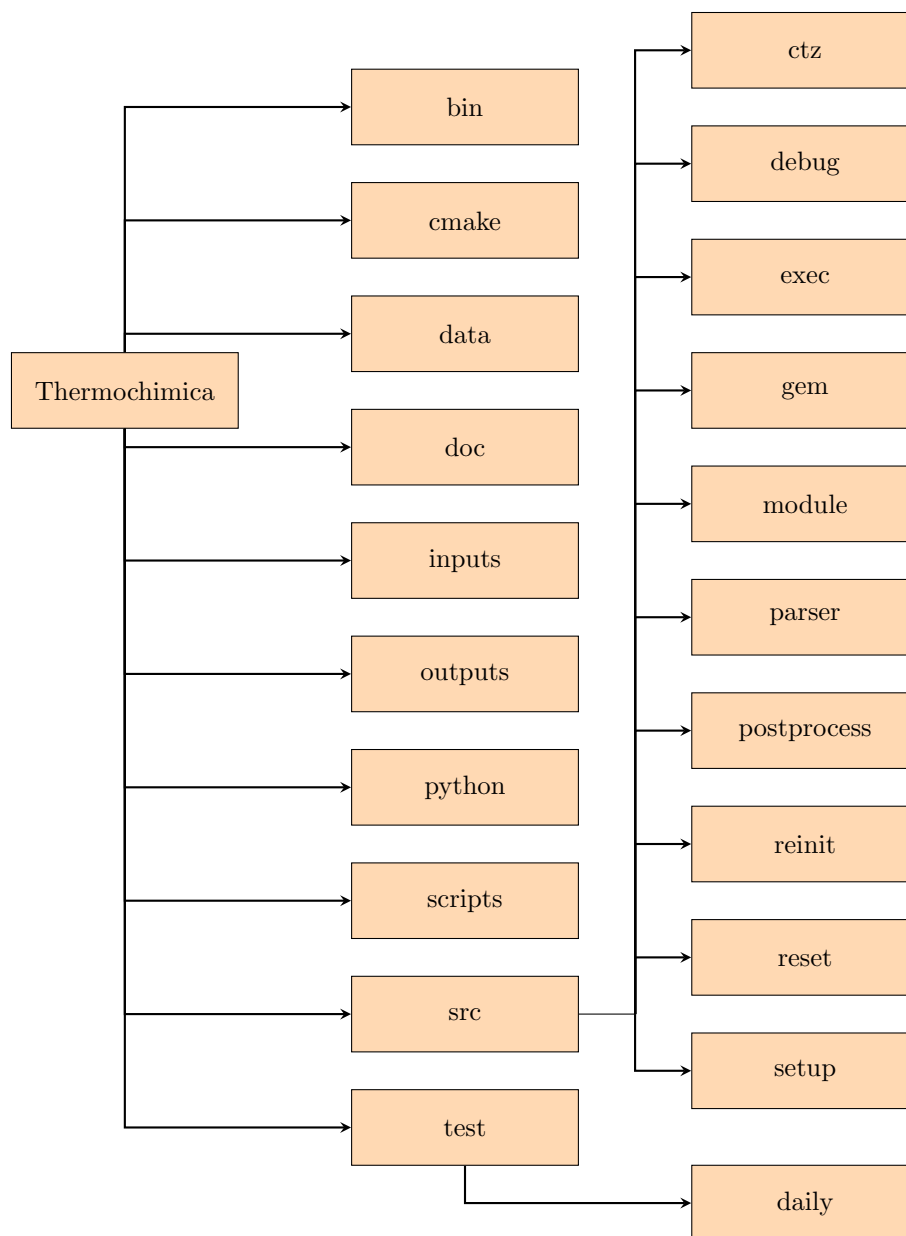


Figure 1: The data structure for Thermochemica.

TestThermo01: FAIL <---

In the event that any tests fail, please report an issue on Thermochemica’s [GitHub page](#) with the data-file that was used and a detailed description of the circumstances of the error. Thermochemica is automatically tested every time an update is made, and both the current status and past test results can be checked by visiting Thermochemica’s [GitHub Actions page](#).

5 Running Stand-Alone Thermochemica Calculations

Now that Thermochemica is installed and all the tests have passed, you may wish to use Thermochemica as a stand-alone application to perform some equilibrium calculations. This can be done via Fortran driver files,

Table 6: Summary of keywords for input files. Underlined keywords are mandatory for `InputScriptMode` but will be ignored by `RunCalculationList`.

Keyword	Type	Description
<u>temperature</u>	Float	System temperature.
<u>pressure</u>	Float	System pressure.
<u>mass(atomic number)</u>	Float	Mass of element given by atomic number argument. (One line per element).
<u>data file</u>	String	Path to data file.
temperature unit	String	Specify temperature unit.
pressure unit	String	Specify pressure unit.
mass unit	String	Specify mass unit.
print mode	Integer	Choose print mode, 0 for no print, 2 for standard print.
debug mode	Logical	Enables extended debug output.
write json	Logical	Enables JSON dump.
reinit	Logical	Enables reinitialization mode.
heat capacity	Logical	Enables heat capacity, entropy, and enthalpy calculation.

input scripts, and graphical user interfaces.

5.1 Fortran driver files

The example Fortran API provided in §3.3 is a good starting point for custom stand-alone calculations. As in that example, you must set the units for temperature (`cUnitTemperature`: K, C, F, and R are accepted), pressure (`cUnitPressure`: atm, psi, bar, Pa, kPa accepted), and mass (`cUnitMass`: mass fraction, kilograms, grams, pounds, mole fraction, atom fraction, atoms, moles, and gram-atoms accepted). You must also supply a ChemSage style `.dat` file, and specify the path to this file in `cThermoFileName`. Next, specify the pressure (`dPressure`) and temperature (`dTemperature`), then the masses of all elements to be included in the system by using their atomic numbers to index into `dElementMass` (1 through 118 accepted). You may also optionally specify the output verbosity through `iPrintResultsMode`, where 0 is no printing, 1 is standard, and 2 adds details such as the functional norm and the numbers of stable pure and solution phases.

With that setup done, you can now run Thermochemica. Just call the parser, Thermochemica, print, reset, and debug in the same order as in the example file `test/Thermo.F90`. Save your custom file in the `test` directory with any name ending with `.F90`, run `make`, and a corresponding executable will be placed in the `bin` directory.

5.2 Input scripts

Alternatively, a number of sample input scripts are included in the Thermochemica repository, in the `inputs` directory. To run this example, use the `InputScriptMode` executable in the `bin` directory, and supply the path to the example input script as a command-line argument (e.g. `run ./bin/InputScriptMode inputs/trial-input.ti` from the root directory). To run your own calculations, make an input script similar to the example given, supplying the same inputs as required when running Thermochemica in the traditional mode. You may then execute the same executable but with the path to your input script as the argument. The input script is flexible with respect to order, spacing, and comments (comment lines start with any special character such as `!`, `#`, `%`, `/`, etc.). There is no need to recompile when using the input script mode!

A second parsed input file option is available, through the `RunCalculationList` executable. This mode accepts most of the same keywords as `InputScriptMode`. However, rather than setting temperature, pressure, and masses as in `InputScriptMode`, a number of elements (`nelements`) and list of atomic numbers (`ielement`) are given. The final keyword in an input to `RunCalculationList` **must be** `ncalc`. Following the `ncalc`

keyword is a list (`ncalc` long) of lines containing (in order): temperature, pressure, and element masses. Calculations for this specified list of conditions are then performed, using the settings specified via supplied keywords. `inputs/trial-runList.ti` is an example input file of this style.

5.3 GUIs

The GUIs for Thermochemica depend on Python(3.8+) and some additional Python packages that can be installed via pip. Installation instructions will be maintained on our GitHub page (<https://github.com/ORNL-CEES/thermochemica#method-3-guis>).

Once installation is complete, the GUIs can be run by calling scripts from the `scripts` directory.

5.3.1 Calculations

To run stand-alone calculations similar to those possible using the Fortran driver files or input scripts methods, run `scripts/thermoGUI.sh`. Through the interface, a single calculation or loop of calculations over temperature, pressure, or mass can be performed. Note that the GUI will automatically generate an input script in the `inputs` directory. You can edit, rename, and reuse this file for future calculations following the instructions in §5.2.

5.3.2 Phase Diagrams

To produce binary phase diagrams, run `scripts/phaseDiagramGUI.sh`. Note that most parameters for phase diagram generation have default values, so do not need to be entered. Under default settings, the initial phase diagram generated will likely be quite coarse and missing features. Users can improve the quality of the diagram by using the functions `Auto Refine` and `Auto Smoothen`. These will add further data points at automatically-selected locations on the phase diagram. For manual selection of these points, the `Refine` function can be employed.

Calculated phase diagram data can be saved using the `Export Diagram Data`. This saves the present state of the diagram. Saved diagrams can be loaded using `Load Diagram`, which allows phase diagrams to be overlaid.

As a user interacts with the phase diagram GUI, their actions are automatically saved in a macro (in memory). To write this macro to a file, use the `Export Macro` function. To clear the list of operations in the macro saved in memory, use `Clear Macro`. To run a saved macro, use `Run Macro`. Saving a macro allows a user to quickly recover the program state at the time of saving. Parameters and function call order can be edited in the macro file. The saved macro will be written to and read from `python/macroPhaseDiagram.py` by default. These macros are actually fully-functional python programs, and thus phase diagrams can be generated from them without using the GUI by running a macro directly with python.

6 Conclusions

A new equilibrium thermochemistry library called Thermochemica 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 Thermochemica 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 Thermochemica. Finally, specific programming details of each variable and subroutine are described in a dOxygen-generated documentation.

References

- [1] M.H.A. Piro, S. Simunovic, T.M. Besmann, B.J. Lewis, and W. T. Thompson. The thermochemistry library Thermochimica. *Computational Materials Science*, 67:266–272, 2013.
- [2] C.W. Bale, P. Chartrand, S.A. Degterov, G. Eriksson, K. Hack, R. Ben Mahfoud, J. Melançon, A.D. Pelton, and S Petersen. FactSage thermochemical software and databases. *Calphad*, 26(2):189–228, 2002.
- [3] D. van Heesch. Main page. <http://www.doxygen.nl/>.
- [4] M.H.A. Piro. *Computation of thermodynamic equilibria pertinent to nuclear materials in multi-physics codes*. Royal Military College of Canada (Canada), 2011.
- [5] M.H.A. Piro and S. Simunovic. Global optimization algorithms to compute thermodynamic equilibria in large complex systems with performance considerations. *Computational Materials Science*, 118:87 – 96, 2016.
- [6] M.H.A. Piro. Updating the estimated assemblage of stable phases in a gibbs energy minimizer. *Calphad*, 58:115 – 121, 2017.
- [7] M. Nic, J. Jirat, and B. Kosata. IUPAC gold book. <http://goldbook.iupac.org/>.
- [8] M. W. Zemansky and R. H. Dittman. *Heat and thermodynamics. An intermediate textbook*. 6.ed. McGraw-Hill, 1981.
- [9] M. Hillert. The compound energy formalism. *Journal of Alloys and Compounds*, 320(2):161–176, 2001.
- [10] M.H.A. Piro and S. Simunovic. Performance enhancing algorithms for computing thermodynamic equilibria. *Calphad: Computer Coupling of Phase Diagrams and Thermochemistry*, 39:104–110, 2012.
- [11] M.H.A Piro, T. M. Besmann, S. Simunovic, B.J. Lewis, and WT Thompson. Numerical verification of equilibrium thermodynamic computations in nuclear fuel performance codes. *Journal of Nuclear Materials*, 414(3):399–407, 2011.
- [12] W.T. Vetterling. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 1988.
- [13] MHA Piro, M Poschmann, and P Bajpai. On the interpretation of chemical potentials computed from equilibrium thermodynamic codes: Applications to molten salts. *Journal of Nuclear Materials*, 526:151756, 2019.
- [14] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users’ guide*, volume 9. Siam, 1999.