General Thermodynamic Package (GTP)

Basic description and documentation, for OC version 7

Bo Sundman, September 21, 2022

This is a preliminary description of the data structure and subroutines in the thermodynamic model package GTP. A background to this initiative to develop a free thermodynamic software can be found in Sundman et al. [16].

This documentation is neither complete nor up to date with the most recent version. The whole structure (and depending subroutines) will be revised when all major parts have been implemented and tested.

The OC software is written in the new Fortran standard and the main reason to use Fortran is that it is specially designed for numerical calculations and there are many numerical libraries available. But it is also due to the age of the developer. Fortran is useful for parallel processing and thermodynamic calculations are needed in many simulation software for materials processee. Extensive use is made of the datatyping available in the new Fortran.

- First version of this documentation in 2015 for OC version 3
- Major update 2018 for OC version 5
- Minor update 2019 for OC version 6
- Major update 2022 for OC version 7

Documentation updating software

One very difficult thing with software that is alive and changing is to update the documentation. So my feeling for documentation is that once a software is fully documented it is dead as it is almost impossible to update source code and documentation in parallel.

In an attempt to callenge this feeling I have written a simple documentation update software which extracts critical parts in the source code identified as "verbatim" parts. In the documentation there are text with additional explanations of the verbatim parts and most verbatim sections are in a separate LaTeX subsection. A small software extract these from the source code and writes them sequentially on a separate file. Each verbatim section normally describes a subroutine or function declaration or some global data declaration. The developer can the compare, manually or with some additional software, if there are new verbatim sections or changes in an old verbatim section and update the documentation manually. New verbatim sections normally require some new explanations. Sometimes the order of subroutines in the source code may need rearrangements to make the documentation easier to follow.

A verbatim section which has changed place in the source code also requires editing of the existing documentation. This is a safe method to find what has changed in the source but it is a quite task for the developer to add new verbatim sections in the source code.

The verbatim sections in the source code is enclosed by

```
!\begin{verbatim}
critical part of the software for example data or a function declaration
!\end{verbatim}
```

The "`!\begin{verbatim}`" and "`!\end{verbatim}`" must be left justified on the line.

The verbatim parts can be inserted anywhere in the source code, typically they enclose important sections like data structure definitions and declarations of subroutines and functions. The documentation software detect new or changed verbatim sections as well as verbatim sections that has disappeared and writes a new LaTeX documentation file which require editing to describe the changes and to add crossreferences. Using this software I hope to keep the software documented also during the development stage. This document is an example of using this software.

The subroutines are documented in the order they appear in the source code. That order is not always logical and sometimes a major restructuring must be made.

A table with the subroutines can be generated for this documentation by adding a line

```
!\addtotable <name of function or subroutine>
```

at each subroutine or function. They are listed in Appendix F

Contents

1 Introduction

In a model for a thermodynamic system one has elements, species (combination of elements with fixed stoichiometric ratios and a possible charge) and phases. The thermodynamic properties of the system is described by the Gibbs energy for each phase as a function of the temperature, T, pressure, P and the fraction of different constituents of the phase, the constituents of a phase are a subset of the species.

At equilibrium at constant T, P and composition the Gibbs energy of a system is at a minimum. With suitable Lagrange transformations of the Gibbs energy one can find the equilibrium of a system for all other sets of conditions (except when a phase has a miscibility gap in the volume such as H_2O vapour/liquid equilibria). From the Gibbs energy one can obtain many important properties of the system like the set of stable phases and their constitution, the chemical potentials of the components, the heat content, heat capacity, volume, thermal expansion and much more. It is also possible to calculate phase diagrams and the models for the phases can be used to extrapolate to non-equilibrium states useful in simulations.

Some efforts has been spent on modeling special physical properties like ferro-magnetism using additional internal variables like the Curie Temperature and how elastic constants (which are not constant) can be included in the modeling.

The method used for storing composition dependent properties of the Gibbs energy parameters is suitable also for other properties that depend on the phase like mobilities, resistivities, viscosities etc and there are provisions to include such data in the software.

The GTP model package is used more application oriented software to calculate equilibria in multicomponent systems and property and phase diagrams as explained in separate documents and papers, most recently in [5].

1.1 New features in version 7 and some earlier versions

This is a full update of the documentation for the release of version 7. There was no documentation for version 6 but during the last years several papers have been published about the use of OC [3, 4, 6, 7, 8, 9, 10]. In this relase a major effort has been made to implement the Modified Quasichemical Model with Quadruplet Approximation (MQMQA) model proposed by Pelton et al. [21] see Appendix E, and this is still onging.

Version 5 of OC was made available early 2018 at the OC homepage [1] but there was no update of the documentation. Frequent updates of the code has been made at github [2], But this is the first update of the documentation since version 5.045. I hope to be able to update the other parts and the User Guide later this year.

OC is shaping up fairly well and there are a few serious users, it is interesting to note that there are several interfaces to OC available on github, even for Python. But OC is now so large it is difficult even for me to find the correct place to update the code and often changes has to be made in many different files. This means documentation becomes even more important! I hope to release version 7 of OC together with a complete documentation update during 2022.

There has been quite a lot of changes in the model package. Recently the UNIQUAC model for fluids was added, see section 8.16.9. Several modifications for the new unary models as the Einstein model, section 2.3.5.1, for the low T heat capacity and the two-state model for amorphous/liquid models, section 2.3.5.2 and the Equi-Entropy Criterion (EEC) [6].

The EEC method to prevent solids to become stable above their melting T is part of the minimizing package as it is related to the equilibrium calculation, not a part of the description of the phases. However, the equi-entropy criterion is activated by setting a positive value in the global datastructure described in section 4.1.

It is now possible to save all workspaces, except results from STEP and MAP commands, on an unformatted Fortran file. This makes it possible to save the current state of a calculation on a file and later retrieve all the data and results from this file. This is currently very useful for simulations and for assessments because all calculated results and all experimenal data and optimizing coefficents can be saved. In a later step saving the results of a STEP or MAP command will be possible.

1.1.1 Features in version 4

Version 4 of OC was released early 2017. A prerelease was available at the opencalphad repository at github [2]. There has been several improvement in various parts. The numeric routines now use LAPACK and BLAS and if a user has a tailored version of these for his computer there may a gain in speed about 5-10%. To simplify the installation there is a subset of the necessary LAPACK and BLAS routines included in this distribution and even these improved the performance with about 20% compared to the previous numeric routines. Other features added are the possibility to use P as variable and V as condition and expressions using amount of moles of components. It is also possible to define other components than the elements and one can calculate and plot isothermal phase diagrams.

There was no new thermodynamic models implemented in version 4. For the equilibrium calculations a paper has been published [15], for more details see the documentation of the equilibrium module, For mapping some improvements have been in the step control but the mapping is very sensitive to the start point. There has been some minor changes in the graphics interface to GNUPLOT. There has been great interest using OC for application software and another paper was published recently [13] together with several users, there will hopefully be a more extensive documentation of the OC Application Software Interface (OCASI) with this release. The possibility to use OC with C++ has been greatly simplified with the OC-isoC interface.

The possibility to save and read the whole datastructure for an equilibrium, mapping or assessment on a file has not been possible to achieve. That will be next release.

1.1.2 Features in version 3 and earlier

The release of version 3 of OC is available at the opencalphad webpage [1] since February 2016. It included several improvements in the already existing software but most important a new assessment procedure to fit model parameters to experimental and theoretical data. This has added a few new data structures and commands to the GTP package. Version 3 can also be used for parallel calculations of several equilibria using the OpenMP package. A significant increase in speed has been noted.

Version 2 of OC was released in February 2015 and the main new feature of this version was the step and map procedures which are documented separately. It also included derivatives of state variables, "dot derivatives", and some improvements of the minimization procedure and main change in the GTP module was implementing the ionic liquid model. This did not really change much in this documentation, the code for handling the variable ratio of sublattices was been integrated in the existing subroutines.

1.2 Features to be added in future versions

A software is never finished but each version has to be tested and debugged before too many new features can be added. One important missing feature is the possibility to save/read unformatted data from STEP and MAP calculations. In OC it is possible to save what in Thermo-Calc is known as PARROT files for assessments.

My main interest in this software is to develop and implement thermodynamic models but a model is not interesting unless it can be used in multicomponent calculations. There are a still models to be added, in particular to handle SRO. As OC is a free software it is possible for any interested scientist to contribute, it is one of the reasons I am writing this documentation.

1.3 The Gibbs energy function

The GTP model package provides the possibility to calculate the Gibbs energy, G, for many phases for any value of T, P and their constitution using the models implemented. In addition OC calculates the first and second derivatives of G with respect to T, P and the constitution. These values can be used directly or within a minimization package [15, 14] developed in parallel, to handle equilibrium calculations in multicomponent system with several phases for various external conditions. The basic formula for the Gibbs energy of a phase α, independent of the model, is taken from the book by Lukas et al. [30].

$$^{\alpha}G_M^{\alpha} = {}^{\rm srf}G_M^{\alpha} - T\,{}^{\rm cfg}S_M^{\alpha} + {}^{E}G_M^{\alpha} + {}^{\rm phys}G_M^{\alpha} \qquad (1)$$

where ${}^{\rm srf}G_M^{\alpha}$ is the surface of reference containing parameters for the endmembers, ${}^{\rm cfg}S_M^{\alpha}$ is the configurational entropy, ${}^{E}G_M^{\alpha}$ is the excess Gibbs energy which is described by interaction parameters and ${}^{\text{phys}}G^{\alpha}_{M}$ is the contribution to the Gibbs energy due to some physical properties, like ferromagnetism, that are modeled separately using additional parameters. The term **endmember** is explained in section 2.3.1.3.

The Gibbs energy is modeled for a formula unit of the phase, the formula unit is defined by the crystallography, and the amount of the components for a formula unit of a phase can vary because the constituents can be for example molecules in a gas phase or vacancies occupying vacant lattice sites. The composition dependence of the modeled Gibbs energy depend on the constituent fractions, denoted y_i, possibly with a sublattice index. The amount of moles of component A per formula unit of the phase α, M^{α}_{A} can be calculated using the formula:

$$M^{\alpha}_{\mathrm{A}} = \sum_s a_s \sum_i b_{i\mathrm{A}} y^{\alpha}_{si} \tag{2}$$

$$x^{\alpha}_{\mathrm{A}} = \frac{M^{\alpha}_{\mathrm{A}}}{\sum_{\mathrm{B}} M^{\alpha}_{\mathrm{B}}} \tag{3}$$

where a^{α}_{s} is the number of sites on sublattice s, $b_{\mathrm{A}j}$ is the stoichiometric ratio of element A in species j and y^{α}_{js} is the constituent fraction of species j on sublattice s. The summation over B is for all components. The sum of site fractions in a sublattice s is unity:

$$\sum_i y_{si} = 1 \tag{4}$$

The terms and factors used in this equation are explained below and for further details please read the book by Lukas et al. [30].

1.4 State functions and variables related to the Gibbs energy

The reader of this documentation is assumed to be familiar with most of the thermodynamic relations but a short summary is given here using what is called Euler integrals where N_i denotes the amount in moles of component i in Table 1.

Table 1: Thermodynamic state functions

Name	Symbol	Relation	Expression
Internal energy	U	$U(S, V, N_i)$	$TS - PV + \sum_i \mu_i N_i$
Enthalpy	H	$H(S, P, N_i) = U + PV$	$TS + \sum_i \mu_i N_i$
Helmhotz energy	A	$A(T, V, N_i) = U - TS$	$PV + \sum_i \mu_i N_i$
Gibbs energy	G	$G(T, P, N_i) = U - TS + PV$	$\sum_i \mu_i N_i$
Grand potential	Ψ	$\Psi(T, V, \mu_i) = U - TS - \sum_i \mu_i N_i$	PV

Note that when learning thermodynamics a system with a single element is used and thus the sum of chemical potentials and amounts of components usually ignored but the properties depend strongly on the amounts of the different elements.

The state functions in Table 1 are normally expressed as differentitals, for example $dU = TdS - PdV + \sum_i \mu_i dN_i$ to emphasize the independent variables. Note that all state functions have at least one extensive variable. The state function without any extensive variable is the Gibbs-Duheim relation which is always zero.

When playing around with partial derivatives of the thermodynamic functions and variables, for example using the Maxwell relations, the thermodynamic square in Table 2 can be useful. The 4 thermodynamic functions, U, H, A and G have as independent variables the two state variables to the left and right or above and below. All of them depend on the sum of the product of the chemical potentials and the amounts of the components.

Table 2: The thermodynamic square, the thermodynamic functions are in bold.

-S	**U**	V
H		**A**
-P	**G**	T

The Maxwell relations, for example $(\frac{\partial\ S}{\partial\ P})_T = \ -(\frac{\partial\ V}{\partial\ T})_P$ or $(\frac{\partial\ P}{\partial\ T})_V = (\frac{\partial\ S}{\partial\ V})_T$ are obtained by relating the properties in the cornes in Table 2.

The state variables derived as partial derivatives of the Gibbs energy $G(T, P, N_i)$ are listed in Table 3.

Table 3: Properties derived from the Gibbs energy

Name	Symbol	Expression
Entropy	S	$-(\frac{\partial G}{\partial T})_{P,N_i}$
Enthalpy	H	$H = G - TS$
Volume	V	$(\frac{\partial G}{\partial P})_{T,N_i}$
Chemical potential of i	μ_i	$(\frac{\partial G}{\partial N_i})_{T,P,N_{k\neq i}}$
Heat capacity	C_P	$-(\frac{\partial^2 G}{\partial T^2})_{P,N_i}$
Thermal expansion	α_T	$\frac{1}{V}(\frac{\partial^2 G}{\partial T \partial P})_{N_i}$
Bulk modulus	β_T	$-\frac{1}{V}(\frac{\partial^2 G}{\partial P^2})_{T,N_i}$

Note that the chemical potential of element i is defined by varying the total amount of the element. When using composition variables such as mole or site fractions related by a $\sum_i\ x_i = 1$, one must include the variation of other fraction variables as explained in the section 1.4.1.

1.4.1 The chemical potential using mole or site fractions

The composition of a phase is frequently models with constituents which are not the same as the elements. For a phase with a CEF model, see section 2.3, with several sublattices and different sets of constituent fractions, $y_{s,i}$ on each sublattice the chemical potential of an **endmember**, I (see section 2.3.1.3), for a phase α is:

$$\mu_I = G_M^\alpha + \sum_s \left(\frac{\partial G_M^\alpha}{\partial y_{s,i}}\right)_{T,P,y_{t,j\neq s,i}} - \sum_s \sum_j y_{s,j} \left(\frac{\partial G_M^\alpha}{\partial y_{s,j}}\right)_{T,P,y_{t,k\neq s,j}} \tag{5}$$

where I specifies a constituent i in each sublattice s, see the paper by Sundman and Ågren [26]. Note that the individual partial derivatives, for example $\left(\frac{\partial\ G_M^\alpha}{\partial\ y_{s,i}}\right)_{T,P,y_{t,k\neq\ s,i}}$ has no physical meaning. Even in a substitutial phase α the derivatives with respect to the mole fractions must be summed in order to have the chemical potential of a component i:

$$\mu_i = G_M^\alpha + \left(\frac{\partial G_M^\alpha}{\partial x_i}\right)_{T,P,x_{j\neq i}} - \sum_j x_j \left(\frac{\partial G_M^\alpha}{\partial x_j}\right)_{T,P,x_{k\neq j}} \tag{6}$$

At equilibrium the relation between the chemical potential of an endmember I with different elements i on each subattice s and the chemical potentials of the elements i is:

$$\mu_I = \sum_s a_s \mu_i \tag{7}$$

where a_s are the site ratios of the sublattices of the phase. In same cases it is impossible to extract the chemical potentials of the individual elements from the chemical potential of a single phase. However, as explained in Sundman et al. [15], the algorithm used to calculate the equilibrium does not depend on this.

1.5 Additions from physical properties

For many phases there are additions from different physical properties, ${}^{\text{phys}}G_M$. In most cases these are defined per mole atoms whereas the Gibbs energy in eq. 1 is defined per mole formula unit. It means we have:

$${}^{\text{phys}}G_M^\alpha = N^\alpha \cdot {}^{\text{phys}}G_m^\alpha \tag{8}$$

where N^α is the number of moles of atoms in α and ${}^{\text{phys}}G_m^\alpha$ the expression for the Gibbs energy of the addition per mole atom. See also sections 2.3.5, 3.2.2 and 4.6.9

1.5.1 Other properties than the Gibbs energy

The Gibbs energy parameters has property index 1 (one) in OC. There are a number of predefined properties in the code, for example the Curie temperature (index 2), average

Bohr magneton number (3) etc. These are defined in the subroutine init_gtp and a tentative list of predefined identifiers is given in Table 4. More types can be added but one must choose a unique symbol for each and each is assigned a sequential index. The symbol must not be mistaken for a state variable symbol either. In the property records, see 4.6.6, for endmembers and interactions this propery index is used to identify the property.

The mobility of an element in a phase depends on the constitution and the temperature but the mobility does not contribute to the Gibbs energy. Anyway it is possible to define and enter mobilities in GTP to handle their T and composition dependence. The values of these, and any other separate property like the Curie temperature, can be obtained anytime by a special subroutine call.

One must consider that the mobility for each constituent depend individually on the constitution and a symbol MQ&X(BCC) is used to define the mobility of X in the BCC phase as a function of its constitution. Thus MQ&FE(BCC,FE) is the self diffusion of FE in BCC and MQ&FE(BCC,CR) is the mobility of FE in (almost) pure BCC Cr. If there are more data one can have interaction parameters like MQ&FE(BCC,CR,FE). Each mobility parameter can depend on T and P. The mobility can also depend on the magnetism.

Other properties that may be described by this model package are the Einstein θ, lattice parameters, elastic constants, resistivity, etc. see Table 4.

1.5.2 Model parameters identifiers

In addition to calculate the Gibbs energy for a phase the user can also enter parameters for other properties, some of which may be an addition to the Gibbs energy. These parameters can depend on T, P and the constitution of the phase. In this package all parameters are identified by:

- a model parameter identifier, listed in Table 4 and in Appendix B. For the use of properties in additions see section 4.6.9.
- the phase name,
- the constituent array (specifying the constituents in each sublattice, the fraction of which are multiplied with the parameter, see also section 2.3.4), and
- a degree with a value 0 to 9, usually a Redlish-Kister power, see section 2.3.2.

Some examples are G(FCC,FE), G(SIGMA,FE:CR:CR), G(LAVES,FE,MO:MO;2), TC(BCC,FE) or MQ&FE(FCC,FE), LNTH(BCC,FE:VA) see also 1.5.1.

The model parameter identifier (or property symbol) G is used for the Gibbs energy. Other identifiers such as TC is used for the Curie temperature and MQ&XY for the mobility of the element XY the the phase. A suffix **D** for a model parameter idenifier means this parameter belong to the disordered fraction set, see section 2.3.1.4.

This means it is possible to use models that does not give any contribution to the Gibbs energy, like the mobility, and let the gtp model package to handle the T, P and composition dependence of the model parameters like resistivity, viscosity etc. A tentative list of properties that have been defined is given in Table 4 and the actual definitions in the code is listed below.

Table 4: Model parameter identifiers, see also Appendix B. Note a model identifier can have a suffix D indicating it belongs to the disordered fraction set, see section 2.5

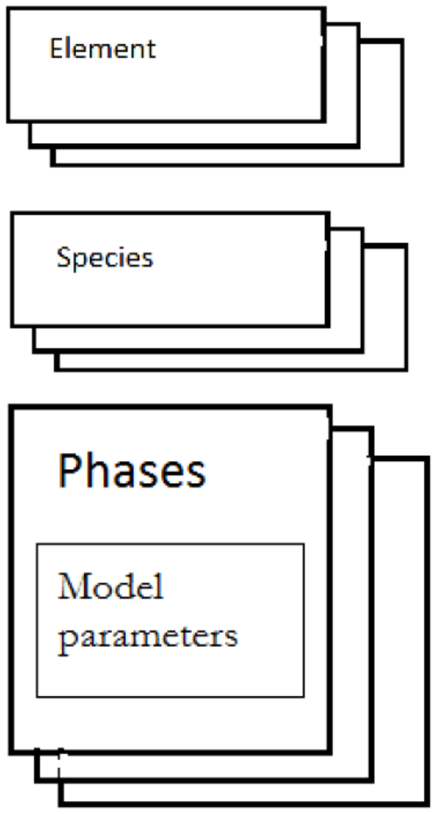
Index	Symbol	T	P	Specification	Meaning
1	G	T	P		Energy
2	TC	-	P		Combined Curie/Neel T
3	BMAG	-	-		Average Bohr magneton number
4	CTA	-	P		Curie temperature
5	NTA	-	P		Neel temperature
6	IBM	-	P	<constituent#sublattice>	Individual Bohr magneton number
7	LNTH	-	P		LN(Einstein temperature)
8	V0	-	-		Volume at T0, P0
9	VA	T	-		Thermal expansion
10	VB	T	P		Bulk modulus
11	VC	T	P		Aternative volume parameter
12	VS	T	P		Diffusion volume parameter
13	MQ	T	P	<constituent#sublattice>	Mobility activation energy
14	MF	T	P	<constituent#sublattice>	RT*ln(mobility freq.fact.)
15	MG	T	P	<constituent#sublattice>	Magnetic mobility factor
16	G2	T	P		Liquid two state parameter
17	THT2	-	P		Smooth step function T
18	DCP2	-	P		Smooth step function value
19	LPX	T	P		Lattice param X axis
20	LPY	T	P		Lattice param Y axis
21	LPZ	T	P		Lattice param Z axis
22	LPTH	T	P		Lattice angle TH
23	EC11	T	P		Elastic const C11
24	EC12	T	P		Elastic const C12
25	EC44	T	P		Elastic const C44
26	UQT	T	P	<constituent#sublattice>	UNIQUAC residual parameter
27	RHO	T	P		Electric resistivity
28	VISC	T	P		Viscosity
29	LAMB	-	P		Thermal conductivity
30	HMVA	T	P		Enthalpy of vacancy form.
31	TSCH	-	P		Schottky anomality T
32	CSCH	-	P		Schottky anomality Cp/R.
33	NONE	T	P		Unused

In case the parameters should affect the Gibbs energy, like the Curie temperature, an “addition” subroutine must be written using the value of this parameter. See section 4.6.9.

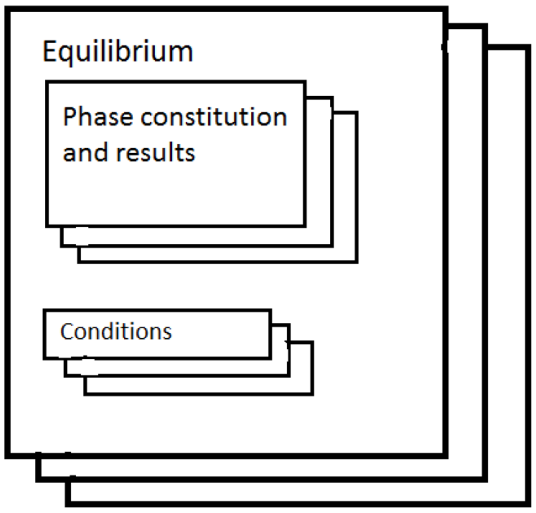
The TC and BMAG parameters are included for compatibility with the current magnetic model, many of the others are tentative.

1.6 Dividing the data in two parts

In OC the data are divided in two distict parts illistated in Fig. 1. All data describing the thermodynamic properties, i.e elements, species and phase structure and model parameters are stored in a “static” part. These data do not depend on the external conditions of the system. The second part is the “dynamic” part which contain the conditions of the equilibrium and the current chemical potentials and the constitutions, Gibbs energy etc. of the phases, at the current values of T, P and constitution of the phases.



(a) Static data



(b) Dynamic data

Figure 1: The static and dynamic momory in OC. The dynamic part is a Fortran TYPE record phase_varres described in section 5.2.4.

The dynamic data are stored in a Fortran TYPE record refered to by a pointer usually called **ceq** meaning "current equilibrium". It is explained in section 5.2.4. The ceq pointer is an argument in the call to all subroutines that change or extract data from the current equilibrium. It is easy to create several instances of equilibria, see section 8.6.12. This makes it easy store results from STEP/MAP commands and to run simulations with a large number of gridpoints, each associated with an equilibrium, and calculated in parallel using the OpenMP package.

1.7 Binary tree structure of the model parameters in each phase

The model parameters of each phase are stored in a binary tree structure as shown in Fig. 2 defining the way the Gibbs energy is calculated. In the property records the parameters for the Gibbs energy and other properties such as the magnetism are stored and the whole structure is part of the static memory data in Fig. 1. The binary tree structure specifies which constituent fractions should be multiplied with each parameter. The constituent fractions are stored in the dynamic memory area together with T, P and the calculated results.

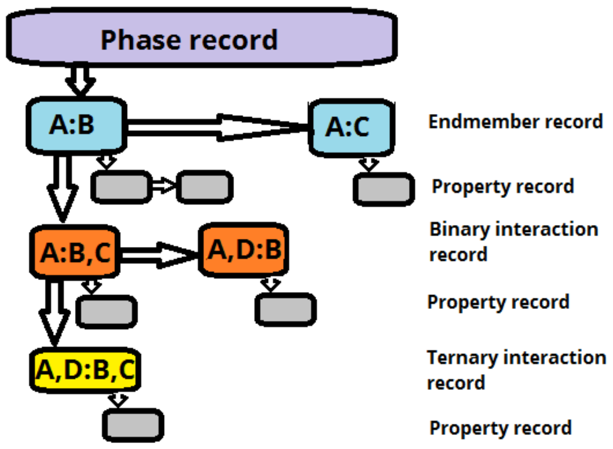


Figure 2: The model parameters are stored in a binary tree structure in the static data memory for a phase with a model (A,D)(B,C). Each box in the figure is a Fortran TYPE record and pointers are used to link them together. The parameters are stored in the property records. This structure makes it easy and fast to calculate the Gibbs energy and its first and second derivatives with respect to the constituent fractions.

The Gibbs energy expression in Fig. 2 represent a Gibbs energy model as

$$G = y_{1,\mathrm{A}} y_{2,\mathrm{B}} ({}^\circ G_{\mathrm{A:B}} + y_{2,\mathrm{C}} (L_{\mathrm{A:B,C}} + y_{1,\mathrm{D}} L_{\mathrm{A,D:B,C}}) + y_{1,\mathrm{D}} L_{\mathrm{A,D:B}}) + y_{1,\mathrm{A}} y_{2,\mathrm{C}} {}^\circ G_{\mathrm{A:C}} \quad (9)$$

where L is used for the excess Gibbs energy parameters. In OC the G is used also for these.

2 Basis

The basic terms and concepts of the GTP model package are explained here. Depending on your background some definitions my not agree with what you are used. OC is case insensitive, it means that an element written as Fe is the same as fe or FE or fE. The element Cobalt can be written co or CO and to specify the species carbon monoxide we must use c1o or c1o1 if there is also c1o2 in the system.

2.1 Elements

The elements have a sequential index arranged alphabetically. They have a one or two-letter symbol and a mass. The symbol must be letters A-Z all in upper case. In addition a ”name” of the element is stored, like “iron” for Fe, the name of a reference phase, the value of H298-H0 for the reference phase and S298 for the reference phase. The element symbol must be unique.

The electron and vacancy are entered as element -1 and 0 respectively but they are not really treated as elements except that they can be included in species. The electron has the symbol /- but one can also use /+ to denote a positive charge (or /- -1). Vacancies are denoted “Va” and have no fixed amount and will always have zero chemical potential at equilibrium. They will thus not affect the Gibbs phase rule.

An element can be be suspended, that means it can be hidden from application programs. If an element is suspended then all species that include this element are also suspended (implicitly) and also any phases which cannot exist unless the species is present as constituent, i.e. the species in the only constituent in a sublattice.

2.2 Species

The species are stoichiometric combinations of elements. They have a symbol and a stoichiometric formula. A species is identified by its symbol, not by its stoichiometric formula. The elements, including the vacancy, are the simplest species. The stoichiometry can be non-integer like a species $ALO_{1.5}$. The symbol must start with a letter A-Z and can contain letters (case insensitive), digits, the period “.”, the underscore character “_”, the slash “/”, the minus sign “-” and the plus sign “+”. No other special characters are allowed, for example parenthesis are not allowed.

The symbol of the species is often the same as the stoichiometric formula but it is not necessary. For example the molecule $C_2H_2Cl_2$ exists in 2 configurations either with CL on the same side or opposite. These can be identified by giving these the names C2H2CL2_CIS and C2H2CL2_TRANS.

In addition the mass is stored but this is redundant as it is calculated from the element masses. The species can have a charge like Al/+3 or O/-2. The charge can be real number.

Note that a species does not represent a phase, just a stoichiometric formula. Only as constituents of a phase the species are associated with a phase. The species H2O can be a constituent of solid ice, liquid water and gas. Thus species have no thermodynamic data. Their only function is to be constituents of phases where they can be assigned data. A species must not be confused with a phase just because many phases have a fixed stoichiometric formula. In OC the same species can be a constituent of many phases, also phases with variable composition.

When entering the stoichiometric formula for a species one may need to use stoichiometric numbers to separate one letter elements as OC is case insensitive. For example to distinguish CO as cobalt from C1O1 as carbon monoxide the digit 1 can be used after the element letter C. But otherwise the stoichiometric number 1 is not needed if the element have a two letter symbol, for example CAO can be used for a species with one atom of CA and one atom of O. One cannot use parenthesis in species symbols or when entering the stoichiometry, the stoichiometry of $Ca(OH)_2$ must be entered as CAO2H2 (or O1CAO1H2 or any other way giving the same stoichiometry).

The fact that a species has a symbol which is independent of its stoichiometry can be used to simplify complicated chemical stoichiometries, for example Acetonitrile, $C_2H_3N_1$ can have the symbol MECN.

In electronic materials the free electron can be entered with the stoichiometry VA/-1 and the "hole" with the stoichiometry VA/+1.

It is important to keep in mind that a species is identified with its symbol, not its stoichiometry. Thus a species with the symbol c1o1 cannot be found by searching for the species o1c1.

2.3 Phases and models

All thermodynamic data used in calculations are stored as part of a phase, see Fig. 2. The Gibbs energy of the phase is modeled as a function of temperature (T), pressure (P) and the constitution of the phase (Y) as given by eq. 1. Many different thermodynamic models can be used and each phase can have a unique model and also several different "additions" to this model, like magnetic and pressure models. But all phases are accessed from outside the model package in the same way. Thus the minimizer which is used to find the equilibrium state of a system does not depend on any particular model feature.

From outside the model package one can obtain information of the constitution of the phase, i.e. which constituents and their fractions (Y). The constituents are species (with fixed stoichiometry). As the compound energy formalism (CEF) is a very general and flexible model framework this has been the first implemented. From version 2 of OC the ionic liquid model, proposed by Hillert et al. [32] and based on the Temkin model [35] is also implemented.

Please refer to the paper or the book by the book by Lukas at al [30] for details.

In CEF one can define sublattices with specific species (elements or molecules) as constituents and the constituent fractions on each sublattice is unity. The constituents are assumed to mix randomly on each sublattice. As special cases CEF also includes the ideal gas, the classical regular solution model and the associated model, all with a single set of sites for the constituents.

One may implement models that do not use sublattices and have non-random entropy in GTP, for example CVM or the modified quasichemical model, see Appendix E. Inside the CEF framework this can be done by treating the phase as having a single lattice and the probabilities of the different clusters are fractions and the clusters are species. Inside the model package the appropriate non-random configurational entropy is then evaluated. The reason to base the model package on the constitution and not the composition of the component is to avoid a two-level minimization when calculating the equilibrium. But there is nothing, except speed, preventing from using the mole fractions as external variables for some models and implement an internal minimization to determine the configuration which will give the minimum Gibbs energy for the given external conditions.

There are some models (not implemented in OC) which allow a variable stoichiometry of the constituents of the phase. Such models must be implemented as a two-level minimization where the model from outside the model package depend only the mole fractions of the elements or some other set of components. It may also be necessary to have some special way of entering parameters for such a model.

2.3.1 Phase specification

A phase has a name and some model information. If it is a CEF model it has number of sublattices, the sites in each sublattice (a fixed real value) and a list of constituents on each sublattice. One may have a single constituent in a sublattice. An empty sublattice can have the vacancy as the only constituent.

Phases can have model parameters for many different types of properties, see section 1.5.2. Normally these depend on the constitution of the phase and on T and P. Some phases has a fixed composition but for most phases has a composition dependence of the model parameters is divided in two groups, for endmembers and interactions between two or more constituents in one or more sublattices. A phase with variable composition has also a contribution to its Gibbs energy from a configurational entropy.

2.3.1.1 Non-ideal configurational entropy, LRO and SRO With few exceptions the phase models implemented in OC assume ideal confingurational entropy on each set sof sites for the elements. The CEF model can describe long range ordering (LRO) using sublattices but the short range order is not possible to model explicitly, it must be included in the various excess parameters, for example the reciprocal parameter explained in section 2.3.2.4.

There are plans to implement the Cluster Variation method (CVM) [34] for tetrahedral clusters in FCC and BCC but larger clusters are prohibitively slow to calculate the Gibbs energy in multicomponent systems. For intermetallic phases which always have LRO, the SRO can easily be included in the CEF model parameters for the LRO [11].

The two cases on non-ideal confingurational entropy is the UNIQUAC model for polymers [12] and the modified quasichemical model (MQMQA) [22, 23, 24, 17]. The equations and the implementation in OC are discussed in Appendix E.

The configurational entropy has normally no model parameters but the UNIQUAC model requires for each constituent a volume and surface area parameter.

2.3.1.2 Models for the liquid phase

The liquid is a problematic phase, in particular SRO, and there are several models that can be used to describe this. The simplest is an associated model where a species is introduced in the liquid with a composition at the SRO composition. When this species is dominant the configurational entropy is very small. But it overestimates the configurational entry at other compositions and also when extrapolating to multicomponent systems.

The ionic 2-sublattice liquid model as proposed by Hillert et al. [32] and is based on the Temkin [35] assumption that cations and anions mix separately. This can describe metal-nonmetal systems such as oxides and also metallic liquids as regular solutions. It has a problem creating unexpected miscibility gaps when extrapolating to higher order systems.

Recently the modified quasichemical model, MQMQA, proposed by Peltion et al. [24] has been implemented but is still tested. It is described in the Appendix E.

2.3.1.3 Endmember parameters

The term ”endmember” refers either to a single constituent in a phase with no sublattices or to a combination of one constituent in each sublattice. The latter defines a compound and this is the origin of the name “compound energy formalism”. When writing an endmember the constituents in each sublattice are separated by a colon.

The thermodynamic model parameters are entered for the endmembers and for possible interaction parameters.

Phases with molecules as constituents (as the gas) or vacancies in some sublattices can have different amount of components per formula unit of the phase and the ionic liquid model has variable number of sites for cations and anions to maintain electro-neutrality. This will be taken care of when calculating the configurational entropy and the mass balance during with the minimization. The parameters for the phase are always per mole formula unit of the phase.

When entering an endmember parameter the software writes the amount and reference states of the elements this is referred to. As already mentioned care must be taken that the parameter value correspond to the correct amount of atoms as this can vary between endmembers of the same phase. For example if the FCC phase is defined with 2 sublattices with one site in each, the endmember G(FCC,TI:VA) is for one mole of atoms whereas G(FCC,TI:C) is for two moles of atoms.

For compatibility with other software [27] parameters for the "mixed quadruplets" in the MQMQA model which represent endmembers, for example AB/X, does not include the reference state when listed but this is calculated internally depending on the stoichiometry of the AB/X quadruplet.

2.3.1.4 Parameters in the disordered fraction set

Phases with a disordered fraction set can have model parameters in both the ordered and the disordered fraction set. The parameters in the disordered fraction set have a suffix **D**. For an ordered FCC phase with an interstital sublattice modelled as

$(A,B)_{0.25}(A,B)_{0.25}(A,B)_{0.25}(A,B)_{0.25}(C,Va)_1$ with a disordered set $(A,B)_1(C,Va)_1$

has endmembers such as GD(FCC,A:Va) and GD(FCC,B:Va) for pure A and B and G(FCC,A:A:A:B:Va) is the Gibbs energy for the $L1_2$ ordered structure relative to A and B in the FCC lattice. However, without the F-option, see section 4.6.1, one has to enter all 4 variants of the $L1_2$ ordering parameter G(FCC,A:A:B:A:VA), G(FCC,A:B:A:A:VA), G(FCC,B:A:A:A:VA) due to symmetry.

Interaction parameters in the disordered fraction set also have the suffix "D", for example GD(FCC,A:C,Va) or GD(FCC,A,B:VA).

2.3.2 Interaction parameters

For interaction parameters one must add one, two or more constituents in any sublattice, separated by a comma, from the other constituents in the same sublattice. The fractions of the constituents specified in the parameter will be multiplied with the parameter value. Sometimes the parameter value depends also on these fractions as in the Redlich-Kister series.

An interaction parameter may also have a degree specifying that it has a more complex composition dependence. The degree is a digit 0 to 9 given after a semicolon. The default degree is zero. The meaning of the degree depend on the excessmodel,

2.3.2.1 Redlich-Kister-Muggianu (RKM) excess model

For a binary interaction in CEF the interaction parameters are by default a Redlich-Kister (RK) power series:

$$L_{ij} = \sum_{\nu=0} (y_i - y_j)^\nu \cdot \ {}^\nu L_{ij} \tag{10}$$

where y_i and y_j are the constituent fractions on the same sublattice and ν is the degree of the parameter. For $\nu = 0$ the parameter is composition independent.

Note that the sign of the odd parameters depend on the order of the constituents and the rule is they should be in alphabetical order of the species name. Each such ${}^{\nu}L_{ij}$ coefficient is a function of temperature and pressure but it is strongly recooemded never use more than linear T dependence unless there are excess heat capacity data. Only small P dependence can be modelled with excess parameters, maximum a few kbar. For higher pressures special volume models ae needed.

The default ternary excess model in OC is the Muggianu model which is symmetric and allows using the multicomponent fractions without any rearrangements. For the Kohler and Toop models see section 2.3.2.5 and Appendix D.

2.3.2.2 Assymetical binary power series

Sometimes an asymetrical power series is used for a binary excess parameter. In a binary system such a power series can be converted to a Redlich Kister power series, see section 2.3.2.1, but may require many more parameters. The converted RK series will give different extrapolations to higher order systems. A power series is basically:

$$L_{ij} = \sum_{\nu \geq 0} \sum_{\mu \geq 0} y_i^{\nu} y^{\mu} \cdot \ {}^{\nu\mu}L_{ij} \tag{11}$$

but this type of parameter does not fit in the OC syntax for parameters. However, it is always possible to rearrange such an expression as a RK series in eq. (10). Using polynomial functions such as eq. 11 require the use of Kohler or Toop models to handel the asymetry.

2.3.2.3 Ternary composition dependent parameters

For a ternary parameter there is also possibilities to have composition dependence:

$$\begin{aligned} L_{ijk} &= v_i \ {}^{0}L_{ijk} + v_j \ {}^{1}L_{ijk} + v_k \ {}^{2}L_{ijk} && (12)\\ v_i &= y_i + (1 - y_i - y_j - y_k)/3 && (13)\\ v_j &= y_j + (1 - y_i - y_j - y_k)/3 && (14)\\ v_k &= y_k + (1 - y_i - y_j - y_k)/3 && (15) \end{aligned}$$

where y_i, y_j and y_k are the constituent fractions on the same sublattice. The v_i are the same as y_i in the ternary system and the additional terms have been added to make this contribution symmetrical in higher order systems.

2.3.2.4 The reciprocal parameter and higher order parameters

For a phase with two or more sublattices with two or more constituents one can use so called reciprocal interaction parameters which defines a simulteneous interaction between two pais of constituents. This gives a contribution to the Gibbs energy, $\Delta\ {}^{\mathrm{rx}}G$, calculated as:

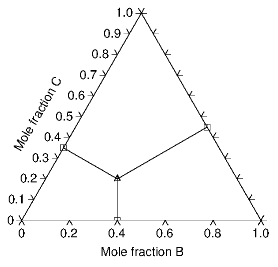
$$\Delta\ {}^{rx}G_M = y_{si} y_{sj} y_{tk} y_{tl} L_{ij:kl} \tag{16}$$

and this can depend on the constitution but I am not really sure how it is implemented, I think it is:

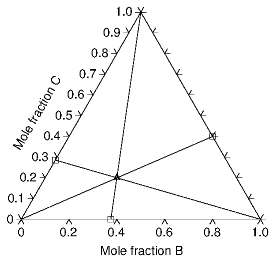
$$L_{ij:kl} \;=\; {}^0L_{ij:kl} + (y_{si} - y_{sj})\;{}^1L_{ij:kl} + (y_{tk} - y_{tl})\;{}^2L_{ij:kl} \qquad (17)$$

In OC one can enter interaction parameters with any number of interacting constituents but with more than two interacting constituents the parameter itself cannot depend on the constituent fractions.

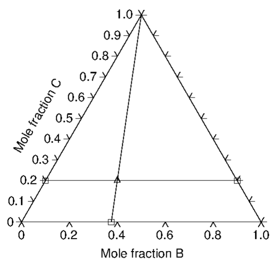
2.3.2.5 Other extrapolation methods of binary parameters In some software other models than Muggianu are used for extrapolation to ternary systems. In a multi-component system each ternary subsystem can have a different extrapolation and in OC an attempt has been made to implement this using the technique explained in [25, 21]. Three of the methods are plotted in Fig. 3.



(a) Muggianu



(b) Kohler



(c) Toop-Kohler

Figure 3: Ternary extrapolation methods of binary excess parameters

The Muggianu model is symmetrical and it is interesting to note that it uses the binary compositions at the shortest distance from from the ternary composition to the binary side. This is identical to use directly the overall composition in the binary parameter so there is no need to need to use extra code if Muggianu model is the only one used. But if the Kohler or Toop extrapolation methods are used for one or more ternary subsystems of a phase the user (or database) must specify which model is used for all ternary subsystem in each sublattice and for the Toop model the asymmetric constituent must also be specified. As these extrapolation methods are used only for interactions inside a sublattice, or for phases without sublattices, the sublattice notation is omitted in this section.

The extra information needed for a binary paraeter L_{ij} when using Kohler or Toop models is the third constituent when this parameter is involved in different ternary extrapolation. The reduced "binary" fractions ξ_{ij}, ξ_{ji} and a quotient σ_{ij} are calculated in the binary $i - j$

system using:

$$\xi_{\mathrm{ij}} = y_i + \sum_{k \in A} y_k \tag{18}$$

$$\sigma_{ij} = \sigma_{\mathrm{ji}} = 1 - \sum_{m \in ijk} y_m \tag{19}$$

where the summation is over k for all ternary subsystems A with constituents $(i - j - k)$ in which j is an asymmetric (i.e. Toop) constituent and the summation over m is for all subsystems where $i - j$ is a Kohler extraplation from m.

When a phase use Kohler or Toop extrapolations the database, or user, must specify the type of extrapolation method for all ternary subsystem $(i - j - k)$. This will affect the calculation of all binary excess parameters L_{ij} in those ternaries. This is done using the command:

AMEND PHASE < *name* > TERNARY_EXTRAPOL
< *Kohler or Toop* > constituent-name1 constituent-name2 constituent-name3
< *Kohler or Toop* > constituent-name1 constituent-name2 constituent-name3
...

The Toop constituent must be the first of the three constituent specified for a Toop model. All ternary subsystem not specified will be assumed to have the Muggianu extrapolation.

The information how to calculate ξ_{ij} depend on the type of ternary extrapolations in which the binary $i - j$ is involved and this information must be stored with the binary parameter for each ternary combination $i - j - k$. The information must include the Toop element or if it is a Kohler extrapolation. A binary which is only involved in symmetric extrapolations has always $\xi_{ij} = y_i$. See Appendix D for a detailed explanation of the data structures and algorithms involved.

2.3.3 A note on excess model parameter assessments

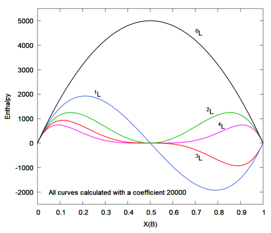
Thermodynamic databases for Calphad consist of data for pure elements in differet phases that is normally agreed internationally because changing this would make many existing assessment useless. Normally an assessment of a binary and ternary system will describe interaction, excess parameters in the liquid and various solod solutions. Additionally Gibbs energies for compounds or intermetalic phases are also fitted to experimental and theoretical data.

By long experience one should restrict the number of interaction parameters fitted for a binary system, rarely more than 3 Redlich-Kister coefficients can be fitted in any binary system for a phase.

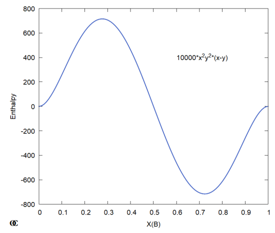
However, there are cases, often related to LRO or SRO, when the higher order RK coefficients have too strong influence close to the pure elements but very little influence in the center of the system, as shown in Fig. 4(a). Instead parameters which depend on higher powers of $x_{\mathrm{A}}x_{\mathrm{B}}$ could be useful. As all binary excess models can be transformed to each other, a parameters depending on $x_A^3 x_B^3$ can be described by combining several RK parameters as listed in the Table 5.

Table 5: Table how to use a set of RK parameters to form another power series. All RK parameters are always multiplied with $x_A x_B$.

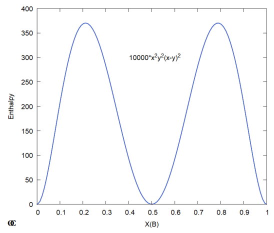
parameter	${}^0L$	${}^1L$	${}^2L$	${}^3L$	${}^4L$	Figure
$\mathrm{M}x_A x_B$	M	0	0	0	0	4(a)
$\mathrm{M}x_A x_B(x_A - x_B)$	0	M	0	0	0	4(a)
$\mathrm{M}x_A x_B(x_A - x_B)^2$	0	0	M	0	0	4(a)
$\mathrm{M}x_A^2 x_B^2$	M/2	0	–M/2	0	0	
$\mathrm{M}x_A^2 x_B^2(x_A - x_B)$	0	–M/4	0	M/4	0	4(b)
$\mathrm{M}x_A^2 x_B^2(x_A - x_B)^2$	0	0	M/4	0	–M/4	4(c)
$\mathrm{M}x_A^3 x_B^3$	M/16	0	–M/8	0	M/16	4(d)



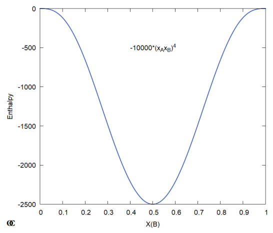
(a) RK



(b) $x_A^2 x_B^2(x_A - x_B)$



(c) $x_A^2 x_B^2(x_A - x_B)^2$



(d) $x_A^4 x_B^4$

Figure 4: The effect on the enthalpy of some excess model parameters. In (a) the normal RK parameters, all calculated for a coefficient 20000, have the same slope close to the elements and small effect in the middle, only the ${}^0L$ parameter is nonzero. In (b), (c) and (d) a combination of several RK parameters, have zero slope of the enthalpy at the pure elements and a larger influence in the center. These combined RK parameters can be used to fit experimental data in the center without changing the properties close to the pure elements.

For example the curve in Fig. 4(d) is the enthalpy curve for $x_A^3 x_B^3$ parameter which was obtained by setting several RK coefficents in Table 5. It is strongly recommended never to assess many RK coefficents independently but in Table 5 show how a single optimizing variable can be used in several RK terms to provide a different composition dependence. In particular when SRO is can be important to have zero contribution from the excess parameters close to the pure elements.

A very good optimizing software may by itself elucidate such relations between the excess model parameters and the experimental data but any help by the user is appreciated. For example if a user wants to assess ${}^0L$, ${}^1L$ and in addition a parameter with a $x_A^2 x_B^2$ depenedence he can enter 3 functions with optimizing variables, F0AB, F1AB and F2AB and then the parameters:

G(phase,A,B;0) as F0AB+0.5F3AB;
G(phase,A,B;1) as F1AB;
G(phase,A,B;2) as -0.5*F3AB.

2.3.4 Wildcards in model parameters

For phases with several sublattices a model parameter may be independent of the specific constituent in a sublattice. This can be indicated by used an asterix, "*", instead of a constituent in that sublattice. This feature can also be used to use model parameters that represent "bonds" rather than endmember energies. For a parameter with a wildcard in a sublattice the fraction for that sublattice is replaced by the factor 1.0.

2.3.5 Physical models

Traditionally CALPHAD models describes the variation of the Gibbs energy as a function of T, P and composition without bothering about the origin of the contributions. In some cases this is no so good and for the contribution due to ferromagnetic transitions one has introduced a more physical contribution that depend on two new variables, the Curie temperature and the Bohr magneton number, both of which depend on the constitution. Such *additions* to the traditional Gibbs energy expression may become more frequent when also phonon contributions are modeled down to 0 K as a function of a constitution dependent Einstein temperature. Hopefully this can lead to better extrapolations to higher temperatures as well as multicomponent systems.

There are no description of the well established additions here because they can be found in Lukas et al [30]. But some of the new models will be described here.

The Gibbs energy in eq. 1 is per mole formula unit of the phase and so is also the ${}^{\text{phys}}G_M$ term but the expressions found in textbooks are always per mole of atom. Thus the Gibbs energy expression for the physical model must be multiplied with the number of atoms per formula unit of the phase. This is done by setting a bit ADDPERMOL in the addition record, see section 3.2.2. How to do that will be described in (a future update of) the User Guide.

2.3.5.1 Einstein heat capacity model The Einstein model is not in Lukas et al [30] and its Gibbs energy contribution, obtained by integrating the heat capacity contribution due to the Einstein model is:

$$\Delta G_m^{Ein} = \frac{3}{2}R\theta + 3RT\ln(1-\exp(-\frac{\theta}{T})) \qquad (20)$$

where θ is the Einstein temperature which can vary with composition and P but not T. The Einstein heat capacity model is preferred to the Debye model because it can be integrated to a closed expression and both models must anyway have additional terms to fit the heat capacity data for any real element. The Einstein T, θ, is not a materials constant but a value fitted to the low T heat capacity curve for the elements. In some cases weighted combinations of several Einstein functions with several θ can be used for an element, for exaple carbon, C.

Note also that eq. 20 is per mole atom, if a phase has several atoms in a sublattice it must be multiplied with this. A phase where the number of atoms vary with composition, for example in interstitial solutions, the value must be multiplied with the number of real atoms as shown in eq. 8. This must also be included in derivatives with the fraction of the constituents.

The composition dependence of θ is best decribed by by adding $\ln(\,^\circ\theta_i)$ for each constituents i (including possibly some excess parameters) and then calculate:

$$\theta = \exp(\sum_i y_i \ln(\,^\circ\theta_i)) \tag{21}$$

$$\frac{\partial\theta}{\partial y_i} = \theta\frac{\partial}{\partial y_i}(\sum_j y_j \ln(\,^\circ\theta_j)) \tag{22}$$

We have to derive the first and second derivatives both wrt T and composition as θ will depend on the composition.

$$G_M = \frac{3}{2}R\theta + 3RT\ln(1-\exp(-\frac{\theta}{T})) \tag{23}$$

$$\begin{aligned}\frac{\partial G_M}{\partial T} &= 3R\ln(1-\exp(-\frac{\theta}{T})) + 3RT\frac{-\frac{\theta}{T^2}\exp(-\frac{\theta}{T})}{1-\exp(-\frac{\theta}{T})} \\ &= 3R\ln(1-\exp(-\frac{\theta}{T})) - \frac{3R\frac{\theta}{T}}{\exp(\frac{\theta}{T})-1}\end{aligned} \tag{24}$$

$$\begin{aligned}\frac{\partial G_M}{\partial y_i} &= R(\frac{3}{2}+\frac{3}{\exp(\frac{\theta}{T})-1})\frac{\partial\theta}{\partial y_i} \\ &= 1.5RT(\frac{\exp(\theta/T)+1}{\exp(\theta/T)-1})(\theta/T)\frac{\partial\ln(\theta)}{\partial y_i}\end{aligned} \tag{25}$$

$$\begin{aligned}\frac{\partial^2 G_M}{\partial T^2} &= 3R(-\frac{\frac{\theta}{T^2}}{\exp(\frac{\theta}{T})-1} - \frac{-\frac{\theta}{T^2}(\exp(\frac{\theta}{T})-1)+(\frac{\theta}{T})^2\exp(\frac{\theta}{T})}{(\exp(\frac{\theta}{T})-1)^2}) \\ &= 3R(-\frac{\frac{\theta}{T^2}}{\exp(\frac{\theta}{T})-1} + \frac{\frac{\theta}{T^2}}{\exp(\frac{\theta}{T})-1} - \frac{\theta^2}{T^3}\frac{\exp(\frac{\theta}{T})}{(\exp(\frac{\theta}{T})-1)^2}) \\ &= -3R\frac{\theta_E^2}{T^3}\frac{\exp(\frac{\theta}{T})}{(\exp(\frac{\theta}{T})-1)^2}\end{aligned} \tag{26}$$

The heat capacity at constant P is:

$$C_P = -T\frac{\partial^2 G_M}{\partial T^2} = 3R(\frac{\theta}{T})^2\frac{\exp(\frac{\theta}{T})}{(\exp(\frac{\theta}{T})-1)^2} \tag{27}$$

which is the correct expression for C_V according to Einstein:

$$C_V \quad = \quad 3R(\frac{\theta}{T})^2\frac{\exp(\frac{\theta}{T})}{(\exp(\frac{\theta}{T})-1)^2} \tag{28}$$

This gives $C_P = 0$ at 0 K. The value for S is also zero at 0 K from eq. 24 as $\exp(-\frac{\theta}{T}) = 0$ when $T = 0$ as $\theta > 0$.

$$S(T=0) \quad = \quad -\left(\frac{\partial G}{\partial T}\right)_{T=0} = 0 \tag{29}$$

and for the Gibbs energy the value at $T = 0$ will be $\frac{3}{2}R\theta$.

2.3.5.2 The liquid two-state model The liquid two-state model is not in Lukas et al. [30] and the model equation and some partial derivatives are given here as part of the documentation.

In the liquid two-state model the Gibbs energy of the liquid is described as:

$$G_M^{\text{am+liquid}} \quad = \quad G_M^{\text{Ein}}(\theta) - G_M^{\text{2state}} + G_M^{\text{corr}} \tag{30}$$

$$G_M^{\text{2state}} \quad = \quad RT\ln(1+\exp(-\frac{G2}{RT})) \tag{31}$$

where G_M^{Ein} is the Gibbs energy of an amorphous state with an Einstein heat capacity contribution which depends on an estimated composition dependent θ for the amorphous state. The function G_M^{2state} describes the transition from the amorphous to the liquid state. At 0 K it gives zero contribution to the Gibbs energy, entropy and heat capacity and above the melting T it should decribe the real liquid properties. The parameter $G2$ can depend on T, P and the composition of the liquid. For a substitutional liquid that would be

$$G2 \quad = \quad \sum_i (x_i\ {}^\circ G2_i + (\sum_{j>i} x_j G2_{ij} + \cdots)) \tag{32}$$

where x_i is the mole fraction of component i. For a pure element ${}^\circ G2_i$ is:

$${}^\circ G2_i \quad = \quad a0_i + a1_i T + a2_i T\ln(T) + \cdots \tag{33}$$

where $a0_i$ is related to the enthalpy of melting and $a1_i$ to the entropy of melting (close to R for many metals according to Richard's rule). But $a0_i, a1_i$ and $a2_i$ and possibly more terms can be treated as parameters to be fitted to all the experimental data like the melting T of the crystalline solid, the enthalpy and entropy of melting and the heat capacity of the stable liquid.

The final term G_M^{corr} in eq. 30 can be used to adjust the Gibbs energy of the stable liquid in addition to the previous terms. In fact this G_M^{corr} is the parameter used to describe the liquid phase without the two-state model. It is also composition dependent and for a substitutional liquid it can be written:

$$G_M^{\text{corr}} = \sum_i (x_i\ {}^\circ G_i^{\text{corr}} + (\sum_{j>i} x_j G_{ij}^{\text{corr}} + \cdots)) \tag{34}$$

For a pure element i:

$${}^\circ G_i^{\text{corr}} = b0_i + b1_i T + b2_i T^2 \tag{35}$$

where $b0_i$ can be interpreted as the enthalpy difference between a metastable liquid and a (slightly less metastable) amorphous phase at 0 K for component i. $b1_i$ is an entropy at $T = 0$ K due to its disordered state and $b2_i$ can be used to fit the properties of the amorphous phase. One can use higher powers in T but not any $T\ln(T)$ or negative powers because such terms would give a contribution to the heat capacity at 0 K.

At 0 K eq. 30 should represent the Gibbs energy of the amorphous state with $C_P = 0$. There can be a finite entropy at 0 K because the amorphous phase is not a defect free crystal. At increasing T the heat capacity for the amorphous state should follow an Einstein curve.

Anyway, to implement this model we have to calculate some derivatives of G_M^{2state} with respect to T and constitution, x_i of the liquid. We must also use the derivatives of the Einstein heat capacity function for the amorphous phase in section 2.3.5.1. For simplicity the superscipt "2state" is skipped in the following equations. Note that $G2$ and all its derivatives are calculated together with other parameters for each iteration.

$$G_M = RT\ln(1+\exp(-\frac{G2}{RT})) \tag{36}$$

$$\begin{aligned}\frac{\partial G_M}{\partial T} &= R\ln(1+\exp(-\frac{G2}{RT})) + RT\frac{(-\frac{1}{RT}\frac{\partial G2}{\partial T} + \frac{G2}{RT^2})\exp(-\frac{G2}{RT})}{1+\exp(-\frac{G2}{RT})} \\ &= R\ln(1+\exp(-\frac{G2}{RT})) + \frac{-\frac{\partial G2}{\partial T} + \frac{G2}{T}}{\exp(\frac{G2}{RT})+1}\end{aligned} \tag{37}$$

$$\frac{\partial G_M}{\partial y_i} = -\frac{\partial G2}{\partial y_i}\frac{1}{\exp(\frac{G2}{RT})+1} \tag{38}$$

And for the second derivatives:

$$\frac{\partial^2 G}{\partial T^2} = (-\frac{\partial^2 G2}{\partial T^2} + \frac{(\frac{G2}{T})^2 + (\frac{\partial G2}{\partial T})^2 - 2\frac{G}{T}\frac{\partial G2}{\partial T}}{RT(1+\exp(-\frac{G2}{T}))})/(1+\exp(\frac{G2}{RT})) \tag{39}$$

For the moment I skip the 2nd derivatives wrt fractions ...

2.3.5.3 The Equi Entropy Criterion (EEC) This has been proposed as part of the new 3rd generation unary project in order to avoid breakpoints in the extraplation of the

Gibbs energy of pure elements or compounds above their melting T. It is described in Sundman et al. [6].

The idea is that at any T a solid phase with higher entopy than the liquid must not be allowed to become stable.

The EEC is set by the command “SET ADVANCED EEC Y 1000”
where the final value, 1000, is the lower T limit for testing the entropy.

2.3.5.4 The magnetic model The magnetic model has been modified from that described by Hertzman and Sundman [33] to that proposed by to Qing and Sundman [20] has been implemented together with the effective Bohr magneton number according to Wei et al. [18].

The new magnetic model is selected by specifying the “Antiferromagnetic factor” equal to zero (0) in the command “AMEND PHASE”.

2.4 Components

The term ”component” is not used for the modeling, only for controlling the system externally. The term component is reserved for an irreducible set of species that describe the composition space of the system, sometimes called “system components”. In most cases these components are the elements. The species that can dissolve in a phase are called constituents and can be vacancies, elements, ions or molecules. For most simple models the constituents of the phases are identical to the elements, i.e. to the components.

The components are important as they determine the number of conditions that can be set in order to calculate the equilibrium of the system because one can only set as many conditions as one has components and for T and P. Many conditions, like the amounts or chemical potentials, can only be specified for the components, not for an individual species. This does not mean that the calculation of equilibria is limited to conditions on the amounts of the components, only that the number of components determine the number of conditions that must be set in order to calculate the equilibrium.

In GTP the elements are by default the components but the user may change this to any orthogonal set of species. When working with oxides with no degree of freedom for the oxygen content, like in a quasibinary CaO-SiO_2 system, it may be convenient to select the oxides CaO and SiO_2 as components and then leave the content of the third element undetermined by setting an arbitrary (positive) value of its activity. But in such cases it is necessary to know that the models used for the phases does not allow the content of this element to vary independently. If a gas phase is included in the system it may be more convenient to set the gas as fixed with zero amount, although that may increase the calculation time.

2.5 Fraction sets

In some cases phases with LRO modeled with several sublattices have some properties that are independent of the configuration, only of the composition. In such a case it can be convenient to split the total Gibbs energy in two separate Gibbs energy functions, this is called a "partitioned" or "splitted" model. It was originally used for phases with order/disorder transformations like A1/L1$_2$ or A2/B2 where the phase can be completely disordered. All parameters to describe the disordered state where collected in a phase that had no sublattices for ordering (it could have a sublattice for interstitials). This made it also easier to include ordering in multicomponent databases where many subsystems have A1 (fcc) or A2 (bcc) phases without ordering.

The Gibbs energy of a partitioned phase used to be written in Thermo-Calc:

$$G_M = G_M^{\text{dis}}(x) + \Big(G_M^{\text{ord}}(y) - G_M^{\text{ord}}(y=x)\Big) \tag{40}$$

$$\Delta G_M^{\text{ord}} = G_M^{\text{4SL}}(y) - G_M^{\text{4SL}}(y=x) \tag{41}$$

The parameters for the ordered phase, as disordered, was included in the disordered phase.

There is no reason to do this with the OC software as the disordered parameters are included in the same phase as the ordered. The Gibbs energy for a partitioned ohase is simply:

$$G_M = G_M^{\text{dis}}(x) + G_M^{\text{4SL}}(y) \tag{42}$$

where the parameters in the disordered part only has parameters that describe the disordered contribution, independent of the parameters for the ordering.

A variant of this model have been applied also to intermetallic phases that never disorder completely like σ, μ, Laves phase and similar. The equation is then slightly different as there is no need to describe the completely disordered state

$$G_M = \Big(G_M^{\text{dis}}(x) + T\ {}^{\text{cfg}}S_M^{\text{dis}}(x)\Big) + G_M^{\text{ord}}(y) \tag{43}$$

By adding $T\ {}^{\text{cfg}}S_M^{\text{dis}}(x)$ the configurational entropy is only calculated for the ordered part.

This second variant of the disordered model is very interesting because many intermetallic phases like the σ phase have a very large number of end members in multicomponent systems and they must all be given a reasonable value, although in databases today most of them are just equal to the sum of the reference energies of the endmembers, i.e. they can be set to zero. With the increased use of first principles calculations for ordered endmembers, only those which will decrease the Gibbs energy need to be included.

In the OC software we have taken this into account from the beginning and due to the parameters in the disordered part it is not necessary to enter all endmembers of the ordered part, only those which is known to be stable or close to be stable. *Ab initio* data can be used for this when experimental data are missing. The disordered part is entered as an extra "fraction set" of the ordered phase and one simply specifies how many sublattice that should be added together to for the disordered part. For an fcc phase with 5 sublattices, 4 for ordering and one interstitial, one simply gives 4, assuming the 5th sublattice is the interstitial one. For a σ phase modeled with 3 sublattices that all disorder together, one gives 3.

For a ternary σ phase with 5 sublattices one have more than 250 endmembers which all must be calculated. With the new software this may be reduced to less than 10 which will result in a significant increase in calculation speed compared to the case when all endmembers have a value. The maintanance of the database will also be simplified as fewer endmembers are stored.

The new software will create a fraction set record that gives the way to add fractions (the coefficients b_{ij} are stored there as they are also needed to calculate the contribution to the partial derivatives). A link to this fraction set record is stored in the phase_varres record linked from the phase record. When setting the constitution of the phase, i.e. the site fractions, the disordered fractions will be automatically calculated and saved in another phase_varres recored linked from the ordered fraction set record. In this way it will thus be straightforward to calculate the ordered and disordered part with the same subroutine, one just have to organize the calculations of each part separately and then add all values, including first and second derivatives, together in the end.

2.6 Miscibility gaps and composition sets

Composition sets is the way OC handles the case when a phase with the same structure can appear simultaneously with two or more compositions. One case this is necessary is when a phase split up into two or more miscibility gaps like the bcc phase in the Cr-Mo system. Less obvious cases is when a phase has an order/disorder transition like the fcc phase in Au-Cu forms $L1_2$ and $L1_0$ structures. But also in that case the ordered and disordered phases are described with the same thermodynamic data. Finally this is technique is also used to describe the cubic and hexagonal carbides where the metallic atoms are the same as fcc and hcp lattices but interstitial atoms like C and N can occupy most of the available interstitial sites.

Outside the OC package, for example in the TQ package, one can obtain information about the number of composition sets for a phase using the function "nocs(iph)" for each phase "iph". Inside the OC package the phase number and composition set number are usually given separately. A phase must have at least one composition set and the maximum number is 9. The composition sets larger than 1 is normally suffixed after the phase name separated by a # like BCC#2. If omitted composition set 1 is assumed.

If there are several composition sets of a phase there is a separate phase_varres record for each composition set. Normally these have identical set of constituents but it is possible to suspend constituents in each composition set separately. At the time when a fraction set is added to a phase this must not have any suspended constituents.

Fraction set records have no array or free list, they exist only inside the phase_varres records. (This created some extra headache when implementing them as a simple assignement of a record to another did not create any new record in Fortran08).

2.7 State variables

State variables are, as the name indicate, only dependent on the state of the system, not how this state was reached. In principle they have a well-defined value only at equilibrium but in the modeling they can be extended also outside this range.

The state variables recognized by GTP and that can be used are defined in Table 6. Many of them can have several indices and/or normalizing quantities. In the box below there are some examples of using these in conditions and otherwise.

N=1	means the system has one mole of atoms
N(H)=1	means the system has 1 mole of component H
N(GAS,H)=1	means the gas phase has 1 mole of component H
NP(LIQUID)=1	means the liquid phase has 1 mole of components
X(H)=0.3	means the mole fraction of component H in the system is 0.3
X(GAS,H)=0.3	means the gas phase has a mole fraction of 0.3 of H
B=1000	means the system has a mass of 1000 grams
W%(CR)=18	means the system has 18 mass percent of component CR
W(LIQUID,C)=0.01	means the liquid has 0.01 mass fraction of carbon
H	is the total enthalpy of the system
H(LIQUID)	is the enthapy of the current amount of liquid (can be zero)
HM(LIQUID)	is the enthalpy of liquid per mole components
HV(LIQUID)	is the enthalpy of liquid per m^3
HW(LIQUID)	is the enthalpy of liquid per gram
HF(LIQUID)	is the enthalpy of one formula unit of liquid

3 Fortran data structures

A number of arrays must be dimensioned, this is done by defining some constants (called PARAMETER in Fortran) and then using these to dimension arrays with fixed size. It may be possible to set these limits in a more flexible way by allocating the arrays in a special subroutine for initiation.

As I am learning Fortran08 by this programming project I discover new possibilities now and again. It seems possible to allow the initialization subroutine to do this dimensioning which means one may tailor the dimensioning of the arrays depending on the kind of calculation one will make.

Table 6: A very preliminary table with the state variables and their internal representation. Some model parameter properties are also included. The "z" used in some symbols like Sz means the optional normalizing symbol M, W, V or F.

Symbol	Id	Index 1	Index 2	Normalizing suffix	Meaning
			Intensive properties		
T	1	-	-	-	Temperature
P	2	-	-	-	Pressure
MU	3	component	-/phase	-	Chemical potential
AC	4	component	-/phase	-	Activity
LNAC	5	component	-/phase	-	LN(activity)
			Extensive properties		
U	10	-/phase#set	-	-	Internal energy for system
UM	11	-/phase#set	-	M	Internal energy per mole
UW	12	-/phase#set	-	W	Internal energy per mass
UV	13	-/phase#set	-	V	Internal energy per m^3
UF	14	phase#set	-	F	Internal energy per formula unit
Sz	2z	-/phase#set	-	-	entropy
Vz	3z	-/phase#set	-	-	volume
Hz	4z	-/phase#set	-	-	enthalpy
Az	5z	-/phase#set	-	-	Helmholtz energy
Gz	6z	-/phase#set	-	-	Gibbs energy
NPz	7z	phase#set	-	-	Moles of phase
BPz	8z	phase#set	-	-	Mass of phase
Qz	9z	phase#set	-	-	Stability of phase
DGz	10z	phase#set	-	-	Driving force of phase
Nz	11z	-/phase#set/comp	-/comp	-	Moles of component
X	111	phase#set/comp	-/comp	0	Mole fraction
X%	111	phase#set/comp	-/comp	100	Mole per cent
Bz	12z	-/phase#set/comp	-/comp	-	Mass of component
W	122	phase#set/comp	-/comp	0	Mass fraction
W%	122	phase#set/comp	-/comp	100	Mass per cent
Y	130	phase#set	const#subl	-	Constituent fraction
			Some model parameter identifiers		
TC	-	phase#set	-	-	Curie temperature
BMAG	-	phase#set	-	-	Aver. Bohr magneton number
MQ&X	-	phase#set	constituent	-	Mobility of X
LNTH	-	phase#set	-	-	LN(Einstein temperature)

3.1 User defined data types

The presentation of the data structures defined here follows the way they are declared in the Fortran source code. That may sometimes not be entirely logical but it makes it simpler to update the documentation.

In Fortran08 the programmer can define specific data types and several such are used to store the model information. A data type can contain many different kinds of variables like characters and numerical information. One can also have links between instances (records) of the same and different kinds of datatypes.

This kind of method to organize data is available in all modern computing languages and has been severely missed in Fortran. In Thermo-Calc it was handled by storing data in the self-designed workspace.

Some data types are straightforward to implement like the element. Thus one defines a data type which has all the necessary information for an element and then allocates an array to hold as many elements that is necessary.

Other datatypes are more involved, like the record to hold information about an endmember parameter of a phase. This record must refer to the constituents that makes up the endmember and also a link to a list of properties that can be calculated for this end member. Finally it must be possible to link the endmember record in a list for a specific phase and it must be possible to have a link to interaction records based on this endmember. Before one can figure out how to store endmember data it is thus necessary to have a data structure containing many different kinds of data. One must also consider which data that may be different in different parallel processes dealing with the phase and which data that are "static". Many of these records may also be of different size for different phases and for example an endmember record for a phase with 3 sublattices need 3 times as many places for constituent indices than a phase with a single lattice. This can be handled with the dynamic allocation feature in Fortran08.

The TYPE definition feature in Fortran08 may be a little less flexible than in other programming languages but it also avoids some of the strange features of these like the "this" operator needed at strange places in some cases. There is no "new" operator either, records are usually declared as single entities or as arrays with fixed (although allocatable) size. But one can use pointers to allocate new records which can be found only by these pointers.

Some of the structures are used to define arrays, like the element, species and phases. Some are created dynamically and can only be accessed by pointers, like endmembers and property records. Some are used locally inside subroutines and declared globally only if used in several.

3.2 Bits of information

Many records contain information that is of YES/NO or ON/OFF type and they are often stored as bits in a status word. Some of these are set automatically and there are subroutines and commands to change others. They are summarized below for all records.

The use of each bit is explained in the text from the source code.

```
!-Bits in GLOBAL status word (GS) in globaldata record
! level of user: beginner, occational, advanced; NOGLOB: no global gridmin calc
! NOMERGE: no merge of gridmin result,
! NODATA: not any data,
! NOPHASE: no phase in system,
! NOACS: no automatic creation of composition set for any phase
! NOREMCS: do not remove any redundant unstable composition sets
! NOSAVE: data changed after last save command
! VERBOSE: maximum of listing
! SETVERB: permanent setting of verbose
! SILENT: as little output as possible
! NOAFTEREQ: no manipulations of results after equilibrium calculation
! XGRID: extra dense grid for all phases
! NOPAR: do not run in parallel
! NOSMGLOB do not test global equilibrium at node points
! NOTELCOMP the elements are not the components
! TGRID use grid minimizer to test if global after calculating equilibrium
! OGRID use old grid generator
! NORECALC do not recalculate equilibria even if global test after fails
! OLDMAP use old map algorithm
! NOAUTOSP do not generate automatic start points for mapping
! GSYGRID extra dense grid
! GSVIRTUAL (CCI) enables calculations with a virtual element
! >>>> some of these should be moved to the gtp_equilibrium_data record
  integer, parameter :: &
       GSBEG=0,       GSOCC=1,        GSADV=2,      GSNOGLOB=3,  &
       GSNOMERGE=4,   GSNODATA=5,     GSNOPHASE=6,  GSNOACS=7,   &
       GSNOREMCS=8,   GSNOSAVE=9,     GSVERBOSE=10, GSSETVERB=11,&
       GSSILENT=12,   GSNOAFTEREQ=13, GSXGRID=14,   GSNOPAR=15,  &
       GSNOSMGLOB=16, GSNOTELCOMP=17, GSTGRID=18,   GSOGRID=19,  &
       GSNORECALC=20, GSOLDMAP=21,    GSNOAUTOSP=22,GSYGRID=23,  &
       GSVIRTUAL=24
!----------------------------------------------------------------
!-Bits in ELEMENT record
  integer, parameter :: &
       ELSUS=0,       ELDEL=1
!----------------------------------------------------------------
!-Bits in SPECIES record
```

```
! SUS   Suspended,
! IMSUS implicitly suspended (when element suspended)
! EL    species is element,
! VA    species is the vacancy
! ION   species have charge,
! SYS   species is (system) component
! UQAC  species used in uniquac model (2 extra reals for area and volume)
  integer, parameter :: &
       SPSUS=0, SPIMSUS=1, SPEL=2, SPVA=3, &
       SPION=4, SPSYS=5,   SPUQC=6
```

3.2.1 Phase record bits

The phase record has many bits specifying various things. Additionally the composition set record has other bits, most important if the composition set is ENTERED, meaning it can be stable, FIX, meaning it is a condition that it is stable, SUSPENDED, meaning it must not be stable, and DORMANT meaning it must not be stable but the driving force is calculated.

A user can change these bits but he/she cannot set the CSSTABLE bit which is only set if the phase is stable after a full equilibrium calculation.

```
!-Bits in PHASE record STATUS1 there are also bits in each phase_varres record!
! HID phase is hidden (not implemented)
! IMHID phase is implictly hidden (not implemented)
! ID phase is ideal, substitutional and no interaction
! NOCV phase has no concentration variation
! HASP phase has at least one parameter entered
! FORD phase has 4 sublattice FCC ordering with parameter permutations
! BORD phase has 4 sublattice BCC ordering with parameter permutations
! SORD phase has TCP type ordering (do not subract ordered as disordered, NEVER)
! MFS phase has a disordered fraction set
! GAS this is the gas phase (first in phase list)
! LIQ phase is liquid (can be several but listed directly after gas)
! IONLIQ phase has ionic liquid model (I2SL)
! AQ1 phase has aqueous model (not implemented)
! 2STATE elemental liquid twostate model parameter (not same as I2SL!)
! QCE phase has corrected quasichemical entropy (Hillerst-Selleby-Sundman)
! CVMCE phase has some CVM ordering entropy (not implemented)
! EXCB phase need explicit charge balance (has ions)
! XGRID use extra dense grid for this phase
! MQMQA (old FACTCE) phase has FACT quasichemical SRO model (not implemented)
! NOCS not allowed to create composition sets for this phase
! HELM parameters are for a Helmholz energy model (not implemented),
! PHNODGDY2 phase has model with no analytical 2nd derivatives
! not implemented ELMA phase has elastic model A (not implemented)
! EECLIQ this is the condensed phase (liquid) that should have highest entropy
! PHSUBO special use testing models DO NOT USE
! PALM interaction records numbered by PALMTREE NEEDED FOR PERMUTATIONS !!!
! MULTI may be used with care
! BMAV Xion magnetic model with average Bohr magneton number
! UNIQUAC The UNIQUAC fluid model
! TISR phase has the TSIR entropy model (E Kremer)
! SROT phase has the tetrahedral SRO
  integer, parameter :: &
       PHHID=0,     PHIMHID=1,    PHID=2,      PHNOCV=3, &     ! 1 2 4 8 : 0/F
       PHHASP=4,    PHFORD=5,     PHBORD=6,    PHSORD=7, &     !
       PHMFS=8,     PHGAS=9,      PHLIQ=10,    PHIONLIQ=11, &  !
       PHAQ1=12,    PH2STATE=13,  PHQCE=14,    PHCVMCE=15,&    !
       PHEXCB=16,   PHXGRID=17,   PHMQMQA=18,  PHNOCS=19,&     !
       PHHELM=20,   PHNODGDY2=21, PHEECLIQ=22, PHSUBO=23,&     !
       PHPALM=24,   PHMULTI=25,   PHBMAV=26,   PHUNIQUAC=27, & !
       PHTISR=28,   PHSSRO=29,    PHSROT=30                            !
!
!----------------------------------------------------------------
!-Bits in PHASE_VARRES (constituent fraction) record STATUS2
! CSDFS is set if record is for disordred fraction set, then one must use
!     sublattices from fraction_set record
! CSDLNK: a disordred fraction set in this phase_varres record
! CSDUM2 and CSDUM3 not used
! CSCONSUS set if one or more constituents suspended (status array constat
!     specify constituent status)
! CSORDER: set if fractions are ordered (only used for BCC/FCC ordering
!     with a disordered fraction set).
! CSABLE: set if phase is stable after an equilibrium calculation ?? needed
! CSAUTO set if composition set created during calculations
! CSDEFCON set if there is a default constitution
! CSTEMPAR set if created by grid minimizer and can be suspended afterwards
!       when running parallel
! CSDEL set if record is not used but has been and then deleted (by gridmin)
! CSADDG means there are terms to be added to G
! CSTEMPDOR means this compset was temporarily set dormant at an
!       equilibrium calculation
   integer, parameter :: &
        CSDFS=0,    CSDLNK=1,  CSDUM2=2,    CSDUM3=3, &
        CSCONSUS=4, CSORDER=5, CSABLE=6,    CSAUTO=7, &
        CSDEFCON=8, CSTEMPAR=9,CSDEL=10,    CSADDG=11,&
        CSTEMPDOR=12
```

3.2.2 Some more bits

In particular the gtp_equilibrium_data record there are some bits that specifies how the equilibrium calculation can be done, for example if the software can use global grid minimization to find start values.

For the parameter property record there are also bits specifying of the property may depend on T and P or if it has a constituent or component specifier (like the mobility).

```
!-Bits in CONSTAT array for each constituent
! For each constituent:
! SUS constituent is suspended (not implemented)
! IMSUS is implicitly suspended,
! VA is vacancy
! QCBOND the constituent is a binary quasichemical cluster
   integer, parameter :: &
        CONSUS=0,   CONIMSUS=1,  CONVA=2,    CONQCBOND=3
!----------------------------------------------------------------
!-Bits in STATE VARIABLE FUNCTIONS (svflista)
! SVFVAL V symbol evaluated only when explicitly referenced (mode=1 in call)
! SVFEXT X symbol value taken from equilibrium %eqnoval
! SVCONST C symbol is a constant (can be changed with AMEND)
! SVFTPF - bit not used, replaced by export/import
! SVFDOT D symbol is a DOT function, like cp=h.t (also SVFVAL bit)
! SVFNOAM N symbol cannot be amended (only R, RT and T_C)
! SVEXPORT E symbol value exported to assessment coeff (TP constant)
! SVIMPORT I symbol value imported from TP-function (incl assessment coeff)
! ONLY ONE BIT CAN BE SET except for D and C+I and C+E,
! OTHER COMBINATIONS ARE NOT ALLOWED!!
!
   integer, parameter :: &
        SVFVAL=0,     SVFEXT=1,     SVCONST=2,     SVFTPF=3,&
        SVFDOT=4,     SVNOAM=5,     SVEXPORT=6,    SVIMPORT=7
!----------------------------------------------------------------
!-Bits in CEQ record (gtp_equilibrium_data)
! EQNOTHREAD set if equilibrium must be calculated before threading
! (in assessment) for example if a symbol must be evaluated in this
! equilibrium before used in another like H(T)-H298
! EQNOGLOB set if no global minimization
! EQNOEQCAL set if no successful equilibrium calculation made
! EQINCON set if current conditions inconsistent with last calculation
! EQFAIL set if last calculation failed
! EQNOACS set if no automatic composition sets ?? not used !! see GSNOACS
! EQGRIDTEST set if grid minimizer should be used after equilibrium
! EQGRIDCAL set if last calculation was using only gridminimizer
! EQMIXED set if mixed reference state for the elements
```

```
   integer, parameter :: &
        EQNOTHREAD=0, EQNOGLOB=1, EQNOEQCAL=2,  EQINCON=3, &
        EQFAIL=4,     EQNOACS=5,  EQGRIDTEST=6, EQGRIDCAL=7, &
        EQMIXED=8
!-----------------------------------------------------------------
!-Bits in parameter property type record (gtp_propid)
! no T or P dependence (constant)
! only P dependence
! only T dependence
! there is an element suffix (like mobility),
! there is a constituent suffix
! Property has no addition (used when entering and listing data)
   integer, parameter :: &
        IDNOTP=0, IDONLYP=1, IDONLYT=2, IDELSUFFIX=3, IDCONSUFFIX=4,&
        IDNOADD=5
!-----------------------------------------------------------------
!- Bits in condition status word (some set in onther ways??)
! singlevar means T=, x(el)= etc, singlevalue means value is a number
! phase means the condition is a fix phase
  integer, parameter :: &
       ACTIVE=0, SINGLEVAR=1, SINGLEVALUE=2, PHASE=3
!-----------------------------------------------------------------
!- Bits in assessment head record status
! ahcoef set means coefficients are entered
  integer, parameter :: &
       AHCOEF=0
!
!-----------------------------------------------------------------
!- Bits in addition record status word gtp_phase_add
! havepar set if the phase has parameters for this addition
! if not set the addition is not listed
! permol set if addition should be muliplied with number of atoms
  integer, parameter :: &
       ADDHAVEPAR=0, ADDPERMOL=1,ADDBCCMAG=2
!
! >>> Bits for symbols and TP functions missing ???
```

3.2.3 Phase status revision

A phase status can vary, the default is ENTERED which means that the phase can be stable or not depending on the conditions set by the user. The status values are defined as symbols according to the list below:

```
! some constants, phase status
  integer, parameter :: EECDORM=-5
  integer, parameter :: PHHIDDEN=-4
  integer, parameter :: PHSUS=-3
  integer, parameter :: PHDORM=-2
  integer, parameter :: PHENTUNST=-1
  integer, parameter :: PHENTERED=0
  integer, parameter :: PHENTSTAB=1
  integer, parameter :: PHFIXED=2
  character (len=12), dimension(-5:2), parameter :: phstate=&
       (/'EEC_DORMANT ','HIDDEN      ','SUSPENDED   ','DORMANT     ',&
         'ENTERED UNST','ENTERED     ','ENTERED STBL','FIXED       '/)
```

The phase status are also available as character strings in the array phstate, defined above, for use in listings.

The HIDDEN status is questionable and not implemented. A phase with the HIDDEN status should not be included in the normal list of phases but stored in a separate list. If the status of a phase is changed to or from being HIDDEN the system should be re-initiated. The possibility to implement HIDDEN is kept as the current database interface does not allow selection of phases when reading from a database.

3.2.4 Predefined and user defined additions

The additions provided or added by programmers must be uniquely identified. This is an attempt to do that with those already available. The model parameter identifiers are explaned in section 1.5.2 and listed in Table 4.

```
! The number of additions to the Gibbs energy of a phase is increasing
! This is a way to try to organize them.  Each addtion has a unique
! number identifying it when created, listed or calculated.  These
! numbers are defined here
  integer, public, parameter :: INDENMAGNETIC=1
  integer, public, parameter :: XIONGMAGNETIC=2
  integer, public, parameter :: DEBYECP=3
  integer, public, parameter :: EINSTEINCP=4
  integer, public, parameter :: TWOSTATEMODEL1=5
  integer, public, parameter :: ELASTICMODEL1=6
  integer, public, parameter :: VOLMOD1=7
  integer, public, parameter :: UNUSED_CRYSTALBREAKDOWNMOD=8
  integer, public, parameter :: SECONDEINSTEIN=9
  integer, public, parameter :: SCHOTTKYANOMALY=10
  integer, public, parameter :: DIFFCOEFS=11
! with composition independent G2 parameter NOT USED
  integer, public, parameter :: TWOSTATEMODEL2=12
! name of additions:
  character(len=24) , public, dimension(12), parameter :: additioname=&
```

```
       ['Inden-Hillert magn model','Inden-Xiong magn model  ',&
       'Debye CP model          ','Einstein Cp model       ',&
       'Liquid 2-state model    ','Elastic model A         ',&
       'Volume model A          ','Unused CBT model        ',&
       'Smooth CP step          ','Schottky Anomaly        ',&
       'Diffusion coefficients  ','                        ']
!       123456789.123456789.1234   123456789.123456789.1234
! Note that additions often use extra parameters like Curie or Debye
! temperatures defined by model parameter identifiers stored in gtp_propid
```

3.3 The global error code

The error code is the only member of the gtp_parerr record. From verson 7 it is declared in the METLIB module. For parallel execution each thread must have its own value and it must be declared threadprivate.

4 Basic thermodynamic data structures

Below the data types in GTP will be explained and in some cases also how they are declared as arrays or inside other data types. Many of the declared variables are "private" which means they cannot be changed directly from outside the module. For such data that the user will manipulate there are subroutines or functions provided.

4.1 Global data

Information that is valid for the whole system is stored here. Note that T and P may be different in different parallel processes and if there is a prescribed value that is stored in a condition record. Maybe this is not really needed.

```
  TYPE gtp_global_data
! status should contain bits how advanced the user is and other defaults
! it also contain bits if new data can be entered (if more than one equilib)
! sysparam are variables for different things
! sysparam(1) unused
! sysparam(2) number of equilibria between each check of spinodal at STEP/MAP??
! sysparem(3-10) unused ...
! sysreal(1) is the minimum T for EET check (equi-entopy T, Hickel)
!           if zero no EET c
     integer status
     integer :: encrypted=0
     character name*24
     double precision rgas,rgasuser,pnorm
! these are explicitly set to zero in new_gtp
     double precision, dimension(10) :: sysreal=zero
     integer :: sysparam(10)=0
  END TYPE gtp_global_data
  TYPE(gtp_global_data) :: globaldata
```

4.2 Version identification of the data structure

As the data structure is always open to changes and as such changes may affect other parts of the software most of the TYPE definitions have a "version" value. Whenever a change is made in a TYPE definition this version value should be incremented. The version number can be tested in other parts of the software, in particular when writing the data structure on a file and at a later time reading it back from a file. If the version number is not the same it may not be possible to read the file or the software must have provisions for reading records with different version numbers.

4.3 The ELEMENT data type

Elements are the building blocks of other data. They have a symbol, mass, reference state and some other values defined.

The elements in a system are stored in an array ELLISTA of these records in the order they are entered. Another integer array ELEMENTS have the elements in alphabetical order. These arrays are allocated in the subroutine init_gtp.

There are two predefined elements, the electron with symbol "/-" and the vacancy with symbol "Va". They have the indices -1 and 0. All real elements have numbers from 1 and higher.

The OC software is case insensitive, UPPER and lower case letters are treated identically. Thus Va, va and VA is the same. Elements with a single letter symbol must be followed by a space or a non-alphabetic character, for example a stoichiometric factor, to separate it from a following element.

The advantage with ELLISTA is that the element is stored at an index which never change, it does not change when other elements are entered which would change the alphabetical order. Note that there is a cross index stored in ellista so it is possible to know the alphabetical order of the element.

It also means there are 3 ways to specify an element, its symbol, its alphabetical index (its index in ELEMENTS) and its index in ELLISTA. The ELLISTA index is most important as it will never change whereas the ELEMENTS index may change when new elements are entered which may change the alphabetical order. The link to an element from a species are always the ELLISTA index.

In the species record the stoichiometric formula is stored by giving the index of the element in ELLISTA. The elements array is useful to make alphabetically ordered listing of data. Elements are normally never deleted but may be suspended i.e. hidden from application programs (this cannot be allowed during parallel processing).

```
! this constant must be incremented whenever a change is made in gtp_element
  INTEGER, parameter :: gtp_element_version=1
  TYPE gtp_element
! data for each element: symbol, name, reference state, mass, h298-h0, s298
     character :: symbol*2,name*12,ref_state*24
     double precision :: mass,h298_h0,s298
! splink: index of corresponding species in array splink
! Status bits are stored in the integer status
! alphaindex: the alphabetical order of this elements
! refstatesymbol: indicates H0 (1), H298 (0, default) or G (2) for endmembers
     integer :: splink,status,alphaindex,refstatesymbol
  END TYPE gtp_element
! allocated in init_gtp
  TYPE(gtp_element), private, allocatable :: ellista(:)
  INTEGER, private, allocatable :: ELEMENTS(:)
```

4.4 Species

Here some of the useful features of Fortran08 data structuring appears and can be explained. The species have a name, mass and charge as fixed attributes. It also has a status word and a link back to the array where the species are arranged in alphabetical order. It has an integer giving the number of elements in the species (must be larger than zero). Finally there are two arrays which have no size specified in the declaration, these must be allocated when the species is entered. The reason to have arrays that can be allocated is that some species has just one element, other may have two, three or more. The dimension of these arrays are stored in noofel (It can actually also be obtained by the Fortran08 build-in function SIZE). The indices stored in ellinks are the location, i.e. index to ellista. This does not change even if the alphabetical order of elements is changed.

For saving and reading a stored datastructure from a file it is necessary to have the number of elements as a stored integer because this is needed to allocate space for these arrays before reading the content of these arrays from the file. The SIZE function is useless in that case.

One can declare arrays that should be allocated dynamically either as ALLOCATABLE or POINTER. Note that allocating POINTER variables is a likely source of memory leaks.

The species record is an interesting example of an array of structures that in itself contains allocatable arrays that can vary in size depending on the data stored there. This feature will be utilized extensively when storing data for phases.

For the UNIQUAC model an allocatable array, spextra, with double precision variables has been added in version 5 of OC. This allows storing the area and volume parameter associated with a component in the UNIQUAC model.

It is also a very interesting feature to use indices to other arrays as links rather than pointers which is necessary in most other languages with data structures. There are pointers also in Fortran08 but pointers have a bad habit of being confusing and complicated to use, one is never sure if one has the address of a pointer or the content (which is also a memory address).

The integer array "species" has the species in alphabetical order by providing the correct index to the SPLISTA.

```
! this constant must be incremented whenever a change is made in gtp_species
  INTEGER, parameter :: gtp_species_version=2
  TYPE gtp_species
! data for each species: symbol, mass, charge, extra, status
! mass is in principle redundant as calculated from element mass
     character :: symbol*24
     double precision :: mass,charge
! alphaindex: the alphabetical order of this species
! noofel: number of elements
! nextra: number of extra properties (size of spextra)
     integer :: noofel,status,alphaindex
! Use an integer array ellinks to indicate the elements in the species
! The corresponing stoichiometry is in the array stochiometry
     integer, dimension(:), allocatable :: ellinks
     double precision, dimension(:), allocatable :: stoichiometry
! Can be used for extra species properties as in UNIQUAC models (area, volume)
     double precision, dimension(:), allocatable :: spextra
  END TYPE gtp_species
! allocated in init_gtp
  TYPE(gtp_species), private, allocatable :: splista(:)
  INTEGER, private, allocatable :: SPECIES(:)
```

4.5 Components

It will be possible to define different components for each equilibrium. A component must be a species. Initially the species identical to the elements are the components. The components are part of the "gtp_equilibrium_data" record and after a calculation the chemical potentials are stored in these records.

```
! this constant must be incremented whenever a change is made in gtp_component
  INTEGER, parameter :: gtp_component_version=1
  TYPE gtp_components
! The components are simply an array of indices to species records
! the components must be "orthogonal".  There is always a set of "systems
! components" that by default is the elements.
! Later one may implement that the user can define a different "user set"
! and maybe also specific sets for each phase.
! The reference state is set as a phase and value of T and P.
! The name of the phase and its link and the link to the constituent is stored
! the endmember array is for the reference phase to calculate GREF
! The last calculated values of the chemical potentials (for user defined
! and default reference states) should be stored here.
! molat is the number of moles of components in the defined reference state
     integer :: splink,phlink,status
     character*16 :: refstate
     integer, dimension(:), allocatable :: endmember
     double precision, dimension(2) :: tpref
     double precision, dimension(2) :: chempot
     double precision mass,molat
  END TYPE gtp_components
! allocated in gtp_equilibrium_data
```

4.6 Phase datatypes

The data stored for a phase is very complex. All data are accessed from a phase "root" record (except when save/read from a file). The phase has one oe more phase_varres records which is stored in the gtp_equilibrium_data record. Each equilibrium has such a record and in these records the last calculated results are kept. The user can create several equilibria with different sets of conditions as thus the results will be different in each equilibrium. This is a way to handle parallel processing but it is also useful to store experimental data in assessments.

In the phase_varres record all fraction variables (and some additional information) that can be different for each parallel process is also stored. If the phase has two or more composition sets each of these has its own phase_varres record. The phase_varres records are an array inside the gtp_equilibrium_data record and each phase store the links to its composition sets by indices to this array. This is necessary as one can have several gtp_equilibrium_data records and one must have the same number of composition sets for a phase in all equilibria. Whenever new composition sets are created in parallel computing it must be done for all threads at the same time.

The model parameter data is the same for all equilibria, it is part of the "static" data structure. The root record of a phase has data and links to all necessary information listed below:

- The endmember record list. There can be two endmember lists, one for each fraction types.

- The phase_varres record representing a composition set. Each each phase kan have up to 9 composition sets. The phase_varres records are declared as an array in the gtp_equilibrium_data record because ech equilibrium must have a unique set of fractions and results. In this record there are also some status bits that may be different in different parallel processes like if the phase (composition set) is FIXED/ENTERED/etc.

  A phase may exist simultaneously with different constituent fractions like in a miscibility gap or when a phase can order. To identify a composition set one can use the number symbol "#" follwed by a digit. When the composition set is created one can also add a pre- and suffix.

  In the phase_varres record there are also arrays to store all calculated G and its first and second derivatives (also derivatives with respect to fractions). Calculated values of other properties like TC, V, MQ etc (and their derivatives) are also stored if there are parameters for these properties.

  For the ionic liquid model the number of sites depend on the constitution and thus there are also provisions to have the number of sites on each sublattice in this record.

- The addition list. This is a pointer to an addition record (there can be several linked sequentially). The only current addition record is used to specify the magnetic magnetic contribution and for other additions new subroutines must be written to enter, list and calculate the additions. See section 4.6.9.

These records may in turn point to other records, as already mentioned the endmember record has links to another endmember records and to an interaction records and both of these records has a link to a list of property records where there are links to TP functions. Some of these links are indexes to arrays of these records but for example the parameters stored in endmember, interactions and property records are created dynamically and can only be accessed by pointers. The property records have links to TP functions but these links are indices to the array of TP function structure. The reason to have an array for these is that there are separate arrays with the calculated results of the TP functions in the gtp_equilibrium_data record as these must be local to each equilibrium, whereas the function expressions are the same for all equilibria.

4.6.1 Permutations of constituents

A complication for endmembers and interaction records is to handle permutations of constituents when one has ordering. It is clumsy to store identical permutation of constituents (like A:A:A:B, A:A:B:A, A:B:A:A, B:A:A:A for a 4 sublattice FCC ordering) in separate endmembers but one should have a single endmember for all permutations. It is necessary that the software can handle such permutations for multicomponent systems, otherwise the databases must contain several 1000 permutations of identical values of the same parameter. A 4 component FCC ordered system with 4 sublattices have totally 256 endmembers but only 35 unique ones.

The permutations create complications also for the interaction records which depend on the order of constituents in the endmember. An interaction records for an interaction between A and B belonging to an end member A:A:A:A (which has no permutations) but there are 4 permutations of the interaction record (B can be in any of the 4 sublattices). So each combination must be considered.

The second order interaction A,B:A,C:A:A this has 3 additional permutations compared to the first order as the interaction with C can be placed in any of the 3 remaining sublattices. But the second order interaction A,B:A,B:A:A has a more complicated permutations. If the first interaction with B is in the first sublattice, the second interaction with B can be placed in any or the remaining 3 but if the first level interaction in B is in the second sublattice the second level interaction can only be in the third or forth. In Table 7 some permuations are shown.

Table 7: Possible permutations of some FCC endmembers and interaction parameters. In the table the component arrays within parenthesis represent a permutation. There are no permutations of a parameter G(FCC,A:A:A:A) but 4 permutations of the parameter L(FCC,A,B:A:A:A) and 6 permuatations of L(FCC,A,B:A,B:A:A). The 2nd level permutations are linked from the corresponding 1st level in order to increase the speed of calculation.

endmember	1st level interaction	2nd level interaction
A: A: A: A	A,B: A: A: A	A,B: A,B: A: A
		(A,B: A: A,B: A)
		(A,B: A: A: A,B)
	(A: A,B: A: A)	(A: A,B: A,B: A)
		(A: A.B: A: A,B)
	(A: A: A,B: A)	(A: A: A,B: A,B)
	(A: A: A: A,B)	no permutation
A: A: A: B	A,B: A: A: B	A,B: A,B: A: B
		(A,B: A: A,B: B)
	(A: A,B: A: B)	(A: A,B: A,B: B)
	(A: A: A,B: B)	no permutation
(A: A: B: A)	(A,B: A: B: A)	(A,B: A,B: B: A)
		(A,B: A: B: A,B)
	(A: A,B: B: A)	(A: A,B: B: A,B)
	(A: A: B: A,B)	no permutation
etc.		

Reciprocal parameters like L(fcc,A,B:A,B:A:A) are important to approximate the short range order contibutions to the Gibbs energy.

In order to enter and modify parameters that are stored as permutation it is essential to have the constituents in a strict order. Of course the user is allowed to enter the components

for each sublattice in any order but the program will rearrange them according to there rules, assuming a system like
(A B C D)(A B C D)(A B C D)(A B C D):

- The alphabetically first component will be in sublattice 1 and the other ordered in increasing order. For option B where the first two and the following two are different the order applies to each pair of sublattices. Thus a parameter entered as G(BCC4,D:B:C:A) will be sorted as G(A:C:B:A).
- For interactions the alphabetically first constituent will be included in the endmember. Interactions will be moved to the first possible sublattice. For reciprocal parameters with interactions in two sublattices and when the endmember component is the same the alphabetically first interacting component will come in a sublattice before the other. For example L(BCC,B:C:A,C:A,B) will be sorted as L(BCC,A,B:A,C:B:C)

Without such strict ordering of constituents it will not be possible to find a parameter in the data structure.

Table 8: Som more permutations of some endmembers and interaction records for BCC. In this case the endmember, 1st and 2nd interactions have the same number of permutations.

endmember	1st level interaction	2nd level interaction
A: B: A: C	A,B: B: A: C	A,B: B: A,C: C
(B: A: A: C)	(B: A,B: A: C)	(B: A,B: A,C: C)
(A: B: C: A)	(A,B: B: C: A)	(A,B: B: C: A,C)
(A: C: A: B)	(A: C: A,B: B)	(A,C: C: A,B: B)
(C: A: A: B)	(C: A: A,B: B)	(C: A,B: A,C: B)
(C: A: B: A)	(C: A: B: A,B)	(C: A,C: B : A,B)
etc.		

The permutations means some complications when entering parameters as links to the fractions of all permutations are created at that stage. It also means that all constituents must be ordered, always in alphabetical order, so it is possible to find a parameter if its value should be changed. When calculating with the permutations the data structure created when entering the parameter must contain information on the number of permutations inherited from the endmember up to the second level interaction. No interactions higher than the 2nd order are allowed for a phase with permutations.

Since the first release of OC most of the permutations for the tetrahedron 4 sublattice fcc (and hcp) ordering has been implemented. The 4 sublattice bcc permutations are more complex but are now also implemented. All endmember permutations and all first level level have been implemented but only a limted set of 2nd order interactions, i.e. all for binary and ternary system. See also sections 8.6.9 and 8.6.10

An ordered phase will in many cases have a disordered fraction set for the disordered state and there will be parameters which depend on these fractions. There can be endmembers and interaction parameters for each fraction set. The parameters for each fraction set will be evaluated separately and added together at the end, taking into account the chain rule for the derivatives.

4.6.2 Property types for different fraction sets

An endmember or interaction can have several property records, normally one for the Gibbs energy but also for Curie temperature, mobilities etc. But there can be endmembers and interaction records without property records just because some higher interaction record have a property record.

For mobilities one can use a special type of property that is specific to a constituent like MQ&FE. In a property record one must have an adjustable array for function links, for example when there are several RK terms, see 2.3.2. These links can be allocated dynamically and if there are new RK terms added or deleted this array can be extended or decreased dynamically. For each property record one can store a reference to the origin of the data.

I am not sure exactly how one should in listings identify parameters belonging to different fraction sets. In a SIGMA phase with 3 sublattices one will have endmember parameters for some like G(SIGMA,FE:CR:FE), which are multiplied with the site fractions of these constituents, $y_{1,\mathrm{Fe}}y_{2,\mathrm{Cr}}y_{3,\mathrm{Fe}}$. But we will also have a fraction set, which in this case, use the mole fractions of Fe and Cr,

$x_{\mathrm{Fe}} = (10y_{1,\mathrm{Fe}} + 4y_{2,\mathrm{Fe}} + 16y_{3,\mathrm{Fe}})/30$

An endmember parameter for this fraction set is currently specified as GD(SIGMA,FE) and an interaction as GD(SIGMA,CR,FE). If in the future one is interested to extend to several levels of fraction sets one should maybe use an integer specification like G#2(SIGMA,FE) etc. but that seems unlikely for the next 10 years at least. In any cases a property without specification will belong to the primary site fraction set which is the major one, all other fractions are calculated from the fractions in the primary fraction set.

For FCC ordering with interstitials the second fraction set will not be mole fractions but the sum of the site fractions of the substitutional sublattices

$z_{Fe} = 0.25\sum_{s=1,4} y_{s,\mathrm{Fe}}$

4.6.3 Selected_element_reference

With Thermo-Calc it is has been a little awkward that one may not read the normal reference phase for an element from the database. For example the gas phase may not be included for calculations with nitrogen present but one would like to have partial pressures or activities of nitrogen referenced to the gas.

An attempt to handle this has been made by entering by default a phase called "SELECT_ELEMENT_REFERENCE". This phase is always hidden and cannot be included in any calculation. Its location and index is 0 (zero) so it does not show up in the number of phases. But the user can list the data of this phase and also amend them. The idea is that a database will automatically enter the appropriate data for each element in this phase and in calculations this phase will be the default reference state for activities or enthalpies (at current temperature). One must have a phase and not just a TP function to handle elements with magnetic transitions. This reference phase is in fact a different phase for each element, one must be able to use different magnetic functions for different elements. All details for this are not worked out and it may not even be a very good idea.

4.6.4 The endmember record

As already described the phase record is a root of two lists of endmembers, one for each of the two possible fraction sets. The endmember records are allocated by pointers when needed and contains several links to other records:

- one link to the next endmember record,
- one link to an interaction record, the root of a binary tree,
- one link to a list of property records and
- links to one constituent in each sublattice

All the links except those to the constituents may be empty. The links to the constituents are not real links but an integer index of the array of constituents stored in the phase record (described later). The endmembers are always arranged in increasing value of these indices, in the order of the sublattices. The endmember properties are multiplied with the fractions of these constituents when a calculation is performed.

When the ordering options has been implemented, see 4.6.1, an endmember may have several permutations of the constituent indices. That is the reason to have two-dimensional arrays for the constituent indices.

One may have endmembers that does not depend on the constituent in a specific sublattice and this is represented by the fraction index for this sublattice is negative, usually -99. This is called a "wildcard" and represented by an asterix "*" when entering or listing the parameter, see section 2.3.4.

```
! this constant must be incremented whenever a change is made in gtp_endmember
  INTEGER, parameter :: gtp_endmember_version=1
  TYPE gtp_endmember
! end member parameter record, note ordered phases can have
! several permutations of fraction pointers like for B2: (Al:Fe) and (Fe:Al).
! There are links (i.e. indices) to next end member and to the interactio tree
! and to a list of property record
! The phase link is needed for SAVE/READ as one cannot know the number of
! sublattices otherwise.  One could just store nsl but a link back to the
! phase record might be useful in other cases.
! noofpermut: number of permutations (for ordered phases: (Al:Fe) and (Fe:Al)
! phaselink: index of phase record
! antalem: sequenial order of creation, not used for anything exept
!          for MQMQA it is the index in %contyp of endmember
! propointer: link to properties for this endmember
! nextem: link to next endmember
! intponter: root of interaction tree of parameters
! fraclinks: indices of fractions to be multiplied with the parameter
     integer :: noofpermut,phaselink,antalem
     TYPE(gtp_property), pointer :: propointer
     TYPE(gtp_endmember), pointer :: nextem
     TYPE(gtp_interaction), pointer :: intpointer
! there is at least one fraclinks per sublattice
! the second index of fraclinks is the permutation (normally only one)
! the first indec of fraclinks points to a fraction for each sublattice.
! The fractions are numbered sequentially independent of sublattices, a
! sigma phase with (FE:CR,MO:CR,FE,MO) has 6 fractions (incl one for FE in
! first sublattice) and the end member (FE:MO:CR) has the fraclinks 1,3,4
! This means these values can be used as index to the array with fractions.
! The actual species can be found via the sublattice record
!    integer, dimension(:,:), pointer :: fraclinks
     integer, dimension(:,:), allocatable :: fraclinks
  END TYPE gtp_endmember
! dynamically allocated when entering a parameter
```

4.6.5 The interaction record

The interaction records forms a binary tree starting from an endmember record. It thus has two pointers to other interaction records, one which is the *next* on the same interaction level, excluding the interacting constituent in the current record, and one to a *higher* level, including the current interacting constituent. In this way any level of interaction parameter can be defined. However, only the first two levels (binary and ternary) can be composition dependent, see 2.3.2.

There is also a link to a list of property records. This link can be empty if there is a link to a higher order interaction.

The interaction record specifies one additional constituent in a sublattice. The remaining constituents are given by the endmember record and any lower interaction records with a higher link to this one. Again one can have permutations of this interacting constituent in the different sublattices, see section 4.6.1.

The interaction records are always linked from the first possible endmember, i.e. that with the lowest set of constituent indices in all sublattices. If there are several levels of interactions the lowest interaction has the lowest constituent index. There is no ordering of interacting constituents on the same level.

```
! this constant must be incremented when a change is made in gtp_interaction
  INTEGER, parameter :: gtp_interaction_version=1
  TYPE gtp_interaction
! this record constitutes the parameter tree. There are links to NEXT
! interaction on the same level (i.e. replace current fraction) and
! to HIGHER interactions (i.e. includes current fraction)
! There can be several permutations of the interactions (both sublattice
! and fraction permuted, like interaction in B2 (Al:Al,Fe) and (Al,Fe:Al))
! The number of permutations of interactions can be the same, more or fewer
! comparared to the lower order parameter (endmember or other interaction).
! The necessary information is stored in noofip.  It is not easy to keep
! track of permutations during calculations, the smart way to store the last
! permutation calculated is in this record ... but that will not work for
! parallel calculations as this record is static ...
! status: may be useful eventually
! antalint: sequential number of interaction record, to follow the structure
! order: for permutations one must have a sequential number in each node
! propointer: link to properties for this parameter
! nextlink: link to interaction on same level (replace interaction)
! highlink: link to interaction on higher level (include this interaction)
! sublattice: (array of) sublattices with interaction fraction
! fraclink: (array of) index of fraction to be multiplied with this parameter
! noofip: (array of) number of permutations, see above.
     integer status,antalint,order
     TYPE(gtp_property), pointer :: propointer
     TYPE(gtp_interaction), pointer :: nextlink,highlink
     TYPE(gtp_tooprec), pointer :: tooprec
     integer, dimension(:), allocatable :: sublattice,fraclink,noofip
  END TYPE gtp_interaction
! allocated dynamically and linked from endmember records and other
! interaction records (in a binary tree)
```

For ternary extrapolation s there can be an addtitional Toop record specifying if the ternary extrapolation model between any 3 constituents should be Kohler or Toop, see section D.

```
! this constant must be incremented when a change is made in gtp_phasetuple
```

```
  INTEGER, parameter :: gtp_tooprec_version=1
  TYPE gtp_tooprec
! this indicates that an binary interaction parameter has Kohler/Toop model
! and which constituents involved.  A binary interaction can have a list of
! several gtp_tooprec records for each ternary system this record is involed.
! The binary fractions multiplied with the parameters will be calculated
! using this list of ternaries indicated by this list, see the documentation.
! const1, const2 and const3 are the constituents in alphabetical order
! (which is also the numerical order).
! toop is 0 if Kohler extrapolation, if 1, 2 or 3 it indicates the Toop element
! extra is used when listing data
! uniqid is a unique identification of the record, used for debugging
     integer toop,const1,const2,const3,extra,uniqid
! Each gtp_tooprec is part of 3 lists for binary interactions
! indicated by the 3 constituents in the gtp_tooprec record.
! next12 is the next link for the 2 (alphaetically) first constituents,
! next13 is the next link for first and third constituents
! next23 is the next link for the second and third constituents
! seq is a sequential link in the order the records created (try to fix bug)
     type(gtp_tooprec), pointer :: next12,next13,next23,seq
! this is very useful to obtain information in the calc_toop subroutine
     type(gtp_phase_varres), pointer :: phres
  end type gtp_tooprec
```

4.6.6 The property record

The endmembers and interaction records is the start of a list of property records. A property list can be empty if an endmember or interaction record is needed to specify some higher interaction.

In the property record there is a property index, see 1.5.1 and a *nextpr* link to another property record. One can have a link to a reference for the parameter (published paper or similar), see 4.6.7.

The actual value of the property is stored as an index to a TP function. These functions are stored in array and cannot be deleted (but they can be changed) so the index is a stable link. The calculated values of a property is stored in another array, local to the equilibrium record, see 5.2.5, as one may have different values in each thread in parallel processing or when running assessments, as these use different equilibrium records.

For binary and ternary interactions one can have a degree larger than zero which means there are several TP functions linked to this property. See 2.3.2.

```
! this constant must be incremented when a change is made in gtp_property
  INTEGER, parameter :: gtp_property_version=2
  TYPE gtp_property
! This is the property record.  The end member and interaction records
! have pointer to this.  Severall different properties can be linked
! from a parameter record like G, TC, BMAGN, VA, MQ etc.
! Some properties are connected to a constituent (or component?) like the
! mobility and also the Bohr mangneton number.
! Allocated as linked from endmembers and interaction records
! reference: can be used to indicate the source of the data
! refix: can be used to indicate the source of the data
! nextpr: link to next property record
! extra: TOOP and KOHLER can be implemented inside the property record
! proptype: type of propery, 1 is G, other see parameter property
! degree: if parameter has Redlich-Kister or similar degrees (powers)
! degreelink: indices of TP functions for different degrees (0-9)
! protect: can be used to prevent listing of the parameter
! antalprop: probably redundant (from the time of arrays of propery records)
     character*16 reference
! this added to avoid problems if model param id has changed between saving
! and reading an unformatted file
     character*4 modelparamid
     TYPE(gtp_property), pointer :: nextpr
     integer proptype,degree,extra,protect,refix,antalprop
     integer, dimension(:), allocatable :: degreelink
  END TYPE gtp_property
! property records, linked from endmember and interaction records, allocated
! when needed.  Each propery like G, TC, has a property record linking
! a TPFUN record (by index to tpfun_parres)
```

4.6.7 Bibliographic references

It is important to document the source of the data and each property can have a unique bibliographic reference stored in this record. It would typically be a published paper or some internal written documentation.

```
! this constant must be incremented when a change is made in gtp_biblioref
! old name gtp_datareference
  INTEGER, parameter :: gtp_biblioref_version=1
  TYPE gtp_biblioref
! store data references
! reference: can be used for search of reference
! refspec: free text
     character*16 reference
!     character*64, dimension(:), allocatable :: refspec
! this is Fortran 2003/2008 standard, not available in GNU 4.8
!     character(len=:), allocatable :: nyrefspec
! Use wpack routines!!!
     integer, dimension(:), allocatable :: wprefspec
  END TYPE gtp_biblioref
! allocated in init_gtp
```

```
  TYPE(gtp_biblioref), private, allocatable :: bibrefs(:)
```

4.6.8 Parameter property identification

The propery list has an index and the meaning of this index is given the gtp_propid record. A few of these are predefined but a programmer can add more, see section 1.5.2. For any added property software must also be written to handle such properties during calculations. The implemented properties can be listed, see 8.8.24 and the values of these properties can be listed similarly like for state variables, see 8.8.25.

```
! this constant must be incremented when a change is made in gtp_propid
  INTEGER, parameter :: gtp_propid_version=1
  TYPE gtp_propid
! this identifies different properties that can depend on composition
! Property 1 is the Gibbs energy and the others are usually used in
! some function to contribute to the Gibbs energy like TC or BMAGN
! But one can also have properties used for other things like mobilities
! with additional especification like MQ&FE
! symbol: property identifier like G for Gibbs energy
! note: short description for listings
! prop_elsymb: additional for element dependent properties like mobilities
     character symbol*4,note*28,prop_elsymb*2
! Each property has a unique value of idprop.  Status can state if a property
! has a constituent specifier or if it can depend on T or P
     integer status
! this can be a constituent specification for Bohr mangetons or mobilities
! such specification is stored in the property record, not here
!    integer prop_spec,listid
! >>> added "listid" as a conection to the "state variable" listing here.
! This replaces TC, BMAG, MQ etc included as "state variables" in order to
! list their values.  In this way all propids become available
  end TYPE gtp_propid
! the value TYPTY stored in property records is "idprop" or
! if IDELSUFFIX set then 100*"idprop"+ellista index of element
! if IDCONSUFFIX set then 100*"idprop"+constituent index
! When the parameter is read the suffix symbol is translated to the
! current element or constituent index
  TYPE(gtp_propid), dimension(:), private, allocatable :: propid
```

4.6.9 Additions to the Gibbs energy

Different contributions to the Gibbs energy can be specified in the software using the data structure below. Such additions normally depend on a number of composition dependent properties like the Curie temperature, the Bohr magneton number, the Einstein temperature etc. These can be added as properties and entered as endmember and interaction parameters with a unique index and symbol see 4.6.8.

When implementing a new addition subroutines must be written for entering, listing and calculating the property, including analytical first and second derivatives with respect to T, P and the constitution.

One may also add properties that does not contribute to the Gibbs energy but depend on the constitution of the phase, like mobilities, viscosities, resistivity etc. Such properties are calculated together with the Gibbs energy and their values can be extracted by appropriate subroutines.

As mentioned in section 1.5 most addition model calculate a Gibbs energy contribution per atom whereas the Gibbs energy model for the phase need the value per mole formula unit. There is a bit ADDPERMOL for this in the status word for additions defined in section 3.2.2.

```
! this constant must be incremented when a change is made in gtp_phase_add
  INTEGER, parameter :: gtp_phase_add_version=2
  TYPE gtp_phase_add
! record for additions to the Gibbs energy for a phase like magnetism
! addrecno: ?
! aff: antiferomagnetic factor (Inden model)
! constants: for some constants needed ?? NEW
! status: BIT 0 set if there are parameters
!         BIT 1 set if magnetic model is for BCC
! need_property: depend on these properties (like Curie T)
! explink: function to calculate with the properties it need (not allocatable?)
! nextadd: link to another addition
     integer type,addrecno,aff,status
     integer, dimension(:), allocatable :: need_property
     double precision, dimension(:), allocatable :: constants
     TYPE(tpfun_expression), dimension(:), pointer :: explink
! The following declaration is illegal ... but above OK and I can allocate
!     TYPE(tpfun_expression), dimension(:), allocatable, pointer :: explink
     TYPE(gtp_phase_add), pointer :: nextadd
     type(gtp_elastic_modela), pointer :: elastica
     type(gtp_diffusion_model), pointer :: diffcoefs
! calculated contribution to G, G.T, G.P, G.T.T, G.T.P and G.P.P
     double precision, dimension(6) :: propval
  END TYPE gtp_phase_add
! allocated when needed and linked from phase record
```

4.6.10 The elastic model

This is a tentative record to model elastic contributions to the Gibbs energy.

```
! addition record to calculate the elastic energy contribution
! declared as allocatable in gtp_phase_add
! this constant must be incremented when a change is made in gtp_elastic_modela
  INTEGER, parameter :: gtp_elastic_modela_version=1
  TYPE gtp_elastic_modela
! lattice parameters (configuration) in 3 dimensions
     double precision, dimension(3,3) :: latticepar
! epsilon in Voigt notation
     double precision, dimension(6) :: epsa
! elastic constant matrix in Voigt notation
     double precision, dimension(6,6) :: cmat
! calculated elastic energy addition (with derivative to T and P?)
     double precision, dimension(6) :: eeadd
! maybe more ...
  end TYPE gtp_elastic_modela
```

4.6.11 The diffusion coefficients

This is a record to handle mobilities and diffusion coefficients. So far it has not been used in any application.

```
! addition record to calculate diffusion coefficients
! declared as allocatable in gtp_phase_add
! this constant must be incremented when a change is made in gtp_elastic_modela
  INTEGER, parameter :: gtp_diffusion_model_version=1
  TYPE gtp_diffusion_model
! status bit 0 set means no calculation of this record
! dilute, simple or magnetic
     integer difftypemodel,status
!  alpha values for magnetic diffusion (for interstitials in constituent order)
     double precision, allocatable, dimension(:) :: alpha
! indices of dependent constituent in each sublattices
     integer, allocatable, dimension(:) :: depcon
! indices of constituents with zerovolume
     integer, allocatable, dimension(:) :: zvcon
! calculated diffusion matrix
     double precision, allocatable, dimension(:,:) :: dcoef
! Maybe we need one for each composition set?? at least to save the matrix
     type(gtp_diffusion_model), pointer :: nextcompset
! maybe more ...
  end TYPE gtp_diffusion_model
```

4.6.12 Data structures to transform TDB to DAT files

With OC it is possible to convert some TDB files to a SOLGASMIX dat file. These are data structures used for that. There are known bugs in the conversion and if you have problems please pay for a commercial converter.

```
  TYPE gtp_tpfun_as_coeff
! this is a TPFUN converted to coefficents without any references to other
! functions.  Each function can have several T ranges and coefficents for T**n
! USED FOR SOLGASMIX
     double precision, dimension(:), allocatable :: tbreaks
     double precision, dimension(:,:), allocatable :: coefs
     integer, dimension(:,:), allocatable :: tpows
! this is used only during conversion
!     type(gtp_tpfun_as_coeff), pointer :: nextcrec
  end type gtp_tpfun_as_coeff
!
!-----------------------------------------------------------------------------
  INTEGER, parameter :: gtp_tpfun2dat_version=1
  TYPE gtp_tpfun2dat
! this is a temporary storage of TP functions converted to arrays of
! coefficients.  Allocated as an array when necessary and the index in
! this array is the same index as for the TPfun
! USED FOR SOLGASMIX calculations and it is very messily implemented
! if debug is nonzero there is additional output and name is displayed
     integer nranges,debug
!     type(gtp_tpfun_as_coeff) :: tpfuncoef
     type(gtp_tpfun_as_coeff) :: cfun
     character*16 :: name
  end type gtp_tpfun2dat
```

4.6.13 The phase and composition set indices

In many cases one must specify both a phase and a composition set and this is done by separate indices in most subroutine calls. As this is not very convenient an attempt to introduce a single index for both phase and composition sets has been med by the "phase tuple" type. This is a simple record with five integers, ixphase is the index of the phase in alphabetical order, compset the composition set index, phaseix is the index of the phase in the phlista array, lokvares is the index of the phase_varres record which is useful to retrieve data. If there is a higher composition set for the same phase then nextcs is the phase tuple index to the phase, otherwise nextcs is zero. Ixphase is probably redundant. There is an array with the phasetuple records called PHASETUPLE which is updated whenever a composition set is created or deleted.

```
! this constant must be incremented when a change is made in gtp_phasetuple
```

```
  INTEGER, parameter :: gtp_phasetuple_version=1
  TYPE gtp_phasetuple
! for handling a single array with phases and composition sets
! ixphase is phase index (often lokph), compset is composition set index
! ADDED also index in phlista (lokph) and phase_varres (lokvares) and
! nextcs which is nonzero if there is a higher composition set of the phase
! A tuplet index always refer to the same phase+compset.  New tuples with
! the same phase and other compsets are added at the end.
! BUT if a compset>1 is deleted tuples with higher index will be shifted down!
! CONFUSING ixphase is usually iph, phases in alphabetical order in phases
!           lokph is usually lokph, location in phlista
     integer lokph,compset,ixphase,lokvares,nextcs
! >>>>>>>>>>>> old     integer phaseix,compset,ixphase,lokvares,nextcs
  end TYPE gtp_phasetuple
```

4.6.14 The phase record

We have finally reached the phase record itself. As this refers to many of the structures above it is declared in the source code after all of them and to simplify the updating of this documentation we follow the order of the declarations in the source code. It is complicated enough already ...

The phase record had originally two parts but these have been merged to a single record. The link to the composition set data (the phase_varres record) has also recently been changed so the link to all of them are stored in the phase record as an array of indices to the array of phase_varres records in the equilibrium record. The phase_varres record contains all *dynamic* data that change during iterations and has also all calculated results. The endmember and interaction records contain the *static* data that does not change (except the calculated values of all TP functions9 but this is also stored the equilibrium record in the tp_res array.

When a phase is created a link to a record in the phase_varres array, i.e. an integer index, is stored in the array linktocs(1). If additional composition sets are created they are stored sequentially in the same array. A phase cannot have more than 9 composition sets.

```
! a smart way to have an array of pointers used in gtp_phase
  TYPE endmemrecarray
     type(gtp_endmember), pointer :: p1
  end TYPE endmemrecarray
!-----------------------------------------------------------------
! this constant must be incremented when a change is made in gtp_phase
  INTEGER, parameter :: gtp_phase_version=1
  TYPE gtp_phaserecord
! this is the record for phase model data. It points to many other records.
! Phases are stored in order of creation in phlista(i) and can be found
! in alphabetical order through the array phases(i)
! sublista is now removed and all data included in phlista
! sublattice and constituent data (they should be merged)
! The constituent link is the index to the splista(i), same function
! as LOKSP in iws.  Species in alphabetcal order is in species(i)
! One can allocate a dynamic array for the constituent list, done
! by subroutine create_constitlist.
! Note that the phase has a dynamic status word status2 in gtp_phase_varres
! which can be differnt in different parallel calculations.
! This status word has the FIX/ENT/SUS/DORM status bits for example
! name: phase name, note composition sets can have pre and suffixes
! model: free text
! phletter: G for gas, L for liquid
! alphaindex: the alphabetcal order of the phase (gas and liquids first)
     character name*24,models*72,phletter*1
     integer status1,alphaindex
! noofcs: number of composition sets,
! nooffs: number of fraction sets (replaces partitioned phases in TC)
     integer noofcs,nooffs
! additions: link to addition record list
! ordered: link to endmember record list
! disordered: link to endmember list for disordered fractions (if any)
     TYPE(gtp_phase_add), pointer :: additions
     TYPE(gtp_endmember), pointer :: ordered,disordered
! To allow parallel processing of endmembers, store a pointer to each here
     integer noemr,ndemr
     TYPE(endmemrecarray), dimension(:), allocatable :: oendmemarr,dendmemarr
! noofsubl: number if sublattices
! tnooffr: total number of fractions (constituents)
! linktocs: array with indices to phase_varres records
! nooffr: array with number of constituents in each sublattice
! Note that sites are stored in phase_varres as they may vary with the
! constitution for ionic liquid)
     integer noofsubl,tnooffr
     integer, dimension(9) :: linktocs
     integer, dimension(:), allocatable :: nooffr
! number of sites in phase_varres record as it can vary with composition
! constitlist: indices of species that are constituents (in all sublattices)
     integer, dimension(:), allocatable :: constitlist
! used in ionic liquid:
! i2slx(1) is index of Va, i2slx(2) is index if last anion (both can be zero)
     integer, dimension(2) :: i2slx
! allocated in init_gtp.
  END TYPE gtp_phaserecord
! NOTE phase with index 0 is the reference phase for the elements
! allocated in init_gtp
  TYPE(gtp_phaserecord), private, allocatable :: phlista(:)
  INTEGER, private, allocatable :: PHASES(:)
```

4.7 State variables and the state variable record

Conditions and results are obtained as values of state variables. There are many of these like $G, H, T, x(< \text{component} >)$ etc. They are stored in the software as a record of the type below where istv is an integer giving the basic type and indices can give additional specification. Some properties like chemical potentials can have a reference state and one can also define a unit like Kelvin or calories. At present this is used only to specify if a composition variable is a fraction or a percent.

```
! this constant must be incremented when a change is made in gtp_state_variable
  INTEGER, parameter :: gtp_state_variable_version=1
  TYPE gtp_state_variable
! this is to specify a formal or real argument to a function of state variables
! statevarid/istv: state variable index >=9 is extensive
! phref/iref: if a specified reference state (for chemical potential
! unit/iunit: 100 for percent, no other defined at present
! argtyp together with the next 4 integers represent the indices(4), only 0-4
! argtyp=0: no indices (T or P)
! argtyp=1: component
! argtyp=2: phase and compset
! argtyp=3: phase and compset and component
! argtyp=4: phase and compset and constituent
! ?? what is norm ?? normalizing like M in HM ?
     integer statevarid,norm,unit,phref,argtyp
! these integers represent the previous indices(4)
     integer phase,compset,component,constituent
! a state variable can be part of an expression with coefficients
! the coefficient can be stored here.  Default value is unity.
! In many cases it is ignored
     double precision coeff
! NOTE this is also used to store a condition of a fix phase
! In such a case statev is negative and the absolute value of statev
! is the phase index.  The phase and compset indices are also stored in
! "phase" and "compset" ??
! This is a temporary storage of the old state variable identifier
     integer oldstv
  end TYPE gtp_state_variable
! used for state variables/properties in various subroutines
```

5 Use of the thermodynamic data structures

This section describes how the basic thermodynamic data structures are used in various applications like calculating the Gibbs energy, handling errors, step for property diagrams, mapping of phase diagram, assessments etc.

5.1 Error handling

There is a global error code defined as part of the defined in the TYPE gtp_parerr. This has just one integer variable called bmperr and there is one variable of this type called gx which is decleared THREADPRIVATE for use in parallel processing. It is declared in the METLIB package.

The error code gx%bmperr is used in the whole OC system. Whenever there is an error gxsubroutine raising the error and usually the subroutine is exited directly. The error code should be tested after each subroutine call because the calling routine may be able to handle the error.

There are error messages defined for the errors generated by GTP.

5.2 Calculations

So far we have described how to handle the data for the phase. In order to calculate an equilibrium many values, like the constituent fractions, amounts of phase, temperature etc may change. As mentioned several times we must also be able to handle several separate equilibria, either as threads in parallel computing or as experimental data in assessments. Each of these separate datasets are stored in a gtp_equilibrium_data record.

5.2.1 Conditions

We must also handle conditions set by the user. Typically a condition is a state variable equal to a value like T=1273 or w(c)=0.01. At present simple assignments of state variables are the main types of conditions allowed but for phase compositions it is possible to use expressions like x(liq,fe)-x(bcc,fe)=0 to find a congruent and to set a state variables equal to a symbolic value like w(liq,b)=whatever.

```
! this constant must be incremented when a change is made in gtp_condition
! NOTE on unformatted SAVE files the conditions are written as texts
  INTEGER, parameter :: gtp_condition_version=1
  TYPE gtp_condition
! these records form a circular list linked from gtp_equilibrium_data records
! each record contains a condition to be used for calculation
! it is a state variable equation or a phase to be fixed
! The state variable is stored as an integer with indices
! NOTE: some state variables cannot be used as conditions: Q=18, DG=19, 25, 26
! There can be several terms in a condition (like x(liq,c)-x(fcc,c)=0)
! noofterms: number of terms in condition expression
! statev: the type of state variable (must be the same in all terms)
!          negative value of statev means phase index for fix phase
! active: zero if condition is active, nonzero for other cases
! unit: is 100 if value in percent, can also be used for temperature unit etc.
! nid: identification sequential number (in order of creation), redundant
! iref: part of the state variable (iref can be comp.set number)
! iunit: ? confused with unit?
! seqz is a sequential index of conditions, used for axis variables
! experimettype: inequality (< 0 or > 0) and/or percentage (-101, 100 or 101)
! symlink: index of symbol for prescribed value (1) and uncertainty (2)
! condcoeff: there is a coefficient and set of indices for each term
! prescribed: the prescribed value
! NOTE: if there is a symlink value that is the prescribed value
! current: the current value (not used?)
! uncertainty: the uncertainty (for experiments)
     integer :: noofterms,statev,active,iunit,nid,iref,seqz,experimenttype
!    TYPE(putfun_node), pointer :: symlink1,symlink2
! better to let condition symbol be index in svflista array
     integer symlink1,symlink2
     integer, dimension(:,:), allocatable :: indices
     double precision, dimension(:), allocatable :: condcoeff
     double precision prescribed, current, uncertainty
! confusing with record statevar and integer statev
     TYPE(gtp_state_variable), dimension(:), allocatable :: statvar
     TYPE(gtp_condition), pointer :: next, previous
  end TYPE gtp_condition
! used inside the gtp_equilibrium_data record and elsewhere
```

5.2.2 State variable functions

In this record the description of a state variable function is stored. The actual expression is stored using the PUTFUN subroutine in the metlib package. The calculated results of a state variable function is stored as a double precision array svfunres in the equilibrium data record. State variable function values are a single value, not 6 as for the TPfuns, as one cannot calculate a derivative with respect to anything.

The "dot" derivative expression available in Thermo-Calc has been implemented for one kind of calculations, $H.T$ meaning the partial derivative of enthalpy with respect to temperature, normally this is the heat capacity. Such a property can only be calculated in connection with an equilibrium calculation or a property diagram.

```
! this constant must be incremented when a change is made in gtp_putfun_lista
  INTEGER, parameter :: gtp_putfun_lista_version=2
  TYPE gtp_putfun_lista
! these are records for STATE VARIABLE FUNCTIONS.  The function itself
! is handelled by the putfun package.
! linkpnode: pointer to start node of putfun expression
! narg: number of symbols in the function
! nactarg: number of actual parameter specifications needed in call
!   (like @P, @C and @S
! status: can be used for various things
! status bit SVFVAL set means value evaluated only when called with mode=1
! SVCONST bit set if symbol is just a constant value (linknode is zero)
! eqnoval: used to specify the equilibrium the value should be taken from
!    (for handling what is called "variables" in TC, SVFEXT set also)
! SVFTPF set if symbol is a TP function, eqnoval is TPFUN index
! if SVIMPORT set then the symbol is set equal to a TP function (only value
!     no derivatives).  TP function index is in TPLINK
! if SVEXPORT set the the value of the symbol is copied to a TP function
!     (must be a constant).   TP function index is in TPLINK
! name: name of symbol
     integer narg,nactarg,status,eqnoval,tplink
     type(putfun_node), pointer :: linkpnode
     character name*16
! THIS IS OLY USED FOR CONSTANTS, VALUES ARE ALSO STORED IN CEQ%SVFUNRES
     double precision svfv
! this array has identification of state variable (and other function) symbols
! It is allocated in various subroutines, maybe be allocatable? 2020-08-31/BoS
     integer, dimension(:,:), pointer :: formal_arguments
  end TYPE gtp_putfun_lista
! this is the global array with state variable functions, "symbols"
  TYPE(gtp_putfun_lista), dimension(:), allocatable :: svflista
! NOTE the value of a function is stored locally in each equilibrium record
! in array svfunres.
! The number of entered state variable functions. Used when a new one stored
  integer, private :: nsvfun
```

5.2.3 Fraction sets

A phase can have several composition sets, meaning that it can be stable with two or more different compositions. A phase can also have two fraction sets, explained in more detail in 4.6.2.

```
! this constant must be incremented when a change is made in gtp_fraction_set
  INTEGER, parameter :: gtp_fraction_set_version=1
  TYPE gtp_fraction_set
```

```
! info about disordered fractions for some phases like ordered fcc, sigma etc
! latd: the number of sublattices added to first disordred sublattice
! ndd: sublattices for this fraction set,
! tnoofxfr: number of disordered fractions
! tnoofyfr: same for ordered fractions (=same as in phlista).
! varreslink: index of disordered phase_varres,
! totdis: 0 indicates no total disorder (sigma), 1=fcc, bcc or hcp
! id: parameter suffix, D for disordered
! dsites: number of sites in sublattices, disordred fractions stored in
!    another phase_varres record with index varreslink (above)
! splink: indices of species record for the constituents
! nooffr: the number of fractions in each sublattice
! y2x: the conversion from sublattice constituents to disordered and
! dxidyj: are the the coeff to multiply the y fractions to get the disordered
!        xfra(y2x(i))=xfra(y2x(i))+dxidyj(i)*yfra(i)
! disordered fractions stored in the phase_varres record with index varreslink
! arrays originally declared as pointers now changed to allocatable
     integer latd,ndd,tnoofxfr,tnoofyfr,varreslink,totdis
     character*1 id
     double precision, dimension(:), allocatable :: dsites
     integer, dimension(:), allocatable :: nooffr
     integer, dimension(:), allocatable :: splink
     integer, dimension(:), allocatable :: y2x
     double precision, dimension(:), allocatable :: dxidyj
! formula unit factor needed when calculating G for disordered sigma etc
     double precision fsites
  END TYPE gtp_fraction_set
! these records are declared in the phase_varres record as DISFRA for
! each composition set and linked from the phase_varres record
```

5.2.4 The phase_varres record for composition sets

Each composition set of a phase has a record as described below. It contains all data that can vary during calculations like the constituent fractions, the amount of the phase, etc. It also contains all results from a calculation. An array phase_varres is allocated in the gtp_equilibrium_record for this purpose and in the phase record there are indices to the phase_varres records for its composition sets.

As composition sets can be created and deleted there is a free list maintained using the integer *nextfree*. The first free phase_varres record is given by the global variable csfree. This list is maintained in the first equilibrium record, which is pointed to by the global variable FIRSTEQ.

```
! this constant must be incremented when a change is made in gtp_phase_varres
! added quasichemical bonds
  INTEGER, parameter :: gtp_phase_varres_version=2
  TYPE gtp_phase_varres
```

```
! Data here must be different in equilibria representing different experiments
! or calculated in parallel or results saved from step or map.
! nextfree: In unused phase_varres record it is the index to next free record
!    The global integer csfree is the index of the first free record
!    The global integer highcs is the higest varres index used
! phlink: is index of phase record for this phase_varres record
! status2: has composition set status bits CSxyz
! phstate: indicate state: fix/stable/entered/unknown/dormant/suspended/hidden
!                           2   1      0        -1      -2      -3       -4
! phtupx: phase tuple index
     integer nextfree,phlink,status2,phstate,phtupx
! abnorm(1): moles of components per formula unit of the phase/composition set
! abnorm(2): mass of components per formula unit
! abnorm(3): moles atoms per formula unit (all abnorm set by SET_CONSTITUTION)
! prefix and suffix are added to the name for composition sets 2 and higher
     double precision, dimension(3) :: abnorm
     character*4 prefix,suffix
! constat: array with status word for each constituent, any can be suspended
! yfr: the site fraction array
! mmyfr: min/max fractions, negative is a minumum
! sites: site ratios (which can vary for ionic liquids)
     integer, dimension(:), allocatable :: constat
     double precision, dimension(:), allocatable :: yfr
     real, dimension(:), allocatable :: mmyfr
     double precision, dimension(:), allocatable :: sites
! for ionic liquid derivatives of sites wrt fractions (it is the charge),
! 2nd derivates only when one constituent is vacancy
! 1st sublattice P=\sum_j (-v_j)*y_j + Qy_Va
! 2nd sublattice Q=\sum_i v_i*y_i
! dpqdy is the abs(valency) of the species, set in set_constitution
! for the vacancy it is the same as the number of sites on second subl.
! used in the minimizer and maybe elsewhere
     double precision, dimension(:), allocatable :: dpqdy
     double precision, dimension(:), allocatable :: d2pqdvay
! disfra: a structure describing the disordered fraction set (if any)
! for extra fraction sets, better to go via phase record index above
! this TYPE(gtp_fraction_set) variable is a bit messy.  Declaring it in this
! way means the record is stored inside this record.
     type(gtp_fraction_set) :: disfra
! this is for saving fractions in the mqmqa liquid model
     type(gtp_mqmqa_var) :: mqmqaf
! complier error when target added when arrays allocated to pointer
!  disapperad after subroutine return
!     type(gtp_mqmqa_var), target :: mqmqaf
! ---
! stored calculated results for each phase (composition set)
! amfu: is amount formula units of the composition set (calculated result)
! netcharge: is net charge of phase
! dgm: driving force
! qcbonds: quasichemical bonds (NOT SAVED ON UNFORMATTED)
     double precision amfu,netcharge,dgm,qcbonds
! qcsro: current value of SRO (for quasichemical model)
     double precision, allocatable, dimension(:) :: qcsro
! Other properties may be that: gval(*,2) is TC, (*,3) is BMAG, see listprop
! nprop: the number of different properties (set in allocate)
! listprop(1): is number of calculated properties
! listprop(2:listprop(1)): identifies the property stored in gval(1,ipy) etc
!   2=TC, 3=BMAG. Properties defined in the gtp_propid record
     integer nprop
     integer, dimension(:), allocatable :: listprop
! gval etc are for all composition dependent properties, gval(*,1) for G
! gval(*,1): is G, G.T, G.P, G.T.T, G.T.P and G.P.P
! dgval(1,j,1): is first derivatives of G wrt fractions j
! dgval(2,j,1): is second derivatives of G wrt fractions j and T
! dgval(3,j,1): is second derivatives of G wrt fractions j and P
! d2gval(ixsym(i,j),1): is second derivatives of G wrt fractions i and j
     double precision, dimension(:,:), allocatable :: gval
     double precision, dimension(:,:,:), allocatable :: dgval
     double precision, dimension(:,:), allocatable :: d2gval
! added for strain/stress, current values of lattice parameters
     double precision, dimension(3,3) :: curlat
! saved values from last equilibrium for dot derivative calculations
     double precision, dimension(:,:), allocatable :: cinvy
     double precision, dimension(:), allocatable :: cxmol
     double precision, dimension(:,:), allocatable :: cdxmol
! terms added to G if bit CSADDG nonzero
     double precision, dimension(:), allocatable :: addg
! integer containing the iteration when invsaved updated
     integer invsavediter
! arrays to save time in calc_dgdyterms, do not need to be saved on unformatted
     double precision, dimension(:,:), allocatable ::invsaved
  END TYPE gtp_phase_varres
! this record is created inside the gtp_equilibrium_data record
```

5.2.5 The equilibrium record

The equilibrium record has all data that may change dynamically during a calculation. One may have several equilibrium records and during parallel computing each thread must have one. Also during assessments each experimental data is stored in a separate equilibrium record as it has its unique set of conditions.

```
! this must be incremented when a change is made in gtp_equilibrium_data
  INTEGER, parameter :: gtp_equilibrium_data_version=1
  TYPE gtp_equilibrium_data
! this contains all data specific to an equilibrium like conditions,
! status, constitution and calculated values of all phases etc
! Several equilibria may be calculated simultaneously in parallel threads
! SO EACH EQUILIBRIUM MUST BE INDEPENDENT
! NOTE: the error code must be local to each equilibria!!!!
! During step and map each equilibrium record with results is saved
! values of T and P, conditions etc.
! Values here are normally set by external conditions or calculated from model
! local list of components, phase_varres with amounts and constitution
! lists of element, species, phases and thermodynamic parameters are global
! status: not used yet?
! multiuse: used for various things like direction in start equilibria
! eqno: sequential number assigned when created
! next: index of next free equilibrium record
!       also index of next equilibrium in a list during step/map calculation.
! eqname: name of equilibrium
! comment: a free text, for example reference for experimental data.
! tpval(1) is T, tpval(2) is P, rgas is R, rtn is R*T
! rtn: value of R*T
! weight: weight value for this experiment, default unity
!     integer status,multiuse,eqno,next
     integer status,multiuse,eqno,nexteq
     character eqname*24,comment*72
     double precision tpval(2),rtn
     double precision :: weight=one
! svfunres: the values of state variable functions valid for this equilibrium
     double precision, dimension(:), allocatable :: svfunres
! the experiments are used in assessments and stored like conditions
! lastcondition: link to condition list
! lastexperiment: link to experiment list
     TYPE(gtp_condition), pointer :: lastcondition,lastexperiment
! components and conversion matrix from components to elements
! complist: array with components (species index or location)??
! compstoi: stoichiometric matrix of compoents relative to elements
! invcompstoi: inverted stoichiometric matrix
     TYPE(gtp_components), dimension(:), allocatable :: complist
     double precision, dimension(:,:), allocatable :: compstoi
     double precision, dimension(:,:), allocatable :: invcompstoi
! one record for each phase+composition set that can be calculated
! phase_varres: here all calculated data for the phases are stored
     TYPE(gtp_phase_varres), dimension(:), allocatable :: phase_varres
! index to the tpfun_parres array is the same as in the global array tpres
! eq_tpres: here local calculated values of TP functions are stored
! should be allocatable, not a pointer
     TYPE(tpfun_parres), dimension(:), allocatable :: eq_tpres
! current values of chemical potentials stored in component record but
! duplicated here for easy acces by application software
     double precision, dimension(:), allocatable :: cmuval
! xconv: convergence criteria for constituent fractions and other things
! dgconv(1) is controlling decrease of DGM for unstable phases
! dgconv(2) not used (yet)
     double precision xconv,gdconv(2)
! delta-G value for merging gridpoints in grid minimizer
! smaller value creates problem for test step3.OCM, MC and austenite merged
!      double precision :: gmindif=-5.0D-2
! testing merging again 190604/BoS
!CCI
     double precision :: gmindif
!CCI
! maxiter: maximum number of iterations allowed
     integer :: maxiter
! CCI
! New parameters based on the work of Joao Pedro Teuber Carvalho (12/2020)
! To scale all changes in phase amount with total number of atoms.
     integer ::  type_change_phase_amount
     double precision :: scale_change_phase_amount

! splitsolver : flag to allow the splitting resolution when conditions lead to a square mass ma
! precondsolver : flag to allow the preconditionning of the matrix before solving linear system
     integer :: precondsolver
     integer :: splitsolver
!CCI
! CCI number of iterations needed for the equilibrium calculation
     integer :: conv_iter
! This is to store additional things not really invented yet ...
! It may be used in ENTER MANY_EQUIL for things to calculate and list
     character (len=80), dimension(:), allocatable :: eqextra
! this is to save a copy of the last calculated system matrix, needed ??
! to calculate dot derivatives, initiate to zero
     integer :: sysmatdim=0,nfixmu=0,nfixph=0
     integer, allocatable :: fixmu(:)
     integer, allocatable :: fixph(:,:)
     double precision, allocatable :: savesysmat(:,:)
! This is temporary data for EEC but must be separate for parallelization
! index of phase_varres for liquid
     integer eecliq
     double precision eecliqs
! temporary array to handle converge problems with change of stable phase set
     integer, dimension(:,:), allocatable :: phaseremoved
```

```
  END TYPE gtp_equilibrium_data
! The primary copy of this structures is declared globally as FIRSTEQ here
! Others may be created when needed for storing experimental data or
! for parallel processing. A global array of these are
  TYPE(gtp_equilibrium_data), dimension(:), allocatable, target :: eqlista
  TYPE(gtp_equilibrium_data), pointer :: firsteq
! This array of equilibrium records are used for storing results during
! STEP and MAP calculations.
  TYPE(gtp_equilibrium_data), dimension(:), allocatable :: eqlines
```

5.3 Records with data shared by several subroutines

The records below have no global variables but are used in some of the calculating subroutines to store complex temporary data, almost like an old-fashioned COMMON area but it is declared inside a subroutine and passed as an argument to the different subroutines.

5.3.1 Parsing data

The data in this record is used parsing the endmember lista and the binary interaction tree.

```
! for each permutation in the binary interaction tree of an endmember one must
! keep track of the permutation and the permutation limit.
! It is not possible to push the value on pystack as one must remember
! them when changing the endmember permutation
! integer, parameter :: permstacklimit=150
! this constant must be incremented when a change is made in gtp_parcalc
  INTEGER, parameter :: gtp_parcalc_version=1
  TYPE gtp_parcalc
! This record contains temporary data that must be separate in different
! parallel processes when calculating G and derivatives for any phase.
! There is nothing here that need to be saved after the calculation is finished
! global variables used when calculating G and derivaties
! sublattice with interaction, interacting constituent, endmember constituents
! PRIVATE inside this structure not liked by some compilers....
! endcon must have maxsubl dimension as it is used for all phases
     integer :: intlat(maxinter),intcon(maxinter),endcon(maxsubl)
! interaction level and number of fraction variables
     integer :: intlevel,nofc
! interacting constituents (max 4) for composition dependent interaction
! iq(j) indicate interacting constituents
! for binary RK+Muggianu iq(3)=iq(4)=iq(5)=0
! for ternary Muggianu in same sublattice iq(4)=iq(5)=0
! for reciprocal composition dependent iq(5)=0
! 2020/BoS not used: Toop, Kohler and simular iq(5) non-zero (not implemented)
```

```
     integer :: iq(5)
! fraction variables in endmember (why +2?) and interaction
     double precision :: yfrem(maxsubl+2),yfrint(maxinter)
! local copy of T, P and RT for this equilibrium
     double precision :: tpv(2),rgast
!    double precision :: ymin=1.0D-30
  end TYPE gtp_parcalc
! this record is declared locally in subroutine calcg_nocheck
```

5.3.2 Fraction product stack

The product of the constituent fractions and their derivatives must be saved now and again during the parsing. The record below is used for that.

```
! this constant must be incremented when a change is made in gtp_pystack
  INTEGER, parameter :: gtp_pystack_version=1
  TYPE gtp_pystack
! records created inside the subroutine push/pop_pystack
! data stored during calculations when entering an interaction record
! previous: link to previous record in stack
! ipermutsave: permutation must be saved
! intrecsave: link to interaction record
! pysave: saved value of product of all constituent fractions
! dpysave: saved value of product of all derivatives of constituent fractions
! d2pysave: saved value of product of all 2nd derivatives of constit fractions
     TYPE(gtp_pystack), pointer :: previous
     integer :: pmqsave
     TYPE(gtp_interaction), pointer :: intrecsave
     double precision :: pysave
     double precision, dimension(:), allocatable :: dpysave
     double precision, dimension(:), allocatable :: d2pysave
  end TYPE gtp_pystack
! declared inside the calcg_internal subroutine
```

5.4 STEP and MAP results data structures

Some application software which is closely related to the equilibrium calculation, like STEP and MAP have some of their data structures declared here as they would otherwise not be able to access the data.

5.4.1 The node point record

In the gtp_eqnode record an equilibrium representing a node point with several lines meeting is stored. It has links to other node points and to two or more lines of calculated equilibria.

```
  INTEGER, parameter :: gtp_eqnode_version=1
  TYPE gtp_eqnode
! This record is to arrange calculated equilibria, for example results
! from a STEP or MAP calculation, in an ordered way.  The equilibrium records
! linked from an eqnode record should normally represent one or more lines
! in a diagram but may be used for other purposes.
! ident is to be able to find a specific node
! nodedtype is to specify invariant, middle, end etc.
! status can be used to supress a line
! color can be used to sepecify color or linetypes (dotted, thick ... etc)
! exits are the number of lines that should exit from the node
! done are the number of calculated lines currently exiting from the node
     integer ident,nodetype,status,color,exits,done
! this node can be in a multilayerd list of eqnodes
     type(gtp_eqnode), pointer :: top,up,down,next,prev
! nodeq is a pointer to the equilibrium record at the node
     type(gtp_equilibrium_data), pointer :: nodeq
! eqlista are pointers to line of equilibria starting or ending at the node
! The equilibrium records are linked with a pointer inside themselves
     type(gtp_equilibrium_data), dimension(:), pointer :: eqlista
! axis is the independent axis variable for the line, negative means decrement
! noeqs gives the number of equilibria in each eqlista, a negative value
! indicates that the node is an endpoint (each line normally has a
! start point and an end point)
     integer, dimension(:), allocatable :: axis,noeqs
! This is a possibility to specify a status for each equilibria in each line
!    integer, dimension(:,:), allocatable :: eqstatus
  end TYPE gtp_eqnode
! can be allocated in a gtp_applicationhead record
```

5.5 Assessment records

Multicomponent thermodynamic databases are created by assessments of many binary and ternary systems based on descriptions of the pure elements. In an assessment experimental and theoretical data are fitted by model parameters for the individual phases. In order to create large databases and extrapolate to multicomponent systems the data and models used for the lower order systems, in particular the pure elements, must be identical.

The assessment of a phase requires that one can include experimental data in the data structure and by using an optimizing software that can vary some of the model parameters. The necessary data structure for this is provided in the GTP package.

The setup of an assessment will include the commands (note the commands listed below may change in later versions of the user interface):

1. ENTER elements, species, phases etc. usually from a macro file.

2. ENTER OPT to enter the number of coefficients to be optimized and workspace needed.
3. ENTER PARAMETERS with coefficients to be optimized.
4. ENTER EQUILIBRIA with experimental data usually from a macro file). The experimental data for each equilibrium added with ENTER EXPERIMENT command, see section 8.9.5
5. SET RANGE_EXP_EQUIL to specify the equilibria with experiments.
6. SET OPTCOEFF_VARIABLE to specify coefficients to be optimized.
7. OPTIMIZE to make a least square fit.
8. LIST OPTIMIZATION to list the result.
9. SAVE UNFORMATTED to create a save file with the current set of data, parameters and results. Save several versions so you can go back to the best one when later versions are messed up.
10. SAVE TDB to create a TDB file with the results (need editing).
11. and other commands as necessry

5.5.1 The assessment head record

This is the record organizing an assessment. It has the all the values related to the optimizing coefficients and has a list of pointers to the equilibria with experimental data. An instance of this is created when starting the software and thus accessible in all packages that use GTP, the arrays declared inside the record can be updated by such packages. There is a possibility to create a linked list of these records to save temporary versions of an assessment. This is not yet implemented and some of the other variables in this recored are not yet used.

The status word has just a single bit defined, AHCOEF, set if the optimizing coefficients have been entered by the command ENTER OPTIMIZE_COEFF. The assessment coefficients are TP function constants named A00 to A99. These TP constants can be used entering other TP functions and phase parameters that should be optimized. There are also subroutines to change the values of these variables from an optimizing software. In order to control the optimization the user can also provide a scaling factor, a minimum and a maximum of a coefficient.

The eqlista array contain pointers to the equilibria with experimental data to be used in the assessment. This is set by a command in the user interface, SET RANGE_EXP_EQUIL. The equilibria with experiments are simply entered with an ENTER EQUILIBRIUM command and for each equilibrium the associated experimental data is entered with the command ENTER EXPERIMENT.

```
! a smart way to have an array of pointers used in gtp_assessmenthead
  TYPE equilibrium_array
     type(gtp_equilibrium_data), pointer :: p1
  end TYPE equilibrium_array
  INTEGER, parameter :: gtp_assessment_version=1
  TYPE gtp_assessmenthead
! This record should summarize the essential information about assessment data
! using GTP.  How it should link to other information is not clear.
! status is status word, AHCOEF is used
! varcoef is the number of variable coefficients
! firstexpeq is the first equilibrium with experimental data
! lwam is allocated workspace at last call to lmdif1
     integer status,varcoef,firstexpeq,lwam
     character*64 general,special
     type(gtp_assessmenthead), pointer :: nextash,prevash
! This is list of pointers to equilibria to be used in the assessnent
! size(eqlista) is the number of equilibria with experimental data
     type(equilibrium_array), dimension(:), allocatable :: eqlista
! These are the coefficients values that are optimized,
! current values, scaling, start values, RSD and optionally min and max
     double precision, dimension(:), allocatable :: coeffvalues
     double precision, dimension(:), allocatable :: coeffscale
     double precision, dimension(:), allocatable :: coeffstart
     double precision, dimension(:), allocatable :: coeffrsd
     double precision, dimension(:), allocatable :: coeffmin
     double precision, dimension(:), allocatable :: coeffmax
! These are the corresponding TP-function constants indices
     integer, dimension(:), allocatable :: coeffindex
! This array indicate currently optimized variables:
!  -1=unused, 0=fix, 1=fix with min, 2=fix with max, 3=fix with min and max
!  10=optimized, 11=opt with min, 12=opt with max, 13=opt with min and max
     integer, dimension(:), allocatable :: coeffstate
! Work arrays ...
     double precision, dimension(:), allocatable :: wopt
  end TYPE gtp_assessmenthead
! this record should be allocated for assessments when necessary
  type(gtp_assessmenthead), allocatable :: ashrecord
!  type(gtp_assessmenthead), pointer :: firstash,lastash
! but this is later allocated, to avoid memory loss ashrecord should be used
! and then this pointer should be set to that record
  type(gtp_assessmenthead), pointer :: firstash
```

5.6 Other application head record

This is a template for other applications that need access to the internal data of OC.

```
  INTEGER, parameter :: gtp_applicationhead_version=1
  TYPE gtp_applicationhead
! This record should summarize the essential information about an application
! using GTP.  How it should link to other information is not clear.
! The character variables should be used to indicate that.
     integer apptyp,status
     character*64 general,special
! These can be used to define axis and other things
     integer, dimension(:), allocatable :: ivals
     double precision, dimension(:), allocatable :: rvals
     character*64, dimension(:), allocatable :: cvals
     type(gtp_applicationhead), pointer :: nextapp,prevapp
! The headnode can be the start of a structure of eqnodes with lines
     type(gtp_eqnode) :: headnode
! this is the start of a list of nodes with calculated lines or
! single equilibria that belong to the application.
     type(gtp_eqnode), dimension(:), allocatable :: nodlista
  end TYPE gtp_applicationhead
! this record is allocated when necessary
  type(gtp_applicationhead), pointer :: firstapp,lastapp
```

6 Some more private variables in GTP

The variables below contain some information used in several subroutines. Additionally many of the record types described earlier are declared as private arrays to protect them somewhat.

```
! counters for elements, species and phases initiated to zero
  integer, private :: noofel=0,noofsp=0,noofph=0
! counter for phase tuples (combination of phase+compset)
  integer, private :: nooftuples=0
! counters for property and interaction records, just for fun
  integer, private :: noofprop,noofint,noofem
! free lists in phase_varres records and addition records
  integer, private :: csfree,addrecs
! free list of references and equilibria
  integer, private :: reffree,eqfree
! maximum number of properties calculated for a phase
  integer, private :: maxcalcprop=20
! highcs is highest used phase_varres record (for copy equil etc)
  integer, private :: highcs
! Trace for debugging (not used)
  logical, private :: ttrace
! Output for debugging gridmin
  integer, private :: lutbug=0
! used for notallowlisting
  double precision :: proda=zero,privilege=zero
! minimum constituent fraction
```

```
  double precision :: bmpymin
! number of defined property types like TC, BMAG etc
  integer, private :: ndefprop
! this is the index of mobility data, set in init_gtp in subroutine gtp3A
  integer, private :: mqindex
! quasichemical model type, 1=classic, 2=corrceted type 1, 3=corrected type 2
  integer :: qcmodel=1
! this is to remember how manytimes find_gridmeen needs to search all gridp
  integer :: ngridseek
! this is to handle EEC in the grid minimizer NOT GOOD FOR PARALLELIZATION
!  integer :: neecgrid
  double precision :: sliqmax,sliqmin,gliqeec,sliqeec
! this is for warnings about using unkown model parameter identifiers
  integer, parameter :: mundefmpi=10
  integer nundefmpi
  character undefmpi(mundefmpi)*4
! this is to give some debug information when reading a database
  logical :: dbcheck=.FALSE.
! this is set zero by new_gtp and incremented each time a Toop record
! is created in any phase
   integer uniqid
! this is to allow select_phases from database files
   integer nselph
   character (len=24), allocatable, dimension(:) :: seltdbph
! This is to indicate mobility parameters, no wildcared fractions allowed
   integer nowildcard(3)
```

7 Data structures for the TP-fun routines

The TP functions are used to enter model parameters and other functions that depend on T and P. They have a limited sytax as the software must be able to calculate their first and second derivatives with respect to T and P.

The expressions are stored only once but the calculated values of the functions are stored separately for each equilibria as each equilibria can have different values of T and P.

7.1 Structure to store expressions of TP functions

In this record the coefficients and powers of T and P and possible unary functions used is stored. There is a simple parser that can read a simple TP function similar to what TC has. My idea has been to extend this a bit by allowing parenthesis in a more flexible way but that is not yet implemented. A TP function can refer to another TP function using the link array. This means a recursive evaluation of TP functions must be implemented.

This information must not be changed during parallel processing. During assessments one may calculate each experimental equilibria in parallel and as the coefficients of TP functions are varied by the assessment procedure one may think that this array must also be separate for each parallel process. But changes of coefficients are not done during the parallel execution so there is no problem.

```
  integer, parameter :: tpfun_expression_version=1
  TYPE tpfun_expression
! Coefficients, T and P powers, unary functions and links to other functions
     integer noofcoeffs,nextfrex
     double precision, dimension(:), pointer :: coeffs
! each coefficient kan have powers of T and P/V and links to other TPFUNS
! and be multiplied with a following LOG or EXP term.
! wpow USED FOR MULTIPLYING WITH ANOTHER FUNCTION!!
     integer, dimension(:), pointer :: tpow
     integer, dimension(:), pointer :: ppow
     integer, dimension(:), pointer :: wpow
     integer, dimension(:), pointer :: plevel
     integer, dimension(:), pointer :: link
  END TYPE tpfun_expression
! These records are allocated when needed, not stored in arrays
```

7.2 Bits used in TP function status

These bits are used for different purposes

```
! BITS in TPFUN
! TPCONST     set if a constant value
! TPOPTCON    set if optimizing value
! TPNOTENT    set if referenced but not entered (when reading TDB files)
! TPVALUE     set if evaluated only explicitly (keeping its value)
! TPEXPORT    set if value should be exported to symbol
! TPIMPORT    set if value should be imported from symbol (only for constants)
! TPINTEIN    set if value should always be calculated
  integer, parameter :: &
       TPCONST=0,    TPOPTCON=1,   TPNOTENT=2,    TPVALUE=3, &
       TPEXPORT=4,   TPIMPORT=5
```

7.3 Function root record type

In this record a name of the function is stored and the number of temperature ranges. For each range a low temperature limit and a link to a tpfun_expression record is stored. There is also a high temperature limit. As the number of ranges can vary the arrays are allocatable (declared as pointers). At the same time as a root record is reserved to store a TP function one also reserves a tpfun_parres record and in this the last calculated result of the function is stored. Saving the last calculated values speeds up calculation because the same function is often used in many parameters and needed many times for the same values of T and P. But in parallel processing the values of T and P can be different in each processor and thus the results must be separate in each.

```
  integer, parameter :: tpfun_root_version=1
  TYPE tpfun_root
! Root of a TP function including name with links to coefficients and codes
! and results.  Note that during calculations which can be parallelized
! the results can be different for each parallel process
     character*(lenfnsym) symbol
! Why are limits declared as pointers?? They cannot be properly deallocated
! limits are the low temperature limit for each range
! funlinks links to expression records for each range
! each range can have its own function, status indicate if T and P or T and V
! nextorsymbol is initiated to next index, then possible symbol link!
! forcenewcalc force new calculation when optimizing variable changed
! rewind is used to check for duplicates reading from TDB file
! not saved on unformatted files
! If bit TPIMPORT set the function must be a constant
!    and nextorsymbol is index of symbol
! If bit TPEXPORT set then the value of the function (not the derivatives)
!    and nextorsymbol is index of symbol
!     integer noofranges,nextfree,status,forcenewcalc
     integer noofranges,nextorsymbol,status,forcenewcalc,rewind
     double precision, dimension(:), pointer :: limits
     TYPE(tpfun_expression), dimension(:), pointer :: funlinks
     double precision hightlimit
  END TYPE tpfun_root
! These records are stored in arrays as the actual function is global but each
! equilibrium has its own result array (tpfun_parres) depending on the local
! values of T and P/V.  The same indiex is used in the global and local arrays.
! allocated in init_gtp
  TYPE(tpfun_root), private, dimension(:), pointer :: tpfuns
```

7.4 Structure for calculated results of TP functions

This record contains the last values of T and P used for calculation the TP function and the results including first and second derivatives of T and P. These are ordered as F, F.T, F.P, F.T.T, F.T.P and F.P.P. The reason to have this as a separate array is that in parallel processing the values of T and P may be different in each process and thus the results also, whereas the values of the coefficients are the same. The size of all the function records are allocated dynamically by the tpfun_init subroutine.

```
  integer, parameter :: tpfun_parres_version=1
  TYPE tpfun_parres
```

```
! Contains TP results, 6 double for results and 2 doubles for T and P
! values used to calculate the results
! Note that during calculations which can be parallelized the
! results can be different for each tread
     integer forcenewcalc
     double precision, dimension(2) :: tpused
     double precision, dimension(6) :: results
  END TYPE tpfun_parres
! This array is local to the gtp_equilibrium_data record
! index is the same as the function
```

8 Subroutines and functions

It is not self evident how to organize the description of the subroutines in GTP. One can base it on the type of service the subroutine provides like entering data, listing data, calculating or on the type of object the action is performed on like on elements, phases, additions etc. A mixed approach is made here where '"find", 'enter" and "list" subroutines operating on general objects are grouped together whereas subroutines specific for "state variables", "additions" and similar things are grouped together for all services like entering, listing and calculations. One reason is that when a new addition is implemented all of these services must be provided together so it is natural to keep them together also in the documentation as a programmer has to provide new subroutines for all of them.

The naming of subroutines is also more or less arbitrary, there are more thn 400 of them in Appendix F. A "find" subroutine usually has a charater as argument to find the index of an element or phase, a "get" subroutine extracts som data for an element of phase etc. "set" usually means to assign a value to a variable but sometimes "enter" or "amend" does the same thing.

The important part of a subroutine declaration is the arguments and the verbatim section normally include the argument list and a short explanation of those. The local variables and the actual code is not interesting in this documentation. The text surrounding the verbatim should explain the action of the subroutine and its connection to the rest of the code. Some functions and subroutines are grouped together in one LaTeX section as they are very similar. But it can be quite a challenge to find a subroutine that perferm a desired action. In some cases one may even find two subroutines doing (almost) the same task because I had forgotten or could not find the first version.

8.1 Variable names

The arguments for the subroutines and functions are normally explained but some standards have been used

Symbol	Type	Meaning
iph	integer	Phase index in PHASES
ics	integer	Composition set number
lokph	integer	Phase location (index to PHLISTA)
lokcs	integer	Index of PHASE_VARRES array for a composition set
ceq	pointer	Pointer to current gtp_equilibrium_data record

8.2 Initiallization, gtp3A

This subroutine must be called before any other in the GTP package. It dimensions arrays and creates some initial data structures. The arguments are presently not used but should be used to dimension arrays and provide default values.

```
 subroutine init_gtp(intvar,dblvar)
! initiate the data structure
! create element and species record for electrons and vacancies
! the allocation of many arrays should be provided calling this routne
! intvar and dblvar will eventually be used for allocations and defaults
   implicit none
   integer intvar(*)
   double precision dblvar(*)
```

8.2.1 Further initializations

Creating an assessment head record and all default values of global parameters.

```
 subroutine assessmenthead(ash)
! create an assessment head record and do more (later)
   type(gtp_assessmenthead), pointer :: ash
--------------------
 subroutine initialize_default_global_parameters(firsteq)
   type(gtp_equilibrium_data), pointer :: firsteq
```

8.2.2 Reinitiate OC for a new system

This requites explicit deliting of phases and some other data.

```
 subroutine new_gtp
!
! DELETES ALL DATA so a new TDB file can be read
!
! this is needed before reading a new unformatted file (or same file again)
! we must go through all records and delete and deallocate each
! separately.  Very similar to gtpread
   implicit none
---------------------------
 subroutine deallocate_gtp(intvar,dblvar)
! deallocate the data structure
   implicit none
   integer allocatestatus
   integer intvar(*)
   double precision dblvar(*)
------------------------------
 subroutine delphase(lokph)
! save data for phase at location lokph (except data in the equilibrium record)
! For phases with disordered set of parameters we must access the number of
! sublattices via firsteq
   implicit none
   integer lokph
```

8.3 Functions to know how many and to find them

The counters for elements etc are private so external software must call functions to find out how many elements etc that the system has. They are all given here. The nooftup replaces the old noofphasetuples_old.

```
 integer function noel()
! number of elements because noofel is private
! should take care if elements are suspended
 =======================
 integer function nosp()
! number of species because noofsp is private
! should take care if species are suspended
 =======================
 integer function noph()
! number of phases because noofph is private
! should take care if phases are hidden
 =======================
!\begin{verbatim} %-
 integer function noofcs(iph)
! returns the number of compositions sets for phase iph
   implicit none
   integer iph
 =======================
 integer function noconst(iph,ics,ceq)
! number of constituents for iph (include single constituents on a sublattice)
! It tests if a constituent is suspended which can be different in each ics.
   implicit none
   integer iph,ics
   TYPE(gtp_equilibrium_data), pointer :: ceq
 =======================
 integer function nooftup()
! number of phase tuples
 =======================
! integer function noofphasetuples_old()
! number of phase tuples REDUNDANT !!
```

8.3.1 How many state variable functions

The number of state variable functions entered is given by this.

```
 integer function nosvf()
! number of state variable functions
```

8.3.2 How many equilibria

There is a global array with equilibria records but only a few of them may be allocated with data. This routine returns the number of allocated equilibrium records.

```
 integer function noeq()
! returns the number of equilibria entered
```

8.3.3 Total number of phases and composition sets

This is needed when dimensioning arrays for phases and composition sets for calculations.

```
 integer function nonsusphcs(ceq)
! returns the total number of unhidden phases+composition sets
! in the system.  Used for dimensioning work arrays and in loops
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.3.4 Find element, species and components

These subroutines translate from name to index or location of data or vice versa. There are other some "find" subroutines for special things like find_gridmin described in 8.17.5.

Some duplication present because I did not whant to change some older subroutines.

```
 subroutine find_element_by_name(name,iel)
```

```
! find an element index by its name, exact fit required
   implicit none
   character name*(*)
   integer iel
 =======================
 subroutine find_component_by_name(name,icomp,ceq)
! BEWARE: one may in the future have different components in different
! equilibria. components are a subset of the species
   implicit none
   character*(*) name
   integer icomp
   TYPE(gtp_equilibrium_data), pointer :: ceq
 =======================
 subroutine find_species_by_name(name,isp)
! locates a species index from its name, unique abbreviation
! or exact match needed
   implicit none
   character name*(*)
   integer isp
 =======================
 subroutine find_species_by_name_exact(name,isp)
! locates a species index from its name, exact match needed
   implicit none
   character name*(*)
   integer isp
 =======================
 subroutine find_species_record(name,loksp)
! locates a species record allowing abbreviations
   implicit none
   character name*(*)
   integer loksp
 =======================
 subroutine find_species_record_noabbr(name,loksp)
! locates a species record no abbreviations allowed
   implicit none
   character name*(*)
   integer loksp
 =======================
 subroutine find_species_record_exact(name,loksp)
! locates a species record, exact match needed
! for parameters, V must not be accepted as abbreviation of VA or C for CR
   implicit none
   integer loksp
   character name*(*)
```

8.3.5 Find a phase, composition set and constituents

These are routines to find a phase by index or name, the phase tuple should be used in application software. Note the phase tuple contain both the phase number and composition set number and also an index to the data structure for the phase.

For the name of the second and higher composition set of a phase use the phase name followed by a hash sign, #, followd by a digit to specify the composition set. If you create composition sets yourself you can add a prefix and suffix, each max 4 letters, to the phase name, separated by an underscore "_".

A composition set created by the grid minimizer has "_AUTO" as suffix.

A phase name must not be an abbreviation of the name of another phase. That seems straightforward but sometimes one comes across phase names like ALPHA and ALPHA_PRIME which is not allowed. See also section 8.4.7.

Constituents in a phase are ordered sequentially across all sublattices. This routine finds this sequential index provided with the sublattice number and sequential order in the sublattice.

```
 subroutine find_phasetuple_by_name(name,phcsx)
! finds a phase with name "name", returns phase tuple index
! handles composition sets either with prefix/suffix or #digit
! When no pre/suffix nor # always return first composition set
   implicit none
   character name*(*)
   integer phcsx
 =======================
 subroutine find_phase_by_name(name,iph,ics)
! finds a phase with name "name", returns address of phase, first fit accepted
! handles composition sets either with prefix/suffix or #digit
! When no pre/suffix nor # always return first composition set
   implicit none
   character name*(*)
   integer iph,ics
 =======================
 integer function find_phasetuple_by_indices(iph,ics)
! subroutine find_phasetuple_by_indices(iph,ics)
! find phase tuple index given phase index and composition set number
   integer iph,ics
 =======================
 subroutine find_phasex_by_name(name,phcsx,iph,zcs)
! finds a phase with name "name", returns index and tuplet of phase.
! All phases checked and error return if name is ambiguous
! handles composition sets either with prefix/suffix or #digit or both
! if no # check all composition sets for prefix/suffix
! special if phcsx = -1 and there are several composition sets then
! zcs is set to -(number of composition sets).  Used when changing status
! phcsx, iph and zcs are values to return!
   implicit none
   character name*(*)
   integer phcsx,iph,zcs
 ======================
 subroutine find_phase_by_name_exact(name,iph,ics)
! finds a phase with name "name", returns address of phase. exact match req.
! handles composition sets either with prefix/suffix or #digit
! no pre/suffix nor # gives first composition set
   implicit none
   character name*(*)
   integer iph,ics
---------------
 subroutine find_constituent(iph,spname,mass,icon)
! find the constituent "spname" of a phase. spname can have a sublattice #digit
! Return the index of the constituent in icon.  Additionally the mass
! of the species is returned.
   implicit none
   character*(*) spname
   double precision mass
   integer iph,icon
----------------
   subroutine findconst(lokph,ll,spix,constix)
! locates the constituent index of species with index spix in sublattice ll
! and returns it in constix.  For wildcards spix is -99; return -99
! THERE MAY ALREADY BE A SIMULAR SUBROUTINE ... CHECK
   implicit none
   integer lokph,ll,spix,constix
```

8.3.6 Find and select equilibrium

As already stated each equilibrium has a separate set of conditions and results. Equilibria are created by calling enter_equilibrium, see section 8.6.12. A “default” equilibrium pointed to by the variable “firsteq” is created when the program is started. The pointer variable “ceq” is used in many subroutine to indicate the “current” equilibrium for which calculations or data are used.

```
 subroutine findeq(name,ieq)
! finds the equilibrium with name "name" and returns its index
! ieq should be the current equilibrium
   implicit none
   character name*(*)
   integer ieq
---------------
 subroutine selecteq(ieq,ceq)
! checks if equilibrium ieq exists and if so set it as current
```

```
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer ieq
```

8.4 Get things

The difference between find and get is not very distinct. Normally the "find" routines requires a name or symbol to return an index or location whereas the "get" routines require an index or location to get more data. Some of the "get" routines are very specific and the documentation is found together with the type of data you want to get.

```
 subroutine get_phase_record(iph,lokph)
! given phase index iph this returns the phase location lokph
   implicit none
   integer iph,lokph
 =======================
 subroutine get_phase_variance(iph,nv)
! returns the number of independent variable fractions in phase iph
   implicit none
   integer iph,nv
 =======================
 subroutine get_constituent_location(lokph,cno,loksp)
! returns the location of the species record of a constituent
! requred for ionic liquids as phlista is private
   implicit none
   integer lokph,loksp,cno
 =======================
 subroutine get_phase_compset(iph,ics,lokph,lokcs)
! Given iph and ics the phase and composition set locations are returned
! Checks that ics and ics are not outside bounds.
   implicit none
   integer iph,ics,lokph,lokcs
```

8.4.1 Get phase constituent name

By supplying a phase index and and the sequential index (counted over all sublattices, this routine returns the name and mass of the constituent.

```
 subroutine get_constituent_name(iph,iseq,spname,mass)
! find the constituent with sequential index iseq in phase iph
! return name in "spname" and mass in mass
   implicit none
   character*(*) spname
   integer iph,iseq
   double precision mass
```

8.4.2 Get element data

The data for an element is returned. The mass of an element can be amended.

```
 subroutine get_element_data(iel,elsym,elname,refstat,mass,h298,s298)
! return element data as that is stored as private in GTP
   implicit none
   character elsym*2, elname*(*),refstat*(*)
   double precision mass,h298,s298
   integer iel
  -----------
 subroutine new_element_data(iel,elsym,elname,refstat,mass,h298,s298)
! set new values in an element record, only mass allowed to change ...
   implicit none
   character elsym*2, elname*(*),refstat*(*)
   double precision mass,h298,s298
   integer iel
```

8.4.3 Get component or species name

Components are by default the elements but the user can (sometimes in the future) change this to any set of species. The selected set of species must be orthogonal, i.e. include all elements. Each equilibrium will be able to have a different set of components.

The set of components are important because one can only use components to set amounts of fractions with conditions like $N(<\text{ components }>)$. There is an alternative method to set amounts using the subroutine set_input_amounts where amounts or mass of different species can be used to give amounts of components.

The get_species_location is used when redefining the set of components.

```
subroutine get_component_name(icomp,name,ceq)
! return the name of component icomp
   implicit none
   character*(*) name
   integer icomp
   TYPE(gtp_equilibrium_data), pointer :: ceq
 ======================
 subroutine get_species_name(isp,spsym)
! return species name, isp is species number
   implicit none
   character spsym*(*)
   integer isp
 ======================
 subroutine get_species_location(isp,loksp,spsym)
! return species location and name, isp is species number
```

```
   implicit none
   character spsym*(*)
   integer isp,loksp
```

8.4.4 Get species data relative elements and components

A species is just a stoichiometric arrangement of elements like a molecule. It has no thermodynamic data as they are stored at with the phase where the species is a constituent. The stoichiometry of a species is fixed. The composition of a phase can vary if there are two ore more species as constituents in one or more sublattices.

The elements are the simplest species. A species can have a charge which may be non-integer. For some models, like UNIQUAC, species can have additional data used for the modeling like area and volume, see section 8.6.15. However, these does not costitute a volume for the phase.

The get_species_component_data returns the stoichiometry of the species using the current set of components. Normally that the components are the same as the elements.

```
 subroutine get_species_data(loksp,nspel,ielno,stoi,smass,qsp,nextra,extra)
! return species data, loksp is from a call to find_species_record
! nspel: integer, number of elements in species
! ielno: integer array, element indices
! stoi: double array, stoichiometric factors
! smass: double, mass of species
! qsp: double, charge of the species
! nextra, integer, number of additional values
! extra: double, some additional values like UNIQUAC volume and area
   implicit none
   integer, dimension(*) :: ielno
   double precision, dimension(*) :: stoi,extra
   integer loksp,nspel,nextra
   double precision smass,qsp
 ======================
 subroutine get_species_component_data(loksp,nspel,compnos,stoi,smass,qsp,ceq)
! return species data, loksp is from a call to find_species_record
! Here we return stoichiometry using components
! nspel: integer, number of components in species
! compno: integer array, component (species) indices
! stoi: double array, stoichiometric factors
! smass: double, mass of species
! qsp: double, charge of the species
   implicit none
   integer, dimension(*) :: compnos
   double precision, dimension(*) :: stoi(*)
   integer loksp,nspel
   double precision smass,qsp
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.4.5 New species stoichiometry

A subroutine provided by a user to change the stoichiometry of a species. The reason for this is unclear.

```
 subroutine set_new_stoichiometry(loksp, new_stoi, ispel)
! provided by Clement Introini
! Change the stoichiometric coefficient of the ispel-th element of loksp-th
! species (the last one when ispel is not given)
! loksp: index of the species (input integer)
! new_stoi: new value of the stoichiometric coefficient (input double precision)
! ispel: index of the element (optional, input integer)
   implicit none
   integer, intent(in):: loksp
   integer, intent(in), optional :: ispel
   double precision, intent(in):: new_stoi
```

8.4.6 Mass of component

For the mass balance calculations during equilibrium calculations the mass of a component is needed frequently. This subroutine just returns the mass of a component.

```
 double precision function mass_of(component,ceq)
! return mass of component
! smass: double, mass of species
   implicit none
   integer :: component
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.4.7 Get phase name

The title says all but there are two variants of this, one using phase tuples, the other integer variables as arguments. There are also some more or less redundant routines to get the record for phase data. See also section 8.3.5. When the development of OC started the phasetup data structure had not been created, thus there are some reduncdancy. The use of phasetup is recommended in any new code but one has to be aware that the number of phasetuples may change whenever new composition sets of a phase are created.

```
 subroutine get_phase_name(iph,ics,name)
! Given the phase index and composition set number this subroutine returns
! the name with pre- and suffix for composition sets added and also
! a \# followed by a digit 1-9 if there are more than one composition sets
   implicit none
   character name*(*)
   integer iph,ics
 ======================
 subroutine get_phasetup_name(phtupx,name)
! phasetuple(phtupx)%phase is index to phlista
! the name has pre- and suffix for composition sets added and also
! a \# followed by a digit 2-9 for composition sets higher than 1.
   implicit none
   character name*(*)
   integer phtupx
 ======================
 subroutine get_phasetuple_name(phtuple,name)
! phtuple is a phase tuple
! the name has pre- and suffix for composition sets added and also
! a \# followed by a digit 2-9 for composition sets higher than 1.
   implicit none
   character name*(*)
   type(gtp_phasetuple) :: phtuple
 ======================
 subroutine get_phasetup_record(phtx,lokcs,ceq)
! return lokcs when phase tuple known
   implicit none
   integer phtx,lokcs
   TYPE(gtp_equilibrium_data), pointer :: ceq
 ======================
 integer function gettupix(iph,ics)
! convert phase and compset index to tuple index
   implicit none
   integer iph,ics
```

8.4.8 Get phase data

This is a very important subroutine used at each iteration during calculations to obtain information about a phase. In the knr array the constituents are given as the integer indices of the species location in the array SPLISTA. The constituents are stored sequentially and the first nkl(1) positions in knr belong to sublattice 1, the next nkl(2) to sublattice 2 etc. The order is alphabetical for each sublattice (not for ionic liquid). In yarr the fractions of the constituents are given sequentially in the same order.

NOTE the ionic liquid has the constituents in the second sublattice ordered by first all anions, then Va (if any), then all neutrals. The anions and the neutrals are ordered alphabetically.

In the array qq the current number of components per formula unit is returned in the first index and the current charge (valence) in the second.

The routine get_phase_structure returns just the number of sublattices and constituents in each. It is needed when calculating the derivatives of the chemical potentials in order to transform mobilities to diffusion coefficients.

```
 subroutine get_constituent_data(iph,ics,icons,yarr,charge,csname,ncel,ceq)
!CCI
   ! return the constitution for phase iph (ics composition set)
   ! yarr: double, fraction of constituent
   ! charge: integer, charge of constituent
   ! ncel: integer, number of element in the constituant
   ! consname: name of the constituent
   ! ceq: pointer, to current gtp_equilibrium_data record
   implicit none
   integer, intent (in) :: iph,ics,icons
   double precision, intent (inout) :: yarr
!CCI (adding ncel)
   integer, intent (inout) :: charge,ncel
!CCI
   character*(*) , intent (inout) :: csname

   TYPE(gtp_equilibrium_data), pointer :: ceq
 ======================
 subroutine get_phase_data(iph,ics,nsl,nkl,knr,yarr,sites,qq,ceq)
! return the structure of phase iph and constituntion of comp.set ics
! nsl: integer, number of sublattices
! nkl: integer array, number of constituents in each sublattice
! knr: integer array, species location (not index) of constituents (all subl)
! yarr: double array, fraction of constituents (in all sublattices)
! sites: double array, number of sites in each sublattice
! qq: double array, (must be dimensioned at least 5) although only 2 used:
! qq(1) is number of real atoms per formula unit for current constitution
! qq(2) is net charge of phase for current constitution
! ceq: pointer, to current gtp_equilibrium_data record
   implicit none
   integer, dimension(*) :: nkl,knr
   double precision, dimension(*) :: yarr,sites,qq
   integer iph,ics,nsl
   TYPE(gtp_equilibrium_data), pointer :: ceq
 ======================
 subroutine get_phase_structure(lokph,nsl,nkl)
! return the number of sblattices and constituents in each.
! nsl: integer, number of sublattices
! nkl: integer array, number of constituents in each sublattice
! USED when calculating derivatives of chemical potentials and diffusion coef
   implicit none
   integer, dimension(*) :: nkl
   integer lokph,nsl
```

8.4.9 Phase tuple array

The subroutine get_phtuplearray is redundant as application software can use the global array PHASETUPLE declared within the OC software.

In the phase tuple array the first indices from 1 to nooph represent the first composition set of the phases. Whenever a phase can be stable with two or more compositions extra composition sets are created. These ar placed after the first nooph phases.

The function nooftup in section 8.3 should be called whenever the grid minimizer has been used as new composition sets may have been created.

```
 integer function get_phtuplearray(phcs)
! copies the internal phase tuple array to external software
! function value set to number of tuples
   type(gtp_phasetuple), dimension(*) :: phcs
---------------
 integer function noofphasetuples()
! number of phase tuples
```

8.5 Set things

Many things can be set but most of the ways to set them are described together with the object to set. How to set conditions is described in 8.9.6 because it involves state variables.

8.5.1 Set constitution

This is a subroutine used when iterating to find the equilibrium. At each iteration the new constitutions of the phases are set using this subroutine. Some internal quantities are also calculated like the number of atoms per mole formula unit of the phase and the charge of the phase if there are ions. These are returned in the call.

The array yfr in the phase_varres record belonging to the composition set (iph,ics) and it is not private and a programmer may thus change the values externally without calling set_constitution. But this is strongly discouraged as the internal variables qq(1) and qq(2) must be updated for each set of fractions to ensure that the massbalance is correct.

The order of the fractions in yfra must be the same as in get_phase_data, see 8.4.8.

```
 subroutine set_constitution(iph,ics,yfra,qq,ceq)
! set the constituent fractions of a phase and composition set and the
! number of real moles and mass per formula unit of phase
! returns number of real atoms in qq(1), charge in qq(2) and mass in qq(3)
! for ionic liquids sets the number of sites in the sublattices
   implicit none
   double precision, dimension(*) :: yfra,qq
   integer iph,ics
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.5.2 Set reference state for a component

The user may change the reference state of an element. For each component the user can select a phase, temperature and pressure as reference state. If a * is given as temperature and pressure the current value of T and P will be used. The reference state is used in calculate_reference_state in section 8.11.10 and will affect the value of a chemical potential of the element.

NOTE: If all elements have the same phase as reference state the integral enthalpy, entropy and Gibbs energy will also refer to this phase as it represents a mixing value of the enthalpy, entropy or Gibbs energy for the phase. But if the elements have different phases as reference state the values of the integral properties will not be affected but refer to the default reference state, normally the stable state at 298.15 K and 1 bar.

```
 subroutine set_reference_state(icomp,iph,tpval,ceq)
! set the reference state of a component to be "iph" at tpval
   implicit none
   integer icomp,iph
   double precision, dimension(2) :: tpval
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.5.3 Set condition

This subroutine changes the degree of freedoms for a calculation. See section 8.9.6.

8.5.4 Amend components

This subroutine changes the set of components of the system. The components are used when setting conditions of the amounts or chemical potentials in the system. By default the elements are the components but in some cases it is convenient to use another set of species for setting conditions. The number of components cannot be changed by this command, the number of new components must be the same as the number of elements. The components must form an othogonal set.

This command is fragile. Beware that when using other components than the elements one may have negative mole fractions of the components and amounts of the phases (the mass and mass fraction is positive).

```
 subroutine amend_components(line,ceq)
! amend the set of components
   implicit none
   character line*(*)
   type(gtp_equilibrium_data), pointer :: ceq
```

8.6 Enter data and other things, gtp3B

The subroutines for entering data and other things can be named as new, enter, add, create etc. according to the mind of the programmer when the subroutine was written. In all cases the data is provided as arguments in the call, there is no interactions with the user.

By default there is one equilibrium record created but the user can create additiona ones. This is useful for assessments or simulations.

8.6.1 Enter element data

All data for an element. Some checks are made. The elements are automatically eneterd also as species so they can be constituents of phases.

```
 subroutine store_element(symb,name,refstate,mass,h298,s298)
! Creates an element record after checks.
! symb: character*2, symbol (it can be a single character like H or V)
! name: character, free text name of the element
! refstate: character, free text name of reference state.
! mass: double, mass of element in g/mol
! h298: double, enthalpy difference between 0 and 298.14 K
! s298: double, entropy at 298.15 K
   implicit none
   CHARACTER*(*) symb,name,refstate
   DOUBLE PRECISION mass,h298,s298
```

8.6.2 Enter species data

All data for an element. Some checks are made. The elements constituting the species must have been entered before. A species can have a positive or negative charge using the element index -1 with a stoichiometric factor.

```
 subroutine enter_species(symb,noelx,ellist,stoik)
! creates a new species
! symb: character*24, name of species, often equal to stoichiometric formula
! noelx: integer, number of elements in stoichiometric formula (incl charge)
! ellist: character array, element names (electron is /-)
! stoik: double array, must be positive except for electron.
   implicit none
   character symb*(*),ellist(*)*(*)
   integer noelx
   double precision stoik(*)
```

8.6.3 Enter phase and model interactivly

This routine asks for name, model, constituents and other information needed to enter a phase. Then it calls the routine in next section.

```
  subroutine enterphase(cline,last)
! interactive entering of phase
    character cline*(*)
    integer last
!    type(gtp_equilibrium_data), pointer :: ceq
```

8.6.4 Enter phase and model

This subroutine is called with the model data needed to create the data structure for a phase (no parameter data). The model variable is just a text, phtype is used to arrange gas (G) and liquids (L) before the alphabetical list of the other phases.

```
 subroutine enter_phase(name,nsl,knr,const,sites,model,phtype,warning,emodel)
! creates the data structure for a new phase
! name: character*24, name of phase
! nsl: integer, number of sublattices (range 1-9)
! knr: integer array, number of constituents in each sublattice
! const: character array, constituent (species) names in sequential order
! sites: double array, number of sites on the sublattices
! model: character, some fixed parts, some free text
! phtype: character*1, specifies G for gas, L for liquid
! emodel: for entropy model and maybe more
   implicit none
   character name*(*),model*(*),phtype*(*)
   integer nsl,emodel
   integer, dimension(*) :: knr
   double precision, dimension(*) :: sites
   character, dimension(*) :: const*(*)
   logical warning
```

8.6.5 Sorting constituents in ionic liquid model

The ionic liquid model 2 sublatte model requires all cations (with positive charge) on the first sublattice in alphabetical order. On the second sublattice the anions (with negative charge) should be first (in alphabetical order), the the hypothetical vacancy (if any), then any neutrals in alphabetical order. This subroutine takes care of that

```
 subroutine sort_ionliqconst(lokph,mode,knr,kconlok,klok)
! sorts constituents in ionic liquid, both when entering phase
! and decoding parameter constituents
! order: 1st sublattice only cations
! 2nd: anions, VA, neutrals
! mode=0 at enter phase, wildcard ok in 1st sublattice if neiher anions nor Va
! mode=1 at enter parameter (wildcard allowed, i.e. some kconlok(i)=-1)
! some  parameters not allowed, L(ion,A+:B,C), must be L(ion,*:B,C), check!
   implicit none
   integer lokph,knr(*),kconlok(*),klok(*),mode
```

8.6.6 Enter composition set

As explained in section 2.6 a phase may exist simultaneously with several different composition sets. This can be due to miscibility gaps or ordering. Some carbides like cubic TiC is modeled as the same phase as the metallic FCC and it may be stable at the same time as the austenite phase in steels. This subroutine creates a new composition set for a phase.

```
 subroutine enter_composition_set(iph,prefix,suffix,icsno)
! adds a composition set to a phase.
! iph: integer, phase index
! prefix: character*4, optional prefix to original phase name
! suffix: character*4, optional suffix to original phase name
! icsno: integer, returned composition set index (value 2-9)
! ceq: pointer, to current gtp_equilibrium_data
!
! BEWARE this must be done in all equilibria (also during parallel processes)
! There may still be problems with equilibria saved during STEP and MAP
!
   implicit none
   integer iph,icsno
   character*(*) prefix,suffix
!   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.6.7 Remove or suspend composition set

Sometimes the grid minimizer creates too many composition sets and the further calculations may be easier if these are removed. But sometimes it is not possible to delete or remove them (when running in parallel) and then they may be suspended.

```
 subroutine suspend_composition_set(iph,parallel,ceq)
! the last composition set is suspended in all equilibria
!
! If parallel is TRUE then execution is not in parallel (threaded)
!
   implicit none
   integer iph
   logical parallel
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine suspend_unstable_sets(mode,ceq)
! suspend extra composition sets that are not stable
   implicit none
   integer mode
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine remove_composition_set(iph,force)
! subroutine delete_composition_set(iph,force)
! the last composition set of phase iph is deleted, update csfree and highcs
! SPURIOUS ERRORS OCCUR IN THIS SUBROUTINE
!
! >>>>>>>>>>>>>>>>>>>>>>>>>>>> NOTE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< !
! Not safe to remove composition sets when more than one equilibrium       !
! >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> !
!
! If force is TRUE delete anyway ... very dangerous ...
!
   implicit none
!
! BEWARE must be for all equilibria but maybe not allowed when threaded
!
   integer iph,jl,tuple
   logical force
-----------------
    subroutine suspend_somephases(mode,invph,dim1,dim2,ceq)
! This was added to handle calculating restricted equilibria during mapping
! to suspend (mode=1) or restore (mode=0) phases not involved
! in an invariant equilibrium.
! invph is array with phases that are involved, it has dimension (dim1,*)
! the current status is saved and restored
    implicit none
    integer mode,dim1,dim2,invph(dim1,*)
    type(gtp_equilibrium_data), pointer :: ceq
-------------------
      subroutine delete_unstable_compsets(lokph,ceq)
! This was added to explictly delete unstable composition sets with AUTO set
! Compsets will be shifted down if a stable compset is after an unstable
! See subroutine TOTO_AFTER in gtp3Y.F90
!
    implicit none
    integer lokph
    type(gtp_equilibrium_data), pointer :: ceq
```

8.6.8 Enter parameter

All kind of parameters in all kinds of phases are entered by this subroutine. Called when reading a TDB file or when the parameter is entered interactively, see 8.9.2.

```
 subroutine enter_parameter(lokph,typty,fractyp,nsl,endm,nint,lint,ideg,&
      lfun,refx)
! enter a parameter for a phase from database or interactivly
! enter_parameter_inter(activly) is in gtp3D for some unknown reason ...
! typty is the type of property, 1=G, 2=TC, ... , n*100+icon MQ&const#subl
! fractyp is fraction type, 1 is site fractions, 2 disordered fractions
! nsl is number of sublattices
! endm has one constituent index for each sublattice
! constituents in endm and lint should be ordered so endm has lowest
! (done by decode_constarr)
! nint is number of interacting constituents (can be zero)
! lint is array of sublattice+constituent indices for interactions
! ideg is degree
! lfun is link to function (integer index) if -1 used for listing
! refx is reference (text)
! if this is a phase with permutations all interactions should be in
! the first or the first two identical sublattices (except interstitals)
! a value in endm can be negative to indicate wildcard
! for ionic liquid constituents must be sorted specially
   implicit none
   integer, dimension(*) :: endm
   character refx*(*)
   integer lokph,fractyp,typty,nsl,nint,ideg,lfun
   integer, dimension(2,*) :: lint
```

8.6.9 Subroutines handling fcc permutations

These subroutines creates all possible permutations of parameters for a 4 sublattice fcc phase up to ternary interactions. The 4 ordering sublattices must be the first and they represent the tetrahedron in the lattice. The number of sites must be the same and the constituents also. There can be additional sublattices for interstitials. It is very complictated. See section 4.6.1.

```
 subroutine fccpermuts(lokph,nsl,iord,noperm,elinks,nint,jord,intperm,intlinks)
! finds all fcc/hcp permutations needed for this parameter
! The order of elements in the sublattices is irrelevant when one has F or B
! ordering as all permutations are stored in one place (with some exceptions)
! Thus the endmembers are ordered alphabetically in the sublattices and also
! the interaction parameters.  Max 2 levels of interactions allowed.
   implicit none
   integer, dimension(*) :: iord,intperm
   integer, dimension(2,*) :: jord
   integer lokph,nsl,noperm,nint
------------------
 subroutine fccip2A(lokph,jord,intperm,intlinks)
! 2nd level interaction permutations for fcc
   implicit none
   integer, dimension(*) :: intperm
   integer, dimension(2,*) :: jord,intlinks
   integer lokph
---------------------
 subroutine fccip2B(lq,lokph,lshift,jord,intperm,intlinks)
! 2nd level interaction permutations for fcc
   implicit none
   integer lq,lokph,lshift
   integer, dimension(*) :: intperm
   integer, dimension(2,*) :: jord,intlinks
------------------------
 subroutine fccint31(jord,lshift,intperm,intlinks)
! 1st level interaction in sublattice l1 with endmember A:A:A:B or A:B:B:B
! set the sublattice and link to constituent for each endmember permutation
! 1st permutation of endmember: AX:A:A:B; A:AX:A:B; A:A:AX;B  4      0 1 2
! 2nd permutation of endmember: AX:A:B:A; A:AX:B:A; A:A:B:AX  3      0 1 3
! 3rd permutation of endmember: AX:B:A:A; A:B:AX:A; A:B:A:AX  3      0 2 3
! 4th permutation of endmember: B:AX:A:A; B:A:AX:A; B:A:A:AX  1 or   1 2 3
! 1st permutation of endmember: A:BX:B:B; A:B:BX:B; A:B:B:BX  4      0 1 2
! 2nd permutation of endmember: BX:A:B:B; B:A:BX:B; B:A:B:BX  1 etc -1 1 2
! 3rd -1 0 2 ; -1 0 1
! suck
   implicit none
   integer lshift
   integer, dimension(2,*) :: jord,intlinks
   integer, dimension(*) :: intperm
----------------------
 subroutine fccint22(jord,lshift,intperm,intlinks)
! 1st level for endmember A:A:B:B with interaction in sublattice jord(1,1)
! 6 permutations of endmember, 2 permutations of interactions, 12 in total
! 1st endmemperm: AX:A:B:B; A:AX:B:B      0  1
! 2nd endmemperm: AX:B:A:B; A:B:AX:B      0  2
! 3rd endmemperm: AX:B:B:A; A:B:B:AX      0  3
! 4th endmemperm: B:AX:B:A; B:A:B:AX      1  3
! 5th endmemperm: B:B:AX:A; B:B:A:AX      2  3
! 6th endmemperm: B:AX:A:B; B:A:AX:B or   1  2
! 1th endmemperm: A:A:BX:B; A:A:B:BX      0  1
! 2nd endmemperm: A:BX:A:B; A:B:A:BX     -1  1
! 3rd endmemperm: A:BX:B:A; A:B:BX:A     -1  0
! 4th endmemperm: BX:A:B:A; B:A:BX:A     -2  0
! 5th endmemperm: BX:B:A:A; B:BX:A:A     -2 -1
! 6th endmemperm: BX:A:A:B; B:A:A:BX     -2  1
   implicit none
   integer lshift
   integer, dimension(2,*) :: jord,intlinks
   integer, dimension(*) :: intperm
-----------------
 subroutine fccint211(a211,jord,lshift,intperm,intlinks)
! 1st level interaction in sublattice l1 with endmember like A:A:B:C
! 12 endmember permutations of AABC; ABBC; or ABCC
! 2 interaction permutations for each, 24 in total
   implicit none
   integer a211,lshift
   integer, dimension(2,*) :: jord,intlinks
   integer, dimension(*) :: intperm
-----------------
 subroutine fccpe211(l1,elinks,nsl,lshift,iord)
! sets appropriate links to constituents for the 12 perumations of
! A:A:B:C (l1=1), A:B:B:C (l1=2) and A:B:C:C (l1=3)
   implicit none
   integer l1,nsl,lshift
   integer, dimension(nsl,*) :: elinks
   integer, dimension(*) :: iord
------------------
 subroutine fccpe1111(elinks,nsl,lshift,iord)
! sets appropriate links to 24 permutations when all 4 constituents different
! A:B:C:D
! The do loop keeps the same constituent in first sublattice 6 times, changing
! the other 3 sublattice, then changes the constituent in the first sublattice
! and goes on changing in the other 3 until all configurations done
   implicit none
   integer nsl,lshift
   integer, dimension(nsl,*) :: elinks
   integer, dimension(*) :: iord
--------------------
 logical function check_minimal_ford(lokph)
! some tests if the fcc/bcc permutation model can be applied to this phase
! The function returns FALSE if the user may set the FORD or BORD bit of lokph
   implicit none
   integer lokph
```

8.6.10 Subroutines handling bcc permutations

Finally implemented and works!! I hope they have no bugs because I never have to look at these subroutines again.

```
 subroutine bccpermuts(lokph,nsl,iord,noperm,elinks,nint,jord,intperm,intlinks)
! finds all bcc permutations needed for this parameter
   implicit none
   integer lokph,nsl,noperm,nint
! iord are the endmember constituent indices
! intperm has dimension 24 and contain propagation of interactions ??
   integer, dimension(*) :: iord,intperm
! jord(1,int) is the interaction subl. and jord(2,int) the constituent index
   integer, dimension(2,*) :: jord
! these must be allocated here and will be stored in the parameter records
! giving the constituent indices for permutations of endmembers and interactions
   integer, dimension(:,:), allocatable :: elinks
   integer, dimension(:,:), allocatable :: intlinks
 =======================
 subroutine bccendmem(lokph,nsl,elal,noperm,elinks)
! generate an bcc endmember with all permutations
   implicit none
   integer lokph,nsl,noperm
! elal are the endmember species indices
   integer, dimension(*) :: elal
! these must be allocated here and will be stored in the parameter records
! giving the sublattice and constituent indices for each permutation
! of an endmembers
   integer, dimension(:,:), allocatable :: elinks
!    integer, dimension(:,:), allocatable :: intlinks
 =======================
 subroutine bccint1(lokph,nsl,elal,noperm,elinks,nint,jord,intperm,intlinks)
! generate all bcc permutations for a first order interaction
   implicit none
! on entry noperm is the number of permutation of the endmember
! on exit  noperm is the number of permutation of the interaction
   integer lokph,nsl,noperm,nint
! elal are the endmember species indices
   integer, dimension(*) :: elal
! intperm has dimension 24 and contain propagation of interactions ??
   integer, dimension(*) :: intperm
! jord(1,int) is the interaction subl. and jord(2,int) the constituent index
   integer, dimension(2,*) :: jord
! these contain the already allocated permutation of the endmember
!    integer, dimension(:,:), allocatable :: elinks
   integer, dimension(nsl,*) :: elinks
! intlinks will be allocated here and will be stored in the parameter records
! giving the constituent indices for permutations of the interactions
! It may be reallocated if the interaction is second level
   integer, dimension(:,:), allocatable :: intlinks
 =======================
 subroutine bccint2(lokph,nsl,elal,noperm,elinks,nint,jord,intperm,intlinks)
! generate all bcc permutations needed for a ternary or reciprocal parameter
   implicit none
! on entry noperm is the number of permutations of the first interaction
! on exit  noperm is the number of permutations of the second interaction
   integer lokph,nsl,noperm,nint
! elal are the endmember species indices
   integer, dimension(*) :: elal
! intperm has dimension 24 and contain propagation of interactions ??
   integer, dimension(*) :: intperm
! jord(1,int) is the interaction subl. and jord(2,int) the constituent index
   integer, dimension(2,*) :: jord
! elinks are the permutations of the endmember
!    integer, dimension(:,:), allocatable :: elinks
   integer, dimension(nsl,*) :: elinks
! on entry intlinks are the permutations of the first interaction
! on exit  intlinks are the permutations of the second interaction
! it must be deallocated and reallocated using int1links
   integer, dimension(:,:), allocatable :: int1links
   integer, dimension(:,:), allocatable :: intlinks
```

8.6.11 Enter the bibliographic reference for parameter data

Bibliographic information for parameters is provided by this subroutine.

```
 subroutine tdbrefs(refid,line,mode,iref)
! store a reference from a TDB file or given interactivly
! If refid already exist and mode=1 then amend the reference text
   implicit none
   character*(*) refid,line
   integer mode,iref
```

8.6.12 Enter equilibrium

The equilibrium record, as explained in section 5.2.5 has all data necessary for specifying an equilibrium: the conditions, components, phases etc. See also Fig. 1 One may have several equilibria with different sets of conditions but they have the same set of phases (the phase status and set of stable phases may differ). This can be used to assess many different experiments or used in simulations where each gridpoint is connected to an equilibrium record. This simplifies parallel processing as each thread can work independently on the data in the equilibrium record.

```
 subroutine enter_equilibrium(name,number)
! creates a new equilibrium.  Allocates arrayes for conditions
! components, phase data and results etc.
! returns index to new equilibrium record
! THIS CAN PROBABLY BE SIMPLIFIED, especially phase_varres array can be
! copied as a whole, not each record structure separately ... ???
   implicit none
   character name*(*)
   integer number
---------------
 subroutine geneqname(text)
! creates a equilibrium name like EQ_x where x is the freeq in text
   implicit none
   character text*(*)
```

8.6.13 Enter many equilibria

This command is specially designed for entering table of experimental data where most conditions are the same but only some values are different. It will ask for a table head which must contain all information needed to calculate the equilibrium. By default all phases are suspended so first give the set of phases to be considered (entered, fixed or dormant). The line with the phase status start with the status, for fix and entered followed by a number and then a list of phases. An asterisk, *, can be used for all phases.

The conditions can be set after just the word "conditions" and experiments after the word "experiment". It is also possible to set reference states and to demand that some symbols are calculated or values of state variables and parameter identifiers listed.

Each line may refer to columns in the table to follow after the head. The columns are specified with teh "@" character followed by the column. Max 9 columns allowed.

The head is finished by the command "table_start" and then on each line the column values must be given. But first on each line there must be a name of the equilibrium (this is not counted as a column, or as column zero).

The lines with values is terminated by a "table_end".

All equilibria in a table can be calculated with the "calculate all" command after the range of equilibria has been set by the "set range" command.

```
 subroutine enter_many_equil(cline,last,pun)
! executes an enter many_equilibria command
! and creates many similar equilibria from a table
! pun is file units for storing experimental dataset, pun(i)>0 if i is open
   implicit none
   character*(*) cline
   integer last,pun(9)
```

8.6.14 Enter data for the MQMQA model

The Modified Quasichemical Model with Quadruplet Approximation (MQMQA) is descibed in detail in Appendix E. It requires some special subroutines for entering the constituents.

Note in particular that mixed quadruplets, AB/X, A/XY and AB/XY are endmembers and must have a Gibbs energy relative to the reference state for the pure elements. All such quadruplets are automatically generated in some software even if there are no parameters for these quadruplets. For the OC implementation all quadruplets must be entered explicitly because sometimes very odd stoichiometries are used. The contribution to the Gibbs energy from the reference state is added automatically, although not listed. A warning about this is issued when listing the data for the MQMQA phase. In OC is is also possible to plot the fractions of such constituents.

```
 subroutine mqmqa_constituents(inline,const,nend,loop)
! can take input from database file or terminal
! add constituent data in mqmqa_data record
! inline is a char with "specie_1,specie_2/specie_3,specie_4 n1 n1 n3 n4 ..."
! all specie_i must be entered
! loop is zero at first call which allocates arrays
! const is array of all quadruplet names, it is returned to calling enterphase
! or readtdb routine
   implicit none
   integer loop,nend
! dimension maxconst! check for overflow
   character inline*(*),const(*)*24
   type(gtp_equilibrium_data), pointer :: ceq
 =======================
 subroutine mqmqa_rearrange(const)
! This routine will scan the mqmqa_data datastructure
! and for all non-endmembers replace links to species by liknks to endmembers
! and calculate an store several useful things
! NOTE the phase is not yet entered!!
   implicit none
! array with names of quadrupole constituents, needed by enter phase!!
   character const(*)*24
 =======================
 subroutine mqmqa_quadbonds(index,values)
! This routine will add missing quads using the pairs
   implicit none
   integer index
   double precision values(*)
 =======================
 subroutine mqmqa_addquads
! This routine will add missing quads using the pairs
   implicit none
```

8.6.15 Enter data for UNIQUAC model

In the UNIQUAC species the configurational entropy depend on the vaolume and surface area of each species. These values can be entered here.

```
 subroutine enter_species_property(loksp,nspx,value)
! enter an extra species property for species loksp
   implicit none
   integer loksp,nspx
   double precision value
--------------
 subroutine set_uniquac_species(loksp)
! set the status bit and allocates spexttra array
   implicit none
   integer loksp
```

8.6.16 Enter material

This subroutine is a kind of supercommand or shortcut to enter a material with a known composition from a database and calculate an equilibrium at a given T and P. It is then possible to calculate a diagram or vary the composition of the material using different "set" commands.

```
 subroutine enter_material(cline,last,nv,xknown,ceq)
! enter a material from a database
! called from user i/f
   implicit none
   integer last,nv
   character cline*(*)
   double precision xknown(*)
   type(gtp_equilibrium_data), pointer :: ceq
```

8.7 Delete and copy things

It can be dangerous to delete different parts of the data structure because it may leave "dangling links" to data that is removed.

8.7.1 Delete all conditions

When amending the set of components all conditions are also deleted for all equilibria. This subroutine is also used when one gives the command SET CONDITION *:=NONE. To avoid memory leaks it is also used when deleting an equilibrium.

```
 subroutine delete_all_conditions(mode,ceq)
```

```
! deletes the (circular) list of conditions in an equilibrium
! it also deletes any experiments
! if mode=1 the whole equilibrium is removed, do not change phase status
! because the phase_varres records have been deallocated !!!
! I am not sure it releases any memory though ...
   implicit none
   integer mode
   type(gtp_equilibrium_data), pointer :: ceq
```

8.7.2 Delete equilibria

This is needed after STEP or MAP to clean up the structure as all equilibria along the lines are saved as equilibrium records.

```
 subroutine delete_equilibria(name,ceq)
! deletes equilibria (needed when repeated step/map)
! name can be an abbreviation line "_MAP*"
! deallocates all data.  Minimal checks ... one cannot delete "ceq"
   implicit none
   character name*(*)
   type(gtp_equilibrium_data), pointer :: ceq
```

8.7.3 Copy equilibrium

As part of STEP and MAP equilibrium records are copied between different lists. The second variant also specifies the index of the equilibrium in the eqlista array.

```
 subroutine copy_equilibrium(neweq,name,ceq)
! creates a new equilibrium which is a copy of ceq.
   implicit none
   character name*(*)
   type(gtp_equilibrium_data), pointer ::neweq,ceq
---------------
 subroutine copy_equilibrium2(neweq,number,name,ceq)
! creates a new equilibrium which is a copy of ceq. THIS IS STILL USED !! ??
! Allocates arrayes for conditions,
! components, phase data and results etc. from equilibrium ceq
! returns a pointer to the new equilibrium record
! THIS CAN PROBABLY BE SIMPLIFIED, especially phase_varres array can be
! copied as a whole, not each record structure separately ... ???
   implicit none
   character name*(*)
   integer number
   type(gtp_equilibrium_data), pointer ::neweq,ceq
```

8.7.4 Copy condition

This is a utility used in MAP and STEP. Note that when copying an equilibrium record the conditions which are linked withing the “ceq” records are NOT copied but points to the same records as in the original “ceq” record. This can cause problems ...

```
 subroutine copy_condition(newrec,oldrec)
! Creates a copy of the condition record "oldrec" and returns a link
! to the copy in newrec.  The links to "next/previous" are nullified
   implicit none
   type(gtp_condition), pointer :: oldrec
   type(gtp_condition), pointer :: newrec
```

8.7.5 Calculate new highcs when a composition set is created or deleted

Highcs is used when saving the allocated phase_varres records and in some other cases. Note that there can be unused phase_varres records below highcs. The handling of reserving and releasing records in the phase_varres array is not well implemented but it seems to work.

```
 integer function newhighcs(reserved)
! updates highcs and arranges csfree to be in sequential order
! highcs is the higest used varres record before the last reservation
! or release of a record.  release is TRUE if a record has been released
! csfree is the beginning of the free list of varres records.
   implicit none
   logical reserved
```

8.8 List things, gtp3C

The routines in this section are intended for the line oriented user interface of GTP. It lists data assuming 80 column width of the screen. In some cases a character variable is returned but in most case the list unit is provided in the call. This can be the screen or a file.

Some listings are described in connection with the objects that are listed, see 8.15.14.

8.8.1 List data for all elements

The element data is listed. Second version for TDB files.

```
 subroutine list_all_elements(unit)
! lists elements
   implicit none
   integer unit
---------------
```

```
 subroutine list_all_elements2(unit)
! lists elements
   implicit none
   integer unit
```

8.8.2 List data for all components

The components may be different in each equilibrium.

```
 subroutine list_all_components(unit,ceq)
! lists the components for an equilibrium
   implicit none
   integer unit
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.3 List data for one element

The data for element “elno” in written to the character variable text from position ipos.

```
 subroutine list_element_data(text,ipos,elno)
   implicit none
   character text*(*)
   integer ipos,elno
```

8.8.4 List data for one species

The data for species “spno” in written to the character variable text from position ipos. The second version is suitable for TDB files.

```
 subroutine list_species_data(text,ipos,spno)
   implicit none
   character text*(*)
   integer ipos,spno
---------------
 subroutine list_species_data2(text,ipos,loksp)
! loksp is species record ...
   implicit none
   character text*(*)
   integer ipos,loksp
```

8.8.5 List data for all species

One line for each species is listed on device unit.

```
 subroutine list_all_species(unit)
   implicit none
   integer unit
```

8.8.6 List sorted phases

The phases are listed with one line each, first the stable phases, then the entered but unstable in decreasing order of stability. If more than 10 phases the remaining are merged into a single line.

```
 subroutine list_sorted_phases(unit,ceq)
! short list with one line for each phase
! suspended phases merged into one line
! stable first, then entered ordered in driving force order, then dormant
! also in driving force order.  Only 10 of each, the others lumped together
   implicit none
   integer unit
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.7 List one line of data for all phases

One line for each phase is listed on device unit for equilibrium ceq.

```
 subroutine list_all_phases(unit,ceq)
! short list with one line for each phase
! suspended phases merged into one line
   implicit none
   integer unit
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.8 List global results

This is part of the "list_result" command in the GTP user i/f.

```
 subroutine list_global_results(lut,ceq)
! list G, T, P, V and some other things
   implicit none
   integer lut
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.9 List components result

This is part of the "list_result" command in the GTP user i/f.

```
 subroutine list_components_result(lut,mode,ceq)
! list one line per component (name, moles, x/w-frac, chem.pot. reference state
! mode 1=mole fractions, 2=mass fractions
   implicit none
   integer lut,mode
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.10 List all phases with positive dgm

This is part of the "list_result" command in the GTP user i/f. If a phase has positive DGM it should either be dormant or there has been an error calculating the equilibrium.

```
 subroutine list_phases_with_positive_dgm(mode,lut,ceq)
! list one line for each phase+comp.set with positive dgm on device lut
! The phases must be dormant or the result is in error.  mode is not used
   implicit none
   integer mode,lut
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.11 List results for one phase

This is part of the "list_result" command in the GTP user i/f. It lists normally only the stable phases with their amounts and compositions. With different values of mode the listing can be changed. Rather clumsy.

```
 subroutine list_phase_results(iph,jcs,mode,lut,once,ceq)
! list results for a phase+comp.set on lut
! mode specifies the type and amount of results,
! unit digit:   0=mole fraction,      othewise mass fractions
! 10th digit:   0=only composition,   10=also constitution
! 100th digit:  0=value order,        100=alphabetical order
! 1000th digit: 0=all phases,         1000=only stable phases
! 10000th digit: 1= constituent fractions times formula unit of phase (Solgas)
   implicit none
   integer iph,jcs,mode,lut
   logical once
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine list_short_results(lut,ceq)
! list short results for all stable phases (for debugging) lut
   implicit none
   integer lut
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.12 Formatted output for constitution

This subroutine formats the output of composition or constitution in nice columns trying to use as few lines as possible.

```
 subroutine format_phase_composition(mode,nv,consts,vals,lut)
! list composition/constitution in alphabetical or value order
! entalsiffra 0 mole fraction, 1 mass fraction, 3 mole percent, 4 mass percent
! tiotalsiffra alphabetical order ... ??
! mode >100 else alphabetical order
! nv is number of components/constitunents (in alphabetical order in consts)
! components/constituents in consts, fractions in vals
   implicit none
   integer nv,mode,lut
   character consts(nv)*(*)
   double precision vals(nv)
```

8.8.13 List data on SCREEN or TDB, LaTeX or macro format

This subroutines list the model parameters for all phases in a selected format. Only SCREEN is implemented.

```
 subroutine list_many_formats(cline,last,ftyp,unit1)
! lists all data in different formats: SCREEN/TDB/MACRO/LaTeX/ODB
!                                         1    2    3     ???
! unfinished
   implicit none
   character cline*(*)
   integer last,unit1,ftyp
------------------
 subroutine list_TDB_format(filename)
! lists all data TDB format
   implicit none
   character filename*(*)
```

8.8.14 List some phase model stuff

This is probably redundant but can be used to check the conversion from site fractions to disordered fractions for phases with several fraction sets.

```
 subroutine list_phase_model(iph,ics,lut,CHTD,ceq)
! list model (no parameters) for a phase on lut
   implicit none
   integer iph,ics,lut
   TYPE(gtp_equilibrium_data), pointer :: ceq
   character CHTD*1
```

8.8.15 List all parameter data for a phase

This is the big listing of the model and data for a phase. It lists the sublattices, sites, constituents. Then all endmembers and all interaction parameters.

The second version is suitable for TDB files.

```
 subroutine list_phase_data(iph,CHTD,lut)
! list parameter data for a phase on unit lut
   implicit none
   integer iph,lut
   character CHTD*1
---------------
 subroutine list_phase_data2(iph,ftyp,CHTD,lut)
! list parameter data for a phase on unit lut in ftyp format, ftyp=2 is TDB
   implicit none
   integer iph,lut,ftyp
   character CHTD*1
```

8.8.16 Format expression of references for endmembers

When listing an endmember parameter for the Gibbs energy this subroutine subtracts the H298 expression.

```
 subroutine subrefstates(funexpr,jp,lokph,parlist,endm,noelin1)
! list a sum of reference states for a G parameter
! like "-H298(BCC_A2,FE)-3*H298(GRAPITE,C)"
   implicit none
   integer jp,lokph,parlist,endm(*)
   character funexpr*(*)
   logical noelin1
```

8.8.17 Encode/decode stoichiometry of species

These subroutines generate a stoichiometric formula for a species including a charge and vice versa.

```
 subroutine encode_stoik(text,ipos,spno)
! generate a stoichiometric formula of species from element list
   implicit none
   integer ipos,spno
   character text*(*)
```

```
----------------------
 subroutine decode_stoik(name,noelx,elsyms,stoik)
! decode a species stoichiometry in name to element index and stoichiometry
! all in upper case
   implicit none
   character name*(*),elsyms(*)*2
   double precision stoik(*)
   integer noelx
```

8.8.18 Encode/decode constituent array for parameters

These subroutines generates a constituent array for a parameter or decodes one. Constituents are species. Constituents in different sublattices are separated by “:”, interacting constituents in same sublattice are separated by “,”. The degree is written after a “;”.

```
 subroutine encode_constarr(constarr,nsl,endm,nint,lint,ideg)
! creates a constituent array
   implicit none
   character constarr*(*)
   integer, dimension(*) :: endm
   integer nsl,nint,ideg
   integer, dimension(2,*) :: lint
-----------------------
 subroutine decode_constarr(lokph,constarr,nsl,endm,nint,lint,ideg)
! deconde a text string with a constituent array
! a constituent array has <species> separated by , or : and ; before degree
   implicit none
   character constarr*(*)
   integer endm(*),lint(2,*)
   integer nsl,nint,ideg,lokph,lord
```

8.8.19 List parameter data references

This subroutine lists the source of one or several bibliographic references for the parameters.

```
 subroutine list_bibliography(bibid,lut)
! list bibliographic references
   implicit none
   integer lut
   character bibid*(*)
```

8.8.20 List conditions on a file or screen

The heading says all. After the list the degrees of freedom is shown.

```
 subroutine list_conditions(lut,ceq)
! lists conditions on lut
   implicit none
   integer lut
   type(gtp_equilibrium_data), pointer :: ceq
```

8.8.21 Extract one condition or experiment in a character veriable

A single condition is written in a character variable. The get_one_experiment routine returns the a text line describing an experiment.

```
 subroutine get_one_condition(ip,text,seqz,ceq)
! list the condition with the index seqz into text
! It lists also fix phases and conditions that are not active
   implicit none
   integer ip,seqz
   character text*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine get_one_experiment(ip,text,seqz,eval,ceq)
! list the experiment with the index seqz into text
! It lists also experiments that are not active ??
! UNFINISHED current value should be appended
   implicit none
   integer ip,seqz
   character text*(*)
   logical eval
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.22 List condition in character variable

All current active conditions in equilibrium ceq is written to the character variable text. This can be written on the screen or used for other purposes. It will also be used for experiments (not implemented yet).

```
 subroutine get_all_conditions(text,mode,ceq)
! list all conditions if mode=0, experiments if mode=1, -1 if no numbers
   implicit none
   integer mode
   character text*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.23 Get the degrees of freedom

The current number of degrees of freedom. To calculate an equilibrium it must be zero.

```
 integer function degrees_of_freedom(ceq)
! returns the degrees of freedom
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.8.24 List model parameter identifiers

The GTP package allows definition of new properties that can be modeled as dependent on the constitution of each phase. Such properties must be defined in the software and they can be listed with this subroutine. Many additions depend on such parameter properties like the Curie temperature and the Einstein temperature.

On can also add properties that does not affect the Gibbs energy but which depend on the constitution of the phase like the mobility, resistivity, lattice parameter etc.

```
 subroutine list_defined_properties(lut)
! lists all parameter identifiers allowed
   implicit none
   integer lut
```

8.8.25 Find defined properties

Although properties like TC (Curie temperature) and BMAG (Average Bohr magneton number) are not state variables they can be listed using the command LIST STATE_VARIABLEs and their values can be obtained by the same subroutines that are used for state variables like get_state_variable. They use the following subroutine to find the properties defined in the gtp_propid structure.

```
 subroutine find_defined_property(symbol,mode,typty,iph,ics)
! searches the propid list for one with symbol or identifiction typty
! if mode=0 then symbol given, if mode=1 then typty given
! symbol can be TC(BCC), BM(FCC), MQ&FE(HCP) etc, the phase must be
! given in symbol as otherwise it is impossible to find the consititent!!!
! A constituent may have a sublattice specifier, MQ&FE#3(SIGMA)
   implicit none
   integer mode,typty,iph,ics
   character symbol*(*)
-------------------------
 subroutine find_defined_property3(symbol,mode,typty,iph,ics)
! Revised version of old routine called by get_many_svar
! allows wildcards in some cases and should handle # and * better ...
! searches the propid list for one with symbol or identifiction typty
! if mode=0 then symbol given, if mode=1 then typty given
! symbol can be TC(BCC), BM(FCC), MQ&FE(HCP) etc, the phase must be
! given in symbol as otherwise it is impossible to find the consititent!!!
! A constituent may have a sublattice specifier, MQ&FE#3(SIGMA)
```

```
   implicit none
   integer mode,typty,iph,ics
   character symbol*(*)
```

8.8.26 List some odd details

Creating a line with all phase that are stable (dgm=0). It is used when listing step/map resuls.

```
 subroutine line_with_phases_withdgm0(line,ceq)
! used in amend lines with stored STEP/MAP results
! enter first 6, two .. and last 2 characters of phase names with abs(dgm)<1-8
! line LIQUID#2, PHYRRO..#2
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   character line*(*)
---------------------
 subroutine list_equilibria_details(mode,teq)
! not used yet ... ??
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: teq
   integer mode
```

8.8.27 List an error message if any

The error code is a global variable gxanytime. If not zero there has been an error detected and usually there is a message associated with the value. This routine can list the message.

```
 logical function gtp_error_message(reset)
! tests the error code and writes the error message (if any)
! and reset error code if reset=0
! if reset >0 that is set as new error message
! if reset <0 the error code is not changed
! return TRUE if error code set, FALSE if error code is zero
   implicit none
   integer reset
```

8.8.28 List coefficients used in an assessment

Listing coefficients used in assessment of model parameters.

```
 subroutine listoptcoeff(mexp,error2,done,lut)
! listing of optimizing coefficients
    integer lut,mexp
```

```
! error2 is an array with 1: old error, 2: new error; 3: normalized error
    double precision error2(*)
    logical done
!    integer lut,mexp
!    double precision errs(*)
!    type(gtp_equilibrium_data), pointer :: ceq
```

8.9 Interactive subroutines, gtp3D

The current user interface to OC and GTP is command oriented and there are subroutines provided in GTP to enter, set, list and get many things. Most subroutines where the user is expected to provide information is collected in this section.

8.9.1 Ask for phase constitution

The used can interactivly set the default constitution or enter a constitution for a specific phase and composition set.

```
 subroutine ask_phase_constitution(cline,last,iph,ics,lokcs,ceq)
! interactive input of a constitution of phase iph
   implicit none
   integer last,iph,ics,lokcs
   character cline*(*)
---------------
 subroutine ask_phase_new_constitution(cline,last,iph,ics,lokcs,ceq)
! interactive input of a constitution of phase iph
   implicit none
   integer last,iph,ics,lokcs
   character cline*(*)
```

8.9.2 Ask for parameter

The user can enter a model parameter with this subroutine. The routine that enters the parameter in the data structure is decribed in section 8.6.8.

```
 subroutine enter_parameter_interactivly(cline,ip,mode)
! enter a parameter from terminal or macro
! NOTE both for ordered and disordered fraction set !!
! mode = 0 for entering
!       1 for listing on screen (kou)
   implicit none
   integer ip,mode
   character cline*(*)
```

8.9.3 Amend global bits

There are a number of global bits that can be set by this subroutine.

```
 subroutine amend_global_data(cline,ipos)
   implicit none
   character cline*(*)
   integer ipos
```

8.9.4 Ask for reference of parameter data

Each parameter in a model should have a bibliographic reference, preferably a published paper. When a parameters is entered by calling enter_parameter_interactivly the reference is asked for but with this routine it is possible to enter such a reference separately.

```
 subroutine enter_bibliography_interactivly(cline,last,mode,iref)
! enter a reference for a parameter interactivly
! mode=0 means enter, =1 amend
   implicit none
   character cline*(*)
   integer last,mode,iref
   logical twotries
```

8.9.5 Enter an experiment

With this subroutine the state variable for an experiment, its value and uncertainty is given for equilibrium ceq. The experiment can also be changed or removed (set equal to NONE).

The logical function is used to check if two state variable records represent the same state variable (because associated would only be true if they were the same record). o

```
 subroutine enter_experiment(cline,ip,ceq)
! enters an experiment, almost the same as set_condition
   implicit none
   character cline*(*)
   integer ip
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 logical function same_statevariable(svr1,svr2)
! returns TRUE if the state variable records are identical
   type(gtp_state_variable), pointer :: svr1,svr2
```

8.9.6 Set a condition

This is the central routine to set a condition for an equilibrium calculation. An alternative is the set_input_amount. When setting the status of a phase as fixed, see section 8.13.4, this subroutine is called automatically to add this as condition.

```
 subroutine set_condition(cline,ip,ceq)
! to set a condition
   implicit none
   character cline*(*)
   integer ip
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine set_cond_or_exp(cline,ip,new,notcond,ceq)
! decode an equilibrium condition, can be an expression with + and -
! the expression should be terminated with an = or value supplied on next line
! like "T=1000", "x(liq,s)-x(pyrrh,s)=0", "mu(cr)-1.5*mu(o)=muval"
! Illegal with number before first state variable !!!
! It can also be a "NOFIX=<phase>" or "FIX=<phase> value"
! The routine should also accept conditions identified with the "<number>:"
! where <number> is that preceeding each condition in a list_condition
! It should also accept changing conditions by <number>:=new_value
! The pointer to the (most recent) condition or experiment is returned in new
! notcond is 0 if a condition should be created, otherwise an experiment
   implicit none
   integer ip,notcond
   character cline*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_condition), pointer :: new
---------------
 subroutine get_experiment_with_symbol(symsym,experimenttype,temp)
! finds an experiment with s symbol index symsym and exp.type
   implicit none
   integer symsym,experimenttype
   type(gtp_condition), pointer :: temp
! NOTE: temp must have been set to ceq%lastcondition before calling this
```

8.9.7 Get the condition record

A condition is typically a state variable assigned a value like $T = 1273$ but it can be much more complicated. One can have conditions like $x(\mathrm{liquid,S}) - x(\mathrm{pyrrhotite,S}) = 0$ to specify the congruent melting point of pyrrhotite. A condition can also be that a phase is fixed, i.e. prescribed to be stable.

A condition is specified by the number of terms, the state variable with possible indices (to specify phase, component or constituent etc), reference state (for chemical potentials) and unit (Joule or calorie or per cent or fraction)

The state variable record was not used when this subroutine was first written and the original version is now called get_condition2. This version is depreciated and should not be used. The subroutine that replaces this has the original name and has a state variable record as argument. An additional subroutine that returns the state variable record is also available.

The first two subroutines return a pointer to the condition record, see gtp_condition for that structure, or an error code if no such condition.

The subroutine to set conditions is decribed in 8.9.6. The first subroutine is when the condition is an expression (array of state variables).

```
 subroutine get_condition_expression(nterm,svrarr,pcond)
! I do not want to change get_condition ,,,,, suck
! finds a condition/experiment record with the given state variable expression
! If nterm<0 svr is irrelevant, the absolute value of nterm is the sequential
! number of the ACTIVE conditions
   implicit none
   integer nterm
   type(gtp_state_variable), dimension(*), target :: svrarr
! NOTE: pcond must have been set to ceq%lastcondition before calling this
! pcond: pointer, to a gtp_condition record for this equilibrium
   type(gtp_condition), pointer :: pcond
-----------------------
 subroutine get_condition(nterm,svr,pcond)
! finds a condition/experiment record with the given state variable expression
! If nterm<0 svr is irrelevant, the absolute value of nterm is the sequential
! number of the ACTIVE conditions
   implicit none
   integer nterm
   type(gtp_state_variable), pointer :: svr
! NOTE: pcond must have been set to ceq%lastcondition before calling this
! pcond: pointer, to a gtp_condition record for this equilibrium
   type(gtp_condition), pointer :: pcond
---------------
 subroutine get_condition2(nterm,coeffs,istv,indices,iref,iunit,pcond)
! finds a condition record with the given state variable expression
! nterm: integer, number of terms in the condition expression
! istv: integer, state variable used in the condition
! indices: 2D integer array, state variable indices used in the condition
! iref: integer, reference state of the condition (if applicable)
! iunit: integer, unit of the condition value
! NOTE: pcond must have been set to ceq%lastcond before calling this routine!!!
! pcond: pointer, to a gtp_condition record for this equilibrium
! NOTE: conditions like expressions x(mg)-2*x(si)=0 not implemeneted
! fix phases as conditions have negative condition variable
   implicit none
   TYPE(gtp_condition), pointer :: pcond
   integer, dimension(4,*) :: indices
   integer nterm,istv,iref,iunit
   double precision coeffs(*)
---------------
 subroutine extract_stvr_of_condition(pcond,nterm)
! finds a condition record with the given state variable record
! returns it as a state variable record !!!
! nterm: integer, number of terms in the condition expression
! pcond: pointer, to a gtp_condition record
   implicit none
   TYPE(gtp_condition), pointer :: pcond
   integer nterm
```

8.9.8 A utility routine to locate a condition record

This is needed by the STEP and MAP commands to find a condition to use as axis variable.

```
 subroutine locate_condition(seqz,pcond,ceq)
! locate a condition using a sequential number
   implicit none
   integer seqz
   type(gtp_condition), pointer :: pcond
   type(gtp_equilibrium_data), pointer :: ceq
```

8.9.9 A utility routine to get the current value of a condition

This subroutine is called several times at each iteration in the equilibrium calculation to formulate the system matrix containing the external conditions. It is still very rudimentary and can be improved.

We do not need to pass the pointer to ceq as that is found via the condition record?

```
 subroutine apply_condition_value(current,what,value,cmix,ccf,ceq)
! This is called when calculating an equilibrium.
! It returns a condition at each call, at first call current must be nullified?
! When all conditions done the current is nullified again
! If what=-1 then return degrees of freedoms and maybe something more
! what=0 means calculate current values of conditions
! calculate the value of a condition, used in minimizing G
! ccf are the coefficients for conditions with several terms
   implicit none
   integer what,cmix(*)
   double precision value,ccf(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_condition), pointer :: current
---------------
 subroutine condition_value(mode,pcond,value,ceq)
```

```
! set (mode=0) or get (mode=1) a new value of a condition.  Used in mapping
   implicit none
   integer mode
   type(gtp_condition), pointer :: pcond
   type(gtp_equilibrium_data), pointer :: ceq
   double precision value
```

8.9.10 Ask for default phase constitution

The user can enter a default constitution for a composition set of a phase. A negative value means a maximum value, a positive means a minimum value.

```
 subroutine ask_default_constitution(cline,last,iph,ics,ceq)
! set values of default constitution interactivly
! phase and composition set already given
   implicit none
   character cline*(*)
   integer last,iph,ics
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.9.11 Set default constitution

The user can enter a "default" constitution of a phase. This can sametimes be used as start values for a calculation. mmyfr is an array with default values for each constituent. Negative value is a maximum, positive a minimum.

```
 subroutine enter_default_constitution(iph,ics,mmyfr,ceq)
! user specification of default constitution for a composition set
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer iph,ics
   real mmyfr(*)
```

8.9.12 Interactive set input amounts

This subroutine allows setting condition by entering the amount of species. The amount of the species is converted to amount of components internally. Redundancy is allowed. If several species contain the same element the amounts are added. For example

set_input_amount N(C1O1)=10, N(H2O1)=5, N(C1H4)=7, N(O2)=20

is translated to the conditions N(C)=17, N(H)=38, N(O)=55.

```
 subroutine set_input_amounts(cline,lpos,ceq)
! set amounts like n(specie)=value or b(specie)=value
! value can be negative removing amounts
! values are converted to moles and set or added to conditions
   implicit none
   integer lpos
   character cline*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.9.13 Utility to decode a model parameter identifier

Used when entering data. The model parameter identifiers are listed in Table 4.

```
 subroutine get_parameter_typty(name1,lokph,typty,fractyp)
! interpret parameter identifiers like MQ&C#2 in MQ&C#2(FCC_A1,FE:C) ...
! find the property associated with this symbol
   implicit none
   integer typty,fractyp,lokph
   character name1*(*)
```

8.10 Save and read data from files, gtp3E

Many of the subroutines in this section are unfinished and many are redundant. The only thing that can be read is a simple TDB file but the UNFORMATTED save seems to work also.

The problem is that whenever a change is made in the data structure these routines must be modified accordingly. And the data structure is still undergoing big changes.

These subroutines are in total disorder.

8.10.1 Save all data

This is the top level routine.

```
 subroutine gtpsave(filename,str)
! save all data on file, unformatted, TDB or macro
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
!
   implicit none
   character*(*) filename,str
```

8.10.2 Save data in LaTeX and other formats

This is just a dream.

```
 subroutine gtpsavelatex(filename,specification)
! save all data on LaTeX format on a file (for publishing)
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
! equilibrium record(s) with conditions, componenets, phase_varres records etc
! anything else?
   implicit none
   character*(*) filename,specification
---------------
 subroutine gtpsavedir(filename,specification)
! save all data on a direct file (random access)
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
! equilibrium record(s) with conditions, componenets, phase_varres records etc
! anything else?
   implicit none
   character*(*) filename,specification
---------------
 subroutine gtpsavetm(filename,str)
! save all data on file in macro format
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
!
   implicit none
   character*(*) filename,str
---------------
 subroutine gtpsavetdb(filename,specification)
```

```
! save all data in TDB format on an file UNFINISHED
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
! equilibrium record(s) with conditions, componenets, phase_varres records etc
! anything else?
   implicit none
   character*(*) filename,specification
```

8.10.3 Save all data unformatted on a file

This is fragile but seems to work. All data are converted to an integer array where pointers are converted to indices in this array. That is the old save format used in Thermo-Calc. Then this array is written unformatted on a file. It can be read back and all data restored.

There is no way to save STEP or MAP results this way as that need a direct (random access) file but experimental data can be saved. Thus it is similar to a PAR file in Thermo-Calc.

It calls a large number of routines included list below.

```
 subroutine gtpsaveu(filename,specification)
! save all data unformatted on an file
! First move it to an integer workspace, then write that on a file
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! state variable functions
! references
! equilibrium record(s) with conditions, componenets, phase_varres records etc
! anything else?
   implicit none
   character*(*) filename,specification
---------------
 subroutine savephases(phroot,iws)
! save data for all phases and store pointer in iws(phroot)
! For phases with disordered set of parameters we must access the number of
! sublattices via firsteq
   implicit none
   integer phroot,iws(*)
---------------
```

```
 subroutine saveequil(lok1,iws)
! subroutine saveequil(lok1,iws,ceq)
! save data for equilibrium record ceq including phase_varres
   implicit none
   integer lok1,iws(*),jeq
---------------
 subroutine svfunsave(loksvf,iws,ceq)
! saves all state variable functions as texts in iws
   implicit none
   integer iws(*),loksvf
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine bibliosave(bibhead,iws)
! saves references on a file
   implicit none
   integer bibhead,iws(*)
---------------
 subroutine saveash(lok,iws)
! saving assessment records
   integer lok,iws(*)
---------------
 integer function ceqrecsize()
! calculates the number of words needed to save an equilibrium record
```

8.10.4 Read unformatted data saved on file

This reverses the action of save unformatted. The OC data structure is filled with data earlier saved on a file. It is fragile.

```
 subroutine gtpread(filename,str)
! read unformatted all data in the following order
! header
! element list
! species list
! phase list with sublattices, endmembers, interactions and parameters etc
! tpfuns
! references
! first equilibrium record with conditions, componenets, phase_varres etc
! state variable functions
! equilibrium record(s) with conditions, componenets, phase_varres, experim etc
! CCI changed to use iso_fortran_env to find file unit number for C++
   use :: iso_fortran_env
! CCI
   implicit none
   character*(*) filename,str
```

```
---------------
 subroutine readphases(kkk,iws)
! read data for phlista and all endmembers etc
! works for test case without disordered fraction test
   implicit none
   integer kkk,iws(*)
---------------
 subroutine readendmem(lokem,iws,nsl,emrec)
! allocates and reads an endmember record and its property record from iws
! emrec is an un-allocated pointer in the parameter tree structure
   implicit none
   integer lokem,nsl,iws(*)
   type(gtp_endmember), pointer :: emrec
---------------
 subroutine readproprec(lokpty,iws,firstproprec)
! allocates and a property record for both endmembers and interactions
   implicit none
   integer lokpty,iws(*)
   type(gtp_property), pointer :: firstproprec
---------------
 subroutine readintrec(lokint,iws,level,intrec)
! allocates and reads an interaction record
   implicit none
   integer lokint,iws(*),level
   type(gtp_interaction), pointer :: intrec
---------------
 subroutine readequil(lokeq,iws,elope)
! subroutine readequil(lokeq,iws,elope,ceq)
! lokeq is index for equilibrium record in iws
   implicit none
   integer lokeq,iws(*),elope
---------------
 subroutine svfunread(loksvf,iws)
! read a state variable function from save file and store it.
! by default there are some state variable functions, make sure
! they are deleted.  Done here just by setting nsvfun=0
   implicit none
   integer loksvf,iws(*)
---------------
 subroutine biblioread(bibhead,iws)
! read references from save file
   implicit none
   integer bibhead,iws(*)
---------------
 subroutine readash(lok,iws)
! reading assessment records
```

```
   integer lok,iws(*)
```

8.10.5 Testing keywords in a TDB file

Some utilities used by the routines reading database files.

```
 logical function iskeyword(text,keyword,nextc)
! compare a text with a given keyword. Abbreviations allowed
! but the keyword and abbreviation must be surrounded by spaces
! nextc set to space character in text after the (abbreviated) keyword
   implicit none
   character text*(*),keyword*(*),key*64
   integer nextc
---------------
 integer function istdbkeyword(text,nextc)
! compare a text with a given keyword. Abbreviations allowed (not within _)
! but the keyword and abbreviation must be surrounded by spaces
! nextc set to space character in text after the (abbreviated) keyword
   implicit none
   character text*(*)
   integer nextc
---------------
 integer function ispdbkeyword(text,nextc)
! compare a text with a given keyword. Abbreviations allowed (not within _)
! but the keyword and abbreviation must be surrounded by spaces
! nextc set to space character in text after the (abbreviated) keyword
   implicit none
   character text*(*)
   integer nextc
```

8.10.6 Read a TDB file

This subroutine can read a TDB file that is not too fancily edited manually. The best is to read a file written from Thermo-Calc. Some TYPE_DEFINITIONS are not handelled. All TYPE_DEFINITIONS should be in the beginning of the TDB file, especially DISORDERD_PART as this has been implemented differently in OC.

```
 subroutine readtdb(filename,nel,selel)
! reading data from a TDB file with selection of elements
!-----------------------------------------------------
! Not all TYPE_DEFS implemented
!-----------------------------------------------------
   implicit none
   integer nel
   character filename*(*),selel(*)*2
---------------
 subroutine readtdbsilent
---------------
 subroutine any_disordered_part(lin,ndisph,disph)
! reading data from a PDB file with selection of elements
!-------------------------------------------------------
! Not all TYPE_DEFS implemented
!-------------------------------------------------------
   implicit none
   integer lin,ndisph
   character disph(*)*(*)
```

8.10.7 Reading a database in PDB format

This format is depreciated. A new generic XML format will be developed.

```
 subroutine readpdb(filename,nel,selel,options)
! reading data from a PDB file with selection of elements
   implicit none
   integer nel
   character filename*(*),selel(*)*2,options*(*)
```

8.10.8 Writing a database in PDB format

NOT IMPLEMENTED. A new format for thermodynamic databases is under development.

```
 subroutine write_pdbformat(unit)
! write a PDB database
   implicit none
   integer unit
```

8.10.9 Check a TDB file exists and extract elements

This is used to check a user typed a correct TDB file name and extracts the elements so the user can select which he wants.

```
 subroutine checkdb(filename,ext,nel,selel)
! checking a TDB/PDB file exists and return the elements
   implicit none
   integer nel
   character filename*(*),ext*4,selel(*)*2
---------------
```

```
 subroutine checkdb2(filename,ext,nel,selel)
! checking a TDB/PDB file exists and return the elements
! It also writes 15 lines from any "DATABASE_INFO" in the file
   implicit none
   integer nel
   character filename*(*),ext*4,selel(*)*2
```

8.10.10 Save data in SOLGASMIX dat format

This routine will attempt to save the current data in OC on a file in the DAT format used by SOLGASMIX. There are also some subroutines handling TP functions involved and they are described in section 8.19.17. NO LONGER SUPPORTED.

```
 subroutine save_datformat(filename,version,kod,ceq)
! writes a SOLGASMIX DAT format file. not (ever?) finished
   implicit none
   integer kod
   character filename*(*),version*(*)
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine incunique(text)
   character text*(*)
---------------
 subroutine expand_wildcards(intconst,nconst,wildloop,iset,lokph)
! Expand a wildcard constituent with all constituents it replaces
! There can be several wildcards
! intconst is the original set of constuents including the wildcards (-99)
! nconst is the number of constituents
! wildloop is set to the number of times the interaction is repeated
! iset is a matrix with the expanded constituents
! phrecord is the phase record where one can find the phase structure
   implicit none
   integer intconst(*)
   integer, allocatable, dimension(:,:) :: iset
   integer nconst,wildloop,lokph
---------------
 subroutine intsort(intc,nint,intx)
! This is just another stupid sorting subroutine
! intc is not changed
   implicit none
   integer intc(*),intx(*),nint
---------------
 subroutine lower_case_species_name(constext,ip,isp)
! writes a species name using lower case for second letter of element
   implicit none
   character constext*(*)
   integer ip,isp
```

8.10.11 Prevent listing of an encrypted database

This is an attempt to prevent listing of parameters read from an encrypted database.

```
 logical function notallowlisting(privil)
! check if user is allowed to list data
   double precision privil
```

8.11 State variable stuff, gtp3F

State variables are important for the setting and extracting results of a calculation. State variables are treated very similarly to Thermo-Calc using symbols like T, P, N, $x(<\text{component}>)$ etc.

The internal syntax of state variables is rather complicated, perhaps it should be revised and defined as a structure? If there are errors or one wants to make modifications it is not easy.

Things like Curie temperature, Einstein temperature, mobilities etc are tread almost like "state variables" although one cannot use them in conditions. Adding more things like elastic constants will be a bit complicated.

The subroutines for manipulations is also a bit complicated and could do with a clean up and renaming.

8.11.1 Get state variable value given its symbol

By providing a state variable as a character variable like T or $x(\text{liquid}, \text{cr})$ this routine returns its current value. Wildcards, "*", are not allowed, see 8.11.2.

A variant checks if a specified phase is stable.

```
 subroutine get_stable_state_var_value(statevar,value,encoded,ceq)
! called with a state variable character
! If the state variable includes a phase it checks if the phase is stable ...
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   character statevar*(*),encoded*(*)
   double precision value
---------------
 subroutine get_state_var_value(statevar,value,encoded,ceq)
! called with a state variable character
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   character statevar*(*),encoded*(*)
   double precision value
```

8.11.2 Get many state variable values

This routine can be called with wildcard, "*", as argument in state variables like $NP(*)$, $x(*, CR)$ etc. It is fragile and currently only available when defining plot axis and some output.

```
 subroutine get_many_svar(statevar,values,mjj,kjj,encoded,ceq)
! called with a state variable name with wildcards allowed like NP(*), X(*,CR)
! mjj is dimension of values, kjj is number of values returned
! encoded used to specify if phase data in phasetuple order ('Z')
! >>>> BIG question: How to do with phases that are note stable?
! If I ask for w(*,Cr) I only want the fraction in stable phases
! but when this is used for GNUPLOT the values are written in a matix
! and the same column in that phase must be the same phase ...
! so I have to have the same number of phases from each equilibria.
!
! CURRENTLY if x(*,*) and x(*,A) mole fractions only in stable phases
! Proposal: use * for all phases, use *S o $ or something else for all stable??
!
! >>>>>>>>>>>>>>>> there is a segmentation fault in this subroutine when
! called from ocplot2 in the map11.OCM
! for the second plot as part of all.OCM
! but not when called by itself.  SUCK
! probably caused by the fact that the number of composition sets are different
! >>>>>>>>>>>>>>>>
! A new segmentation fault for map2 when plotting with 2 maptops and the
! first does not have a new composition set LIQUID_AUTO#2 created in the
! second map.  I do not understand how that has ever worked??
! >>>>>>>>>>>>>>>>
!
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   character statevar*(*),encoded*(*)
   double precision values(*)
   integer mjj,kjj
```

8.11.3 Decode a state variable symbol

This subroutine takes as input a character with a state variable and returns a state variable record with its specification. It can also handle decoding of property parameters symbols like the Curie temperature. The main routine calls the older version of this subroutine with a more complex handling of state variables. This second subroutine will eventually disappear and should not be used.

The new version of this subroutine calls the old but this will be removed in a future release. If there are any changes in the state variables structure several subroutines must be changed like this one, 8.11.6 and 8.11.8.

The subroutine also handle property symbols used in the parameters, see 1.5.2, to make it possible to obtain the value of such a propery after an equilibrium calculation. The value returned in svrindex.

```
 subroutine decode_state_variable(statevar,svr,ceq)
! converts a state variable character to state variable record
   character statevar*(*)
   type(gtp_state_variable), pointer :: svr
   type(gtp_equilibrium_data), pointer :: ceq
! this subroutine using state variable records is a front end of the next:
---------------
 subroutine decode_state_variable3(statevar,istv,indices,iref,iunit,svr,ceq)
! converts an old state variable character to indices
! Typically: T, x(fe), x(fcc,fe), np(fcc), y(fcc,c#2), ac(h2,bcc), ac(fe)
! NOTE! model properties like TC(FCC),MQ&FE(FCC,CR) must be detected
! NOTE: added storing information in a gtp_state_variable record svrec !!
!
! this routine became as messy as I tried to avoid
! but I leave it to someone else to clean it up ...
!
! state variable and indices
! Symbol  no   index1 index2 index3 index4
! T       1    -
! P       2    -
! MU      3    component or phase,component
! AC      4    component or phase,component
! LNAC    5    component or phase,component
!                                           index (in svid array)
! U       10   (phase#set)                    6     Internal energy (J)
! UM      11    "                             6     per mole components
! UW      12    "                             6     per kg
! UV      13    "                             6     per m3
! UF      14    "                             6     per formula unit
! S       2x    "                             7     entropy
! V       3x    "                             8     volume
! H       4x    "                             9     enthalpy
! A       5x    "                            10     Helmholtz energy
! G       6x    "                            11     Gibbs energy
! NP      7x    "                            12     moles of phase
! BP      8x    "                            13     mass of moles
! DG      9x    "                            15     Driving force
! Q       10x   "                            14     Internal stability
! N       11x (component/phase#set,component) 16  moles of components
! X       111   "                            17     mole fraction of components
! B       12x   "                            18     mass of components
! W       122   "                            19     mass fraction of components
! Y       13    phase#set,constituent#subl   20     constituent fraction
!----- model variables <<<< these now treated differently
! TC      -     phase#set                    -      Magnetic ordering T
! BMAG    -     phase#set                    -      Aver. Bohr magneton number
! MQ&     -     element, phase#set           -      Mobility
! LNTH    -     phase#set                    -      LN(Einstein temperature)
!
   implicit none
   integer, parameter :: noos=20
   character*4, dimension(noos), parameter :: svid = &
       ['T   ','P   ','MU  ','AC  ','LNAC','U   ','S   ','V   ',&
        'H   ','A   ','G   ','NP  ','BP  ','DG  ','Q   ','N   ',&
        'X   ','B   ','W   ','Y   ']
!        1      2      3      4      4      6      7      8
   character statevar*(*)
   integer istv,iref,iunit
   integer, dimension(4) :: indices
   type(gtp_equilibrium_data), pointer :: ceq
! I shall try to use this record type instead of separate arguments: !!
!   type(gtp_state_variable), pointer :: svrec
   type(gtp_state_variable), pointer :: svr
```

8.11.4 Calculate molar and mass properties for a phase

This subroutine calculates mole and mass fractions of all components for a phase (mole fractions of components not dissolved is zero). It also returns the total number of moles of components and the mass. In amount the number of moles per formula unit is returned (same as qq(1) in get_phase_data and set_constitution).

The second version used for grid minimization.

```
 subroutine calc_phase_molmass(iph,ics,xmol,wmass,totmol,totmass,amount,ceq)
! calculates mole fractions and mass fractions for a phase#set
! xmol and wmass are fractions of components in mol or mass
! totmol is total number of moles and totmass total mass of components.
! amount is number of moles of components per formula unit.
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer iph,ics
   double precision, dimension(*) :: xmol,wmass
   double precision amount,totmol,totmass
-------------------
 subroutine calc_phase_mol(iph,xmol,ceq)
! calculates mole fractions for phase iph, compset 1 in equilibrium ceq
! used for grid generation and some other things
! returns current constitution in xmol equal to mole fractions of components
```

```
   implicit none
   integer iph
   double precision xmol(*)
   TYPE(gtp_equilibrium_data),pointer :: ceq
```

8.11.5 Sum molar and mass properties for all phases

Sums the mole and mass fractions for all components and also total number of moles and mass over all stable phases using 8.11.4. The second version used to calculate normalizing propertes like V, N and B but also G and S for the whole system. Used when calculating state variable values.

```
 subroutine calc_molmass(xmol,wmass,totmol,totmass,ceq)
! summing up N and B for each component over all phases with positive amount
! Check that totmol and totmass are correct ....
   implicit none
   double precision, dimension(*) :: xmol,wmass
   double precision totmol,totmass
   TYPE(gtp_equilibrium_data), pointer :: ceq
-------------------
 subroutine sumprops(props,ceq)
! summing up G, S, V, N and B for all phases with positive amount
! Check if this is correct
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision props(5)
```

8.11.6 Encode state variable

This converts the internal coding of a state variable into a character variable text starting at position ip. The value of ip is updated inside.

The subroutine also handle property symbols used in the parameters, see 1.5.2, to make it possible to list the symbol of such a propery after an equilibrium calculation. See 8.11.3

The new version of this subroutine uses state variable records, the previous one using individual arguments should not be used as it will eventually disappear.

```
 subroutine encode_state_variable(text,ip,svr,ceq)
! writes a state variable in text form position ip.  ip is updated
   character text*(*)
   integer ip
   type(gtp_state_variable), pointer :: svr
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine encode_state_variable3(text,ip,istv,indices,iunit,iref,ceq)
```

```
! writes a state variable in text form position ip.  ip is updated
! the internal coding provides in istv, indices, iunit and iref
! ceq is needed as compopnents can be different in different equilibria ??
! >>>> unfinished as iunit and iref not really cared for
   implicit none
   integer, parameter :: noos=20
   character*4, dimension(noos), parameter :: svid = &
       ['T   ','P   ','MU  ','AC  ','LNAC','U   ','S   ','V   ',&
        'H   ','A   ','G   ','NP  ','BP  ','DG  ','Q   ','N   ',&
        'X   ','B   ','W   ','Y   ']
   character*(*) text
   integer, dimension(4) :: indices
   integer istv,ip,iunit,iref
   type(gtp_equilibrium_data), pointer :: ceq
```

8.11.7 Encode a state variable record

This is provided to convert a single state variable record to a text.

```
 subroutine encode_state_variable_record(text,ip,svr,ceq)
! writes a state variable in text form position ip.  ip is updated
! the svr record provide istv, indices, iunit and iref
! ceq is needed as compopnents can be different in different equilibria ??
! >>>> unfinished as iunit and iref not really cared for
   implicit none
   integer, parameter :: noos=20
   character*4, dimension(noos), parameter :: svid = &
       ['T   ','P   ','MU  ','AC  ','LNAC','U   ','S   ','V   ',&
        'H   ','A   ','G   ','NP  ','BP  ','DG  ','Q   ','N   ',&
        'X   ','B   ','W   ','Y   ']
   character*(*) text
   type(gtp_state_variable) :: svr
   type(gtp_equilibrium_data), pointer :: ceq
-----------------
 subroutine findsublattice(iph,constix,sublat)
! find sublattice of constituent constix in phase lokph
! is lokph index to gtp_phaserecord or gtp_phase_varres??
! constix is constituent index
   implicit none
   integer iph,constix,sublat
```

8.11.8 Calculate state variable value

This is the subroutine that actually calculates the value of a state variable. The state variable is identified using the internal coding.

The subroutine also handle properties used in the parameters, see 1.5.2, to make it possible to obtain the value of such a propery after an equilibrium calculation. The values of istv etc must be as returned from decode_state_variable, see 8.11.3.

```
 subroutine state_variable_val(svr,value,ceq)
! calculate the value of a state variable in equilibrium record ceq
! It transforms svr data to old format and calls state_variable_val3
   type(gtp_state_variable), pointer :: svr
   double precision value
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine state_variable_val3(istv,indices,iref,iunit,value,ceq)
! calculate the value of a state variable in equilibrium record ceq
! istv is state variable type (integer)
! indices are possible specifiers
! iref indicates use of possible reference state, 0 current, -1 SER
! iunit is unit, (K, oC, J, cal etc). For % it is 100
! value is the calculated values. for state variables with wildcards use
! get_many_svar
   implicit none
   integer, dimension(4) :: indices
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer istv,iref,iunit
   double precision value
```

8.11.9 Calculate eigenvalues of a phase stability matrix

This subroutine calculates the eigenvalues of the phase stability matrix, i.e. the matrix of all second derivatives with respect to the constitution. It returns the vale that is lowest, i.e negative or closest to zero. If it is negative the phase composition is inside a spinodal. It uses a LINPAC routine to calculate the eigenvalues.

Some unused variants still included in the code.

```
 subroutine calc_qf(lokcs,value,ceq)
! calculates eigenvalues of the second derivative matrix, stability function
! lokcs is index of phase_varres
! value calculated value returned
! ceq is current equilibrium
! For ionic liquid and charged crystalline phases one should
! calculate eigenvectors to find neutral directions.
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer :: lokcs
   double precision value
```

```
--------------------
 subroutine calc_qf_otis(lokcs,value,ceq)
! NOT USED ----------------------------
! calculates eigenvalues of the second derivative matrix, stability function
! using Otis reduced Hessian method.  Should work indpent of model!!
! lokcs is index of phase_varres
! value calculated value returned
! ceq is current equilibrium
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer :: lokcs
   double precision value
----------------
 subroutine calc_qf_sub(lokcs,value,ceq)
! NOT USED ----------------------------
! calculates eigenvalues of the second derivative matrix, stability function
! using the Darken matrix with second derivatives: OK FOR SUBSTITUTIONAL
! lokcs is index of phase_varres
! value calculated value returned
! ceq is current equilibrium
! For ionic liquid and charged crystalline phases one should
! calculate eigenvectors to find neutral directions.
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer :: lokcs
   double precision value
-----------------------
 subroutine calc_qf_old(lokcs,value,ceq)
! NOT USED ----------------------------
! calculates eigenvalues of the second derivative matrix, stability function
! this old version that seems to work for Ag-Cu ...
! lokcs is index of phase_varres
! value calculated value returned
! ceq is current equilibrium
! For ionic liquid and charged crystalline phases one should
! calculate eigenvectors to find neutral directions.
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer :: lokcs
   double precision value
```

8.11.10 The value of the user defined reference state

The value of many state variables depend on the selected reference state. By default that is the value at 298.15 K and 1 bar for the stable phase of the pure elements, this is called SER. If the user defines another reference state this subroutine calculates that value. The reference state is set calling the routine in section 8.5.2.

```
 subroutine calculate_reference_state(kstv,iph,ics,aref,ceq)
! Calculate the user defined reference state for extensive properties
! kstv is the typde of property: 1 U, 2 S, 3 V, 4 H, 5 A, 6 G
! It can be phase specific (iph.ne.0) or global (iph=0)
! IMPORTANT
! For integral quantitites (like calculated here) the reference state
! is ignored unless all components have the same phase as reference (like Hmix)
   implicit none
   integer kstv,iph,ics
   double precision aref
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine calculate_reference_state_old(kstv,iph,ics,aref,ceq)
! Calculate the user defined reference state for extensive properties
! kstv is the typde of property: 1 U, 2 S, 3 V, 4 H, 5 A, 6 G
! It can be phase specific (iph.ne.0) or global (iph=0)
! IMPORTANT
! For integral quantitites (like calculated here) the reference state
! is ignored unless all components have the same phase as reference (like Hmix)
   implicit none
   integer kstv,iph,ics
   double precision aref
   type(gtp_equilibrium_data), pointer :: ceq
```

8.12 State variable functions

This section is separated from the state variables itself to make it a little simpler. State variable function can contain any combination of state variables using normal operators like +, -, *, / but also EXP, LN, LOG10, ERF etc. The PUTFUN subroutine in the METLIB package is used. No derivatives can be calculated. A state variable function can refer to another state variable function. Thus when the value of a state variable function is requested all are calculated with some exceptions: not dot derivaties and not those amended as:

- The function is amended to be evaluated only when specified explicitly. Such a symbol can be used as condition for example in STEP EVALUATE (not yet implemented).
- The fucntion is amended to be evaluated only in a specific equilibrium record. This is useful of one has experiements of heat differences relative to an equilibrium at a certain T. The enthalpy of this equilibrium will be evaluated only at this equilibrium.

An extension planned but not yet implemented is to allow formal arguments when defining a state variable function, for example CP(@P)=HM(@P).T where the formal argument @P means a phase. @S would stand for a species and @C for a component. When calling the function an actual argument must be supplied.

Some state variable functions using "dot derivatives" to calculate for example the heat capacity are calculated inside the minimizer routine as they need results for the last equilibrium.

8.12.1 Enter a state variable function

The first subroutine enters a state variable function and the other stores it in the SVFLISTA array. The old version will eventually disappear.

```
 subroutine enter_svfun(cline,last,ceq)
! enter a state variable function
   implicit none
   integer last
   character cline*(*)
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine set_putfun_constant(svfix,value)
! changes the value of a putfun constant
! svfix is index, value is new value
   implicit none
   integer svfix
   double precision value
---------------
 subroutine store_putfun(name,lrot,nsymb,iarr)
! enter an expression of state variables with name name with address lrot
! nsymb is number of formal arguments
! iarr identifies these
   implicit none
   character name*(*)
   type(putfun_node), pointer :: lrot
   integer nsymb
   integer iarr(10,*)
---------------
 subroutine store_putfun_old(name,lrot,nsymb,&
       istv,indstv,iref,iunit,idot)
! enter an expression of state variables
! name: character, name of state variable function
! lrot: pointer, to a putfun_node that is the root of the stored expression
! nsymb: integer, number of formal arguments
! istv: integer array, formal argument state variables typ
! indstv: 2D integer array, indices for the formal state variables
! iref: integer array, reference for the formal state variables
! iunit: integer array, unit of the formal state variables
   implicit none
   type(putfun_node), pointer :: lrot
   integer nsymb
   integer, dimension(*) :: istv,iref,iunit,idot
   integer, dimension(4,*) :: indstv
   character name*(*)
```

8.12.2 List a state variable function

These two subroutines can find a state variable function and list its name in a character respectively.

```
 subroutine find_svfun(name,lrot)
! finds a state variable function called name (no abbreviations)
! ceq not needed!!??
   implicit none
   character name*(*)
   integer lrot
!   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine find_symbol_with_equilno(lrot,eqno)
! finds a state variable function with equilibrium index
   implicit none
   integer lrot,eqno
---------------
 subroutine list_svfun(text,ipos,lrot,ceq)
! list a state variable function
   implicit none
   character text*(*)
   integer ipos,lrot
   type(gtp_equilibrium_data), pointer :: ceq
```

8.12.3 Utility and list subroutines for state variable functions

Routines to store and list state variable identification for a function.

```
 subroutine make_stvrec(svr,iarr)
! stores appropriate values from a formal argument list to a state variable
! function in a state variable record
   implicit none
   type(gtp_state_variable), pointer :: svr
   integer iarr(10)
 subroutine list_all_svfun(kou,ceq)
! list all state variable funtions
   implicit none
   integer kou
   type(gtp_equilibrium_data), pointer :: ceq
```

8.12.4 Some depreciated routines

The double precision function evaluate_svfun_old(lrot,actual_arg,mode,ceq) is now in the minimizer package but still used somewhere.

```
 subroutine evaluate_all_svfun_old(kou,ceq)
! THIS SUBROUTINE MOVED TO MINIMIZER but kept for initiallizing
! cannot be used for state variable functions that are derivatives ...
! evaluate and list values of all functions but it is still used somewhere
   implicit none
   integer kou
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 double precision function evaluate_svfun_old(lrot,actual_arg,mode,ceq)
! THIS SUBROUTINE MOVED TO MINIMIZER
! but needed in some cases in this module ... ???
! envaluate all funtions as they may depend on each other
! actual_arg are names of phases, components or species as @Pi, @Ci and @Si
! needed in some deferred formal parameters  (NOT IMPLEMENTED YET)
! if mode=1 always evaluate
   implicit none
   integer lrot,mode
   character actual_arg(*)*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.13 Status for things, gtp3G

There are many status bits in various records. These subroutines and function set, clear or test these.

Setting means to set the bit to 1 and clearing means to set the bit to 0. A problem is that I always mix up if BTEST return TRUE or FALSE it the bit is set.

The subroutines and functions in this section should be simplified.

8.13.1 Set and test status for elements

An element can be entered or suspended.

```
 subroutine change_element_status(elname,nystat,ceq)
! change the status of an element, can affect species and phase status
! nystat:0=entered, 1=suspended, -1 special (exclude from sum of mole fraction)
!
! suspending elements for each equilibrium separately not yet implemented
!
```

```
   implicit none
   character elname*(*)
   integer nystat
   TYPE(gtp_equilibrium_data), pointer :: ceq
-----------------
 logical function testelstat(iel,status)
! return value of element status bit
   implicit none
   integer iel,status
```

8.13.2 Set and test status for species

A species can be entered or suspended.

```
 subroutine change_species_status(spname,nystat,ceq)
! change the status of a species, can affect phase status
! nystat:0=entered, 1=suspended
   implicit none
   integer nystat
   character spname*(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
--------------
 logical function testspstat(isp,status)
! return value of species status bit
   implicit none
   integer isp,status
```

8.13.3 Get, test, set and clear status phase model bits

Get and test phase status are almost identical but the get routine returns some additional informantion. Note that there are also phase bits that determine the model used.

```
 integer function get_phase_status(iph,ics,text,ip,val,ceq)
! return phase status as text and amount formula units in val
! for entered and fix phases also phase amounts.
! OLD Function value: 1=entered, 2=fix, 3=dormant, 4=suspended, 5=hidden
   implicit none
   character text*(*)
   integer iph,ics,ip
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision val
---------------
 integer function test_phase_status(iph,ics,val,ceq)
! Almost same as get_..., returns phase status as function value but no text
! old: 1=entered, 2=fix, 3=dormant, 4=suspended, 5=hidden
! nystat:-4 hidden, -3 suspended, -2 dormant, -1,0,1 entered, 2 fix
! this is different from in change_phase .... one has to make up one's mind
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer iph,ics
   double precision val
-------------
 subroutine set_phase_status_bit(lokph,bit)
! set the status bit "bit" in status1, cannot be done outside this module
! as the phlista is private
! These bits do not depend on the composition set
   implicit none
   integer lokph,bit
---------------
 subroutine clear_phase_status_bit(lokph,bit)
! clear the status bit "bit" in status1, cannot be done outside this module
! as the phlista is private
! DOES NOT CHANGE STATUS like entered/fixed/dormant/suspended
   implicit none
   integer lokph,bit
---------------
 logical function test_phase_status_bit(iph,ibit)
! return TRUE is status bit ibit for  phase iph, is set
! because phlista is private.  Needed to test for gas, ideal etc,
! DOES NOT TEST STATUS like entered/fixed/dormant/suspended
   implicit none
   integer iph,ibit
```

8.13.4 Test and change FIX/ENTERED/DORMANT/SUSPENDED status for phase

Several subroutines to change the status bits for one or more phases. Note NYSTAT values are different from this in GET and TEST above. Same values should be used.

The command set status <phase> =fix < amount > is a condition for a calculation and reduces the degrees of freedom, see section 8.9.6.

```
 subroutine change_many_phase_status(phnames,nystat,val,ceq)
! change the status of many phases.
! nystat:-4 hidden, -3 suspended, -2 dormant, -1,0,1 entered, 2 fix
! phnames is a list of phase names or *S (all suspeded) *D (all dormant) or
! *E (all entered (stable, unknown, unstable), *U all unstable
! If just * then change_phase_status is called directly
! It calls change_phase_status for each phase
   implicit none
   character phnames*(*)
   integer nystat
   double precision val
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine change_phtup_status(phtupx,nystat,val,ceq)
! change the status of a phase tuple. Also used when setting phase fix etc.
! nystat:-4 hidden, -3 suspended, -2 dormant, -1,0,1 entered, 2 fix
! qph can be -1 meaning all or a specifix phase index. ics compset
!
   implicit none
   integer phtupx,nystat
   double precision val
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine change_phase_status(qph,ics,nystat,val,ceq)
! change the status of a phase. Also used when setting phase fix etc.
! nystat:-4 hidden, -3 suspended, -2 dormant, -1,0,1 entered, 2 fix
! qph can be -1 meaning all or a specifix phase index. ics compset
!
   implicit none
   integer qph,ics,nystat
   double precision val
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine mark_stable_phase(iph,ics,ceq)
! change the status of a phase. Does not change fix status
! called from meq_sameset to indicate stable phases (nystat=1)
! nystat:-4 hidden, -3 suspended, -2 dormant, -1,0,1 entered, 2 fix
!
   implicit none
   integer iph,ics
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.13.5 Set unit for energy etc.

This is not yet implemented.

```
 subroutine set_unit(property,unit)
! set the unit for a property, like K, F or C for temperature
! >>>> unfinished
   implicit none
   character*(*) property,unit
```

8.13.6 Set reference state for component

This is not yet implemented. It can only refer to components, a constituent does not have a reference state. At present the elements are the components.

```
 subroutine set_constituent_reference_state(iph,icon,asum)
! determine the ENDMEMBER to calculate as reference state for this component!!
! Used when giving a chemical potential for a component like MU(GAS,H2O)
   implicit none
   integer iph,icon
   double precision asum
```

8.13.7 Calculate conversion matrix for new components

This will be needed when it is possible to define other components than the elements.

```
 subroutine elements2components1(nspel,dum,ncmp,cmpstoi,ceq)
! converts a stoichiometry array for a species from elements to components
! This subroutine, is it used to get activity for a constituent in gtp3F
! dum is is no longer used
   implicit none
   integer nspel,ncmp
! cmpstoi is stoichiometry as element, changed to be as components
   double precision cmpstoi(*),dum(*)
   double precision, allocatable :: stoi(:)
   type(gtp_equilibrium_data), pointer :: ceq
```

8.14 Internal stuff

The subroutines in this section are internal and not to be used by calls from outside GTP.

8.14.1 Indicate the type of an experiment

An experiment can specify an equal value, =, a larger than limit, > or a lesser than limit, <. This return the type.

```
 subroutine termterm(string,ich,kpos,lpos,value)
! search for first occurance of + - = > or <
! if + or - then also extract possible value after sign
! value is coefficient for NEXT term (if any)
! kpos is last character in THIS state variable, lpos where NEXT may start
   implicit none
   character string*(*)
   integer kpos,ich,lpos
   double precision value
```

8.14.2 Alphabetical ordering

The elements, species and phases are stored in the arrays ELLISTA, SPLISTA and PHLISTA in the order they are entered. They are sorted in alphabetical order in the array ELEMENTS, SPECIES and PHASES. These routines are used to maintain the alphabetical order.

```
 subroutine alphaelorder
! arrange new element in alphabetical order
! also make alphaindex give alphabetical order
   implicit none
---------------
 subroutine alphasporder
! arrange new species in alphabetical order
! also make alphaindex give alphabetical order
   implicit none
---------------
 subroutine alphaphorder(tuple)
! arrange last added phase in alphabetical order
! also make alphaindex give alphabetical order
! phletter G and L and I have priority
! tuple is returned as position in phase tuple
   implicit none
   integer tuple
----------------
 subroutine check_alphaindex
! just for debugging, check that ellist(i)%alphaindex etc is  correct
   implicit none
```

8.14.3 Creates different lists of constituents of a phase

Not much to discuss.

```
 subroutine create_constitlist(constitlist,nc,klist)
! creates a constituent list ...
   implicit none
   integer, dimension(*) :: klist
   integer, dimension(:), allocatable :: constitlist
   integer nc
--------------------
 subroutine create_parrecords(lokph,lokcs,nsl,nc,nprop,iva,ceq)
! fractions and results arrays for a phase for parallel calculations
! location is returned in lokcs
! nsl is sublattices, nc number of constituents, nprop max number if propert,
! iva is an array which is set as constituent status word (to indicate VA)
! ceq is always firsteq ???
```

```
!
! BEWARE not adopted for threads
!
! >>> changed all firsteq below to ceq????
!
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer, dimension(*) :: iva
   integer lokph,lokcs, nsl, nc, nprop
```

8.14.4 Create endmember and interaction parameter records

Finds the correct place to add an interaction parameter. It should always be linked from the first possible endmember (with the alphabetically first constituents) and then in alphabetical order of the interaction elements.

The second routine creates a record for an endmember. Sometimes endmembers have no property records but there can be interaction parameters which must be linked from a missing endmember.

```
 subroutine create_interaction(intrec,mint,lint,intperm,intlinks)
! creates a parameter interaction record
! with permutations if intperm(1)>0
   implicit none
   type(gtp_interaction), pointer :: intrec
   integer, dimension(2,*) :: lint,intlinks
   integer, dimension(*) :: intperm
   integer mint
-----------------
  subroutine create_endmember(lokph,newem,noperm,nsl,endm,elinks)
! create endmember record with nsl sublattices with endm as constituents
! noperm is number of permutations
! endm is the basic endmember (if there are permutations)
! elinks are the links to constituents for all permutations
   implicit none
   integer endm(*)
   integer lokph,noperm,nsl
   type(gtp_endmember), pointer :: newem
   integer, dimension(nsl,noperm) :: elinks
```

8.14.5 Create and extend a property record

All parameter values for an endmember or interaction record are stored in property records. An endmember or interaction parameter may have several property records linked in a list.

An interaction record can have a degree and each degree a function. When entering a function for a higher degree than in the current property record it must be extended.

```
 subroutine create_proprec(proprec,proptype,degree,lfun,refx)
! reservs a property record from free list and insert data
   implicit none
   TYPE(gtp_property), pointer :: proprec
   integer proptype,degree,lfun
   character refx*(*)
--------------------
 subroutine extend_proprec(current,degree,lfun)
! extends a property record  and insert new data
   implicit none
   integer degree,lfun
   type(gtp_property), pointer :: current
```

8.14.6 Add a fraction set record

```
 subroutine add_fraction_set(iph,id,ndl,totdis)
! add a new set of fractions to a phase, usually to describe a disordered state
! like the "partitioning" in old TC
!
! BEWARE this is only done for firsteq, illegal when having more equilibria
!
! id is a letter used as suffix to identify the parameters of this set
! ndl is the last original sublattice included in the (first) disordered set
! ndl can be 1 meaning sublattice 2..nsl are disordered, or nsl meaning all are
!     disordered
! totdis=0 if phase never disorder totally (like sigma)
!
! For a phase like (Al,Fe,Ni)3(Al,Fe,Ni)1(C,Va)4 to add (Al,Fe,Ni)4(C,Va)4
! icon=1 2 3 1 2 3 4 5 with ndl=2
! For a phase like (Fe,Ni)10(Cr,Mo)4(Cr,Fe,Mo,Ni)16 then
! icon=2 4 1 3 1 2 3 4 with ndl=3
! This subroutine will create the necessary data to calculate the
! disordered fraction set from the site fractions.
!
! IMPORTANT (done): for each composition set this must be repeated
! if new composition sets are created it must be repeated for these
!
! IMPORTANT (not done): order the constituents alphabetically in each disorderd
! sublattice otherwise it will not be possible to enter parameters correctly
!
   implicit none
   integer iph,ndl,totdis
   character id*1
```

8.14.7 Copy record for fraction sets

The first routine can be useful to copy fractions from one equilibrium to another during assessments when there is a miscibility gap in a phase.

```
  subroutine copyfracs(fromeq,ceq)
! Copy phase amounts and constitution from equilibrim fromceq to ceq
! Useful to set start constitutions for miscibility gaps during assessments
!
    implicit none
    integer fromeq
    type(gtp_equilibrium_data), pointer :: ceq
----------------------
 subroutine copy_fracset_record(lokcs,disrec,ceq)
! attempt to create a new disordered record  ??? this can probably be done
! with just one statement .. but as it works I am not changing right now
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_fraction_set) :: disrec
   integer lokcs
```

8.14.8 Implicit suspend and restore

If an element is suspended some species may have to be suspended too and if a species is suspended some phases may have to be suspended. This routine should doe that but I do not think it has been tested.

```
 subroutine suspend_species_implicitly(ceq)
! loop through all entered species and suspend those with an element suspended
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine suspend_phases_implicitly(ceq)
! loop through all entered phases and suspend constituents and
! SUSPEND phases with all constituents in a sublattice suspended
!   dimension lokcs(9)
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine restore_species_implicitly_suspended
! loop through all implicitly suspended species and restore those with
! all elements enteded
   implicit none
---------------
 subroutine restore_phases_implicitly_suspended
```

```
! loop through all implicitly suspended phases and restore those with
! at least one constituent entered in each sublattice
   implicit none
```

8.14.9 Add to reference phase

There is a reference phase that should have parameters for each element it its stable state at all temperatures and 1 bar. For each element entered this subroutine adds it to the reference phase. This phase can never be used in calculations. It represent different phases for each element, gas for H, bcc for Cr etc.

A problem with this reference phase is that different magnetic models may be needed for different element. For example Fe has BCC magnetic model and Ni has FCC magnetic model in their reference states.

This reference phase is not used at all at present.

```
 subroutine add_to_reference_phase(loksp)
! add this element to the reference phase
! loksp: species index of new element
   implicit none
   integer loksp
```

8.15 Additions, gtp3H

Creating, handling and calculations of additions to the Gibbs energy. This is a section that will probably be extended with several new subroutines for different kinds of additions. See also sections 1.5, 3.2.4 and 1.5.2,

Note that the Equi-entropy criterion (EEC) is checked in the minimizer as it relates to two different phases (a solid and the liquid). It is not really an addition as it does not depend on property of a single phase.

```
 subroutine addition_selector(addrec,moded,phres,lokph,mc,ceq)
! called when finding an addition record while calculating G for a phase
! addrec is addition record
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! phres is ?
! lokph is phase location
! mc is number of constitution fractions
! ceq is current equilibrium record
   implicit none
   type(gtp_phase_add), pointer :: addrec
   integer moded,lokph,mc
   TYPE(gtp_phase_varres), pointer :: phres
   type(gtp_equilibrium_data), pointer :: ceq
```

8.15.1 Generic subroutine to add an addition

This adds an addition such as magnetic, low-T vibrational (Einstein) contribution etc. This package is unfinished and quite messy.

```
 subroutine add_addrecord(lokph,extra,addtyp)
! generic subroutine to add an addition typ addtyp (including Inden)
   implicit none
   integer lokph,addtyp
   character extra*(*)
```

8.15.2 Utility routines for addition

Searches for the composition dependent properties for a specific addition. Most additions require some specific model parameter such as Curie-T, Einstein T etc to be calculated. Such values (and their derivatives) are calculated together with the Gibbs energy parameters and are needed to calculate the contribution from the addition.

The subroutine “setpermolebit” indicates that the property is calculated per mole of atoms, not per formula unit as default, see section 8.15.8. The property must thus be multiplied with the number of moles of atoms per formula unit (and derivatives thereof) before added to the Gibbs energy (and derivatives thereof).

```
 subroutine need_propertyid(id,typty)
! get the index of the property needed
   implicit none
   integer typty
   character*4 id
---------------
 subroutine setpermolebit(lokph,addtype)
! set bit in addition record that addition is per mole
! lokph is phase record
! addtype is the addtion record type
   implicit none
   integer lokph,addtype
```

8.15.3 Enter and calculate Inden magnetic model

The first two subroutines create the ferromagnetic addition due to Inden model and store all necessary data inside this. The last subroutine is called when calculating the Gibbs energy for a phase if there is a magnetic addition linked to the phase. It must calculate the contribution to G and all first and second derivatives of G. The model parameter identifiers TC and BMAG defined in Table 4 are used.

In the call the pointer to the phase_varres record is provided where current values of G and derivatives can be found. Values of the ferromagnetic temperature and its derivatives with respect to constitution is also stored there. The chain rule for derivatives must be applied.

```
 subroutine create_magrec_inden(addrec,aff)
! enters the magnetic model
   implicit none
   type(gtp_phase_add), pointer :: addrec
   integer aff
---------------
 subroutine calc_magnetic_inden(moded,phres,lokadd,lokph,mc,ceq)
! calculates Indens magnetic contribution
! NOTE: values for function not saved, should be done to save time.
! Gmagn = RT*f(T/Tc)*ln(beta+1)
! moded: integer, 0=only G, S, Cp; 1=G and dG/dy; 2=Gm dG/dy and d2G/dy2
! phres: pointer, to phase\_varres record
! lokadd: pointer, to addition record
! lokph: integer, phase record
! mc: integer, number of constituents
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer moded,lokph,mc
   TYPE(gtp_phase_varres) :: phres
   TYPE(gtp_phase_add), pointer :: lokadd
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.15.4 Enter and calculate Qing/Wei magnetic model

Qing Chen et al. [20] and Wei Xiong at al. [18] have proposed a more advanced magnetic model. It is not yet fully implemented. It should use several model parameter identified in Table 4

```
 subroutine create_xiongmagnetic(addrec,lokph,bcc)
! adds a Xiong type magnetic record, we must separate fcc and bcc by extra
! copied from Inden magnetic model
! The difference is that it uses TCA for Curie temperature and TNA for Neel
! and individual Bohr magneton numbers
   implicit none
   logical bcc
   integer lokph
   type(gtp_phase_add), pointer :: addrec
---------------
 subroutine calc_xiongmagnetic(moded,phres,lokadd,lokph,mc,ceq)
! calculates Indens-Qing-Xiong magnetic contribution
!
! Gmagn = RT*f(T/Tc)*ln(beta+1)
! moded: integer, 0=only G, S, Cp; 1=G and dG/dy; 2=Gm dG/dy and d2G/dy2
! phres: pointer, to phase\_varres record
! lokadd: pointer, to addition record
! lokph: integer, phase record
! mc: integer, number of constituents
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer moded,lokph,mc
! phres points to result record with gval etc for this phase
   TYPE(gtp_phase_varres) :: phres
   TYPE(gtp_phase_add), pointer :: lokadd
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.15.5 Enter and calculate volume model

A very simple volume model with two model parameters, V0 which is the volume at 298.15 K and 1 bar for the endmember, and VA which is the thermal expansion of the endmember defined in Table 4.

```
 subroutine create_volmod1(addrec)
! create addition record for the simple volume model
!
! currently only V = V0 * exp(VA(T))
! V0 is property (typty) 21, VA is 22 and reserved VB (Bulk modulus) 23
! but VB is not implemented yet
   implicit none
   type(gtp_phase_add), pointer :: addrec
---------------
 subroutine calc_volmod1(moded,phres,lokadd,lokph,mc,ceq)
! calculate the simple volume model, CURRENTLY IGNORING COMPOSITION DEPENDENCE
!
! G = P*V0(x)*exp(VA(T,x))
! moded: integer, 0=only G, S, Cp; 1=G and dG/dy; 2=Gm dG/dy and d2G/dy2
! phres: pointer, to phase\_varres record
! lokadd: pointer, to addition record
! lokph: integer, phase record index
! mc: integer, number of constituents
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer moded,lokph,mc
! phres points to result record with gval etc for this phase
   TYPE(gtp_phase_varres) :: phres
   TYPE(gtp_phase_add), pointer :: lokadd
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.15.6 Enter and calculate elastic contribution

This is not implemented. It should create and calculate the elastic record. It is not implemented but there are several model parameters such as LPX, LPY, LPZ LPTH, EC11, EC12, EC44 previewed in Table 4.

```
 subroutine create_elastic_model_a(newadd)
! addition record to calculate the elastic energy contribution
   implicit none
   type(gtp_phase_add), pointer :: newadd
---------------
 subroutine calc_elastica(moded,phres,addrec,lokph,mc,ceq)
! calculates elastic contribution and adds to G and derivatives
   implicit none
   integer moded,lokph,mc
   type(gtp_phase_varres), pointer :: phres
   type(gtp_phase_add), pointer :: addrec
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine set_lattice_parameters(iph,ics,xxx,ceq)
! temporary way to set current lattice parameters for use with elastic model a
   implicit none
   integer iph,ics
   double precision, dimension(3,3) :: xxx
   type(gtp_equilibrium_data), pointer :: ceq
```

8.15.7 Enter and calculate for Einstein solids

This is now implemented. The parameter identifier LNTH in Table 4 is the logarithm of the Einstein T as that should be the reasonable property varying with the composition. See section 2.3.5.1.

```
 subroutine create_einsteincp(newadd)
   implicit none
   type(gtp_phase_add), pointer :: newadd
---------------
 subroutine calc_einsteincp(moded,phres,addrec,lokph,mc,ceq)
! Calculate the contibution due to Einste Cp model for low T
! moded 0, 1 or 2
! phres all results
! addrec pointer to addition record
! lokph phase record
! mc number of variable fractions
! ceq equilibrum record
!
! G = 1.5*R*THET + 3*R*T*ln( 1 - exp( -THET/T ) )
! This is easier to handle inside the calc routine without TPFUN
!
   implicit none
   integer moded,lokph,mc
   type(gtp_phase_varres), pointer :: phres
   type(gtp_phase_add), pointer :: addrec
   type(gtp_equilibrium_data), pointer :: ceq
```

8.15.8 Add conversion per mole to per formula unit

As mentioned earlier many physical models are defined per mole of atoms whereas OC calculates properties per mole of formula units. This the properties must be multiplied with number of moles per formula unit before added to the Gibbs energy (and its derivatives).

```
 subroutine add_size_derivatives(moded,phres,addphm,lokph,mc,ceq)
! Many physical models are defined per mole of atoms, as the Gibbs energy
! is calculate per mole formula unit this routine will handle the
! additional derivatives needed when M*ADD(1 mole)
! mc is number of constituent variables
! addphm(1..6) is G, dG/dt, dG/dp, d2G/dt2, d2G/dtdp, d2G/dp2 for the addition
   implicit none
   integer moded,lokph,mc
   type(gtp_phase_varres), pointer :: phres
!   type(gtp_phase_add), pointer :: addrec
   type(gtp_equilibrium_data), pointer :: ceq
   double precision addphm(6)
```

8.15.9 Enter and calculate the Schottky anomality

This can describe a Schottky anomality i.e. an increase of the heat capacity over a range in T followe by a decrease. It depends on the model parameter identifiers TSCH and CSCH, defined in Table 4.

```
 subroutine create_schottky_anomality(newadd)
! Adding a Schottky anomality to Cp
   implicit none
   type(gtp_phase_add), pointer :: newadd
---------------
 subroutine calc_schottky_anomality(moded,phres,addrec,lokph,mc,ceq)
! Calculate the contibution due to a Schottky anomality
! moded 0, 1 or 2
! phres all results
! addrec pointer to addition record
! lokph phase record
! mc number of variable fractions
! ceq equilibrum record
!
! G = DCP2*T*ln( 1 + exp( -THT2/T ) )
! dG/dT = DCP2*(ln(1+exp(-THT2/T))+(THT/T)*(1+exp(+THT2/T))**(-1)
! d2G/dT2 = -DCP2*THT2**2*T**(-3)*exp(THT2/T)*(1+exp(+THT2/T))**(-2)
!
   implicit none
   integer moded,lokph,mc
   type(gtp_phase_varres), pointer :: phres
   type(gtp_phase_add), pointer :: addrec
   type(gtp_equilibrium_data), pointer :: ceq
```

8.15.10 Enter and calculate a second Einstein contribution

This can describe an smooth increase of the heat capacity to new fix value over a range in T. It depends on the model parameter identifiers THT2 and DCP2, defined in Table 4.

```
 subroutine create_secondeinstein(newadd)
   implicit none
   type(gtp_phase_add), pointer :: newadd
---------------
 subroutine calc_secondeinstein(moded,phres,addrec,lokph,mc,ceq)
! Calculate the contibution due to Einste Cp model for low T
! moded 0, 1 or 2
! phres all results
! addrec pointer to addition record
! lokph phase record
! mc number of variable fractions
! ceq equilibrum record
!
! G = 1.5*R*THET + 3*R*T*ln( 1 - exp( -THET/T ) )
! This is easier to handle inside the calc routine without TPFUN
!
   implicit none
   integer moded,lokph,mc
   type(gtp_phase_varres), pointer :: phres
   type(gtp_phase_add), pointer :: addrec
   type(gtp_equilibrium_data), pointer :: ceq
```

8.15.11 Enter and calculate the two-state model for amorphous liquids

Now it is implemented but several varians are available. The low T liquid is treated as an Einstein solid with an Einstein θ and an possible T dependent terms in the low T range. An additional model parameter, G2, defined in Table 4 will create an increase in the heat capacity before melting.

```
 subroutine create_twostate_model1(addrec)
! newadd is location where pointer to new addition record should be stored
   implicit none
   type(gtp_phase_add), pointer :: addrec
---------------
 subroutine create_newtwostate_model1(addrec)
! newadd is location where pointer to new addition record should be stored
   implicit none
   type(gtp_phase_add), pointer :: addrec
-----------------
 subroutine calc_twostate_model_john(moded,phres,addrec,lokph,mc,ceq)
! subroutine calc_twostate_model1(moded,phres,addrec,lokph,mc,ceq)
! This routine works OK but I am testing a modification
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! addrec is addition record
! phres is phase_varres record
! lokph is phase location
! mc is number of constitution fractions
! ceq is current equilibrium record
   implicit none
   integer moded,lokph,mc
   TYPE(GTP_PHASE_ADD), pointer :: addrec
   TYPE(GTP_PHASE_VARRES), pointer :: phres
   TYPE(GTP_EQUILIBRIUM_DATA), pointer :: ceq
---------------
 subroutine calc_twostate_model1(moded,phres,addrec,lokph,mc,ceq)
! subroutine calc_twostate_modelny(moded,phres,addrec,lokph,mc,ceq)
! The routine _john works OK but I am testing a modification
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! addrec is addition record
! phres is phase_varres record
! lokph is phase location
! mc is number of constitution fractions
! ceq is current equilibrium record
   implicit none
   integer moded,lokph,mc
   TYPE(GTP_PHASE_ADD), pointer :: addrec
   TYPE(GTP_PHASE_VARRES), pointer :: phres
   TYPE(GTP_EQUILIBRIUM_DATA), pointer :: ceq
---------------
 subroutine calc_twostate_model2(moded,phres,addrec,lokph,mc,ceq)
! subroutine calc_twostate_model_nomix(moded,phres,addrec,lokph,mc,ceq)
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! addrec is addition record
!
! IN THIS VERSION G2 is treated as a composition independent parameter
!  thus this just handles the Einsten Cp
! NOTE Einstein LNTH should be composition dependent
!
! phres is phase_varres record
! lokph is phase location
! mc is number of constitution fractions
! ceq is current equilibrium record
   implicit none
   integer moded,lokph,mc
   TYPE(GTP_PHASE_ADD), pointer :: addrec
   TYPE(GTP_PHASE_VARRES), pointer :: phres
   TYPE(GTP_EQUILIBRIUM_DATA), pointer :: ceq
-------------------
 subroutine calc_twostate_model_endmember(proprec,g2values,ceq)
! This calculated G2 (GD in some papers) for a pure endmember
! No composition dependence
! Value calculated here added to ^oG for the endmember
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! phres is phase_varres record
! lokph is phase location
! ceq is current equilibrium record
   implicit none
   TYPE(gtp_property), pointer :: proprec
   TYPE(GTP_EQUILIBRIUM_DATA), pointer :: ceq
   double precision g2values(6)
----------------------
 subroutine calc_twostate_model_old(moded,phres,addrec,lokph,mc,ceq)
! Failed attempt to decrease the hump when the g2 parameter changes sign
! moded is 0, 1 or 2 if no, first or 2nd order derivatives should be calculated
! addrec is addition record
! phres is phase_varres record
! lokph is phase location
! mc is number of constitution fractions
! ceq is current equilibrium record
   implicit none
   integer moded,lokph,mc
   TYPE(GTP_PHASE_ADD), pointer :: addrec
   TYPE(GTP_PHASE_VARRES), pointer :: phres
   TYPE(GTP_EQUILIBRIUM_DATA), pointer :: ceq
```

8.15.12 Debye heat capacity model not implemented

The Debye model has been abandonned because the Einstein model is much simpler both Einsten and Debye models need additional polynomial in T terms to fit experimental data.

```
 subroutine create_debyecp(addrec)
! enters a record for the debye model
   implicit none
   type(gtp_phase_add), pointer :: addrec
---------------
 subroutine calc_debyecp(moded,phres,lokadd,lokph,mc,ceq)
! calculates Mauro Debye contribution
! NOTE: values for function not saved, should be done to save calculation time.
! moded: integer, 0=only G, S, Cp; 1=G and dG/dy; 2=Gm dG/dy and d2G/dy2
! phres: pointer, to phase\_varres record
! lokadd: pointer, to addition record
! lokph: integer, phase record
! mc: integer, number of constituents
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer moded,lokph,mc
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_phase_add), pointer :: lokadd
   TYPE(gtp_phase_varres) :: phres
```

8.15.13 Entering and calculating diffusion data

Diffusion coefficients are needed when there is no information on the chemical potentials of the diffusing elements. In a simulation when thermodynamic data is available there is, in my opinion, no need for diffusion coefficients.

This is inteded to convert mobility data to diffusion coefficients. using the second derivatives of the Gibbs energy. It is unfinished.

```
 subroutine create_diffusion(addrec,lokph,text)
   implicit none
   integer lokph
   character text*(*)
   type(gtp_phase_add), pointer :: addrec
---------------
 subroutine diffusion_onoff(phasetup,bitval)
! switches the bit which calculates diffusion coefficients on/off
! if bitval 0 calculate is turned on, 1 turn off
   implicit none
   integer bitval
   type(gtp_phasetuple) :: phasetup
---------------
 subroutine calc_diffusion(moded,phres,lokadd,lokph,mc,ceq)
! calculates diffusion coefficients
! NOTE: values for function not saved, should be done to save calculation time.
! moded: integer, 0=only G, S, Cp; 1=G and dG/dy; 2=Gm dG/dy and d2G/dy2
! phres: pointer, to phase\_varres record
! lokadd: pointer, to addition record
! lokph: integer, phase record
! mc: integer, number of constituents
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer moded,lokph,mc
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_phase_add), pointer :: lokadd
   TYPE(gtp_phase_varres) :: phres
---------------
 subroutine get_diffusion_matrix(phtup,mdm,dcval,ceq)
! extracts calculated diffusion coefficients for a phase tuple
! phtup phase tuple
! dcval diffusion matrix
! ceq: pointer, to gtp_equilibrium_data
   implicit none
   integer mdm
   double precision dcval(mdm,*)
   TYPE(gtp_phasetuple) :: phtup
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.15.14 List additions

When listing data for a phase with addition the relevant information must be listed also for additions. The model parameters are automatically included.

```
 subroutine list_addition(unit,CHTD,phname,ftyp,lokadd)
! list description of an addition for a phase on unit
! used when writing databases files and phase data
   implicit none
   integer unit,ftyp
! CHTD is letter for TDB files TYPE_DEFINITION ... suck
   character CHTD*1,phname*(*)
   TYPE(gtp_phase_add), pointer :: lokadd
---------------
 subroutine list_addition_values(unit,phres)
! lists calculated values for this addition
! Used for the command CALCULATE PHASE to inform about additions
   implicit none
   integer unit
   TYPE(gtp_phase_varres), pointer :: phres
```

8.15.15 Ternary extrapolations other than Muggianu

In some cases the Toop or Kohler ternary extrapolation models are used. This is an unfinished attempt to include this in OC. See also section D.

```
 subroutine add_ternary_extrapol_method(kou,lokph,typ,toophead,species)
! add a Toop or Kohler extrapolation method for a ternary subsystem to a phase
! interactive or from database
! if kou>0 then the command is given interactivly and questions can be asked
   implicit none
   integer kou,lokph
   character typ*1,species(3)*24
! VERY DIFFICULT TO FIX: I must use an external sequental list for all tooprec
! and allocate the "seq" pointer
! oterwise the allocated tooprecord were destroyed in an arbitrary way ..
   type(gtp_tooprec) :: toophead
```

8.16 Calculation, gtp3X

There are many subroutines involved in calculating the Gibbs energy and related properties for a phase and to retrieve values afterwards. Some are explained in connection with the property they calculate, for example the magnetic contribution in 8.15.3.

The minimizer calculates the properties of all entered phases and tries to find the equilibrium as the lowest common tangent plane for the chemical potential calculated in all phases. It varies the constitution of all phases and the amount of the stable phases and can change the set of stable phases if the amount becomes negative (remove) or its driving force positive (add). The normal external conditions are T, P and the amount of all elements but by using Legendre multipliers one can use other conditions. The minimizer also ensures that all phases with ions are electrically neutral.

8.16.1 Calculate all properties for one phase

This subroutine calculates the Gibbs energy and all first and second derivatives with respect to T, P and constituents for the specified phase and composition set using the current values of T, P and constitution of the phase (set by set_constitution, see 8.5.1). It also calculates all other properties stored in the property records.

It is possible to calculate only G my setting moded=0, only G and first derivatives if moded=1 and also second derivatives with moded=2. This routine calls calcg_internal to do the calculations after some checks.

```
 subroutine calcg(iph,ics,moded,lokres,ceq)
! calculates G for phase iph and composition set ics in equilibrium ceq
! checks first that phase and composition set exists
! Data taken and stored in equilibrium record ceq
! lokres is set to the phase_varres record with all fractions and results
! moded is 0, 1 or 2 depending on calculating no, first or 2nd derivarives
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer iph,ics,moded,lokres
```

8.16.2 Model independent routine for one phase calculation

This is the central subroutine to calculate G and derivatives for all kinds of phases. At present only the CEF and ionic liquid model is implemented. It calls many other calculating subroutines, some are described in connection with the property they calculate, like magnetic contribution.

```
 subroutine calcg_internal(lokph,moded,cps,ceq)
! Central calculating routine calculating G and everyting else for a phase
! ceq is the equilibrium record, cps is the phase_varres record for lokph
! moded is type of calculation, 0=only G, 1 G and first derivatives
!    2=G and all second derivatives
! Can also handle the ionic liquid model now ....
   implicit none
   integer lokph,moded
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_phase_varres), target :: cps
```

8.16.3 Calculate an interaction parameter

The interaction parameters (of all properties) can have different composition dependence and this is calculated by calcg_internal. The calculated values and its derivatives are added to the appropriate property arrays.

```
 subroutine cgint(lokph,lokpty,moded,vals,dvals,d2vals,gz,ceq)
! calculates an excess parameter that can be composition dependent
! gz%yfrem are the site fractions in the end member record
! gz%yfrint are the site fractions in the interaction record(s)
! lokpty is the property index, lokph is the phase record
! moded=0 means only G, =1 G and dG/dy, =2 all
   implicit none
   integer moded,lokph
   TYPE(gtp_property), pointer :: lokpty
   TYPE(gtp_parcalc) :: gz
   double precision vals(6),dvals(3,gz%nofc)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.16.4 Calculate ideal configurational entropy

This calculates the ideal configurational entropy summed over all sublattices.

```
 subroutine config_entropy(moded,nsl,nkl,phvar,tval)
! calculates configurational entropy/R for phase lokph
   implicit none
   integer moded,nsl
   integer, dimension(nsl) :: nkl
   TYPE(gtp_phase_varres), pointer :: phvar
```

8.16.5 Calculate the configurational entropy for other models

There are several non-ideal configurational entropy subroutines, several of them are just tentative. The ionic liquid model assumes ideal mixing on each sublattice but the site ratios are not constant.

```
 subroutine config_entropy_ssro(moded,lokph,phvar,tval)
! test calculates SSRO configurational entropy/R for phase lokph
   implicit none
   integer moded,lokph
!   integer, dimension(nsl) :: nkl
   TYPE(gtp_phase_varres), pointer :: phvar
-----------------------
 subroutine config_entropy_i2sl(moded,nsl,nkl,phvar,i2slx,tval)
! calculates configurational entropy/R for ionic liquid model
! Always 2 sublattices, the sites depend on composition
! P = \sum_j (-v_j) y_j + Q y_Va
! Q = \sum_i v_i y_i
! where v is the charge on the ions. P and Q calculated by set_constitution
   implicit none
   integer moded,nsl,i2slx(2)
   integer, dimension(nsl) :: nkl
   TYPE(gtp_phase_varres), pointer :: phvar
 ----------------------
 subroutine config_entropy_qcwithlro(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for the quasichemial liquid with LRO
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
------------------------
 subroutine config_entropy_cqc_classicqc(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for the quasichemial liquid with LRO
!
! THIS ROUTINE NOT USED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! THIS WORKS OK 2019-01-10: DO NOT CHANGE ANYTHING!! works for qcmodel=1
! routine for qcmodel=2 and 3 laret
!
! only question is the parameter, value of G = K*T*R/2;
! K=-10, T=600 means G= -10*600*R/2 = -3000*R gives same curves as in paper.
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
----------------------------
 subroutine config_entropy_qchillert(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for the corrected quasichemial liquid
! Hillert-Selleby-Sundman
! Rewritten 2019-01-12 based on cqc-classicqc which seems correct
! test1: qcmodel=1: OK for zhalf=1 and 3
! test2: qcmodel=2: OK!!
! test3: qcmodel=3: OK for SRO, problmens for miscibility gap
!
! only question is the parameter, value of G = K*T*R/2;
! K=-10, T=600 means G= -10*600*R/2 = -3000*R gives same curves as in paper.
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
------------------------
 subroutine config_entropy_cvmce(moded,ncon,phvar,phrec,tval)
!
! calculates the classical QC and CVM models sith LRO
! started 2021-02-17
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
 ======================
 subroutine config_entropy_tisr(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for the Kremer liquid SRO model
! started 2021-02-12
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents, each cell a constituent
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
! to obtain current fractions and store results
   TYPE(gtp_phase_varres), pointer :: phvar
! to obtain phase and constituent inforamation
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
 ======================
 subroutine config_entropy_srot(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for tetrahedron SRO model
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents, each cell a constituent
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
 ======================
 subroutine config_entropy_mqmqa(moded,ncon,phvar,phrec,tval)
!
! calculates configurational entropy/R for the MQMQA liquid
! started 2021-01-11
! Finally understood the model 2021-10-11
!
! moded=0 only G, =1 G and dG/dy, =2 G, dG/dy and d2G/dy1/dy2
! ncon is number of constituents
! phvar is pointer to phase_varres record
! phrec is the phase record
! tval is current value of T
   implicit none
   integer moded,ncon
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_phaserecord) :: phrec
   double precision tval
```

8.16.6 Push/pop constituent fraction product on stack

These subroutines are used to push/pop current values of the product of constituent fractions and its derivatives before calculating an interaction parameter.

```
 subroutine push_pyval(pystack,intrec,pmq,pyq,dpyq,d2pyq,moded,iz)
! push data when entering an interaction record
   implicit none
   integer pmq,moded,iz
   double precision pyq,dpyq(iz),d2pyq(iz*(iz+1)/2)
   type(gtp_pystack), pointer :: pystack
   type(gtp_interaction), pointer :: intrec
---------------
 subroutine pop_pyval(pystack,intrec,pmq,pyq,dpyq,d2pyq,moded,iz)
! pop data when entering an interaction record
   implicit none
   integer iz,pmq,moded
   double precision pyq,dpyq(iz),d2pyq(iz*(iz+1)/2)
   type(gtp_pystack), pointer :: pystack
   type(gtp_interaction), pointer :: intrec
```

8.16.7 Calculate disordered fractions from constituent fractions

This is used when there are several fraction sets of a phase. The values of the second fraction set (also called the disordered fraction set) is calculated by this subroutine. These disordered fractions can be used to calculate a "disordered" part of the Gibbs energy with its own set of parameters.

```
 subroutine calc_disfrac(lokph,lokcs,ceq)
! calculate and set disordered set of fractions from sitefractions
! The first derivatives are dxidyj.  There are no second derivatives
!   TYPE(gtp_fraction_set), pointer :: disrec
   implicit none
   integer lokph,lokcs
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine calc_disfrac2(phord,phdis,ceq)
! calculate and set disordered set of fractions from sitefractions
! The first derivatives are dxidyj.  There are no second derivatives
!   TYPE(gtp_fraction_set), pointer :: disrec
   implicit none
   TYPE(gtp_phase_varres), target :: phord
   TYPE(gtp_phase_varres), pointer :: phdis
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.16.8 Disorder constituent fractions

The Gibbs energy for the "partitioned" phases, like FCC and BCC which can have order/disorder transformations, see 2.5, the "ordered part" is calculated twice, once with the original constituent fractions and once with these set equal to their disordered value. The reason for this is that the "disordered part" should be complete i.e. include also the disordered part of the "ordered part" as the disordered partitions.

This is now an optional way to handle partioning, the recommended way is to simply add the Gibbs energy is simply added from the two fraction sets. This subroutine sets the fractions to their disordered values.

```
 subroutine disordery(phvar,ceq)
! sets the ordered site fractions in FCC and other order/disordered phases
! equal to their disordered value in order to calculate and subtract this part
! phvar is pointer to phase_varres for ordered fractions
   implicit none
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine disordery2(lokdcs,phvar,disrec,ceq)
! subroutine disordery2(phdis,phvar,disrec,ceq)
! sets the ordered site fractions in FCC and other order/disordered phases
! equal to their disordered value in order to calculate and subtract this part
! phvar is pointer to phase_varres for ordered fractions
! phdis is pointer to phase_varres for disordered fractions
   implicit none
   TYPE(gtp_phase_varres), pointer :: phvar
   TYPE(gtp_fraction_set), pointer :: disrec
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.16.9 Subroutine to calculate the UNIQUAC model

A routine to calculate configurational entropy and residual energy for the UNIQUAC model. It seems to work! Documentation of the model is missing (except comments inside this subroutine) but there is a macro file, “uniquac.OCM”, that explains how to enter species and calculate binary and ternary diagrams. There are several parts of this documentation describing how bits, species etc. has been been changed to accomodate this model.

```
 subroutine uniquac_model(moded,ncon,phres,ceq)
! Calculate the Gibbs energy of the UNIQUAC model (Abrams et al 1975)
! Modified 2018/Oct, Nov, Dec
! It returns UNIQUAC G and first and second derivatives of G in phres%gval etc.
! The values of q_i and r_i are be stored in species record, not identifiers
! The residual term should be stored as a UQTAU identifier
   implicit none
   integer moded,ncon
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_phase_varres), pointer :: phres
```

8.16.10 Set driving force for a phase explicitly

A failed attempt to handle convergence problems.

```
 subroutine set_driving_force(iph,ics,dgm,ceq)
! set the driving force of a phase explicitly
   implicit none
   type(gtp_equilibrium_data), pointer :: ceq
   integer iph,ics
   double precision dgm
```

8.16.11 Extract mass balance conditions

This is used in global grid minimization to extract the set of mass balance conditions. If the current conditions are not all mass balance there is an error return, otherwise the conditions of T and P, the total number of moles and the mole fractions of all components are returned.

```
 subroutine extract_massbalcond(tpval,xknown,antot,ceq)
! extract T, P,  mol fractions of all components and total number of moles
! for use when minimizing G for a closed system.  Probably redundant
   implicit none
   double precision, dimension(*) :: tpval,xknown
   double precision antot
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.16.12 Saving and restoring a phase constitution

These are used during step and map to help handling convergence problems

```
 subroutine save_constitutions(ceq,copyofconst)
! copy the current phase amounts and constituitions to be restored
! if calculations fails during step/map
! DANGEROUS IF NEW COMPOSITION SETS CREATED
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision, allocatable, dimension(:) :: copyofconst
---------------
 subroutine restore_constitutions(ceq,copyofconst)
! restore the phase amounts and constitutions from copyofconst
! if calculations fails during step/map
! DANGEROUS IF NEW COMPOSITION SETS CREATED
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision copyofconst(*)
----------------------------
   subroutine save_phase_constitutions(rw,ceq,copyofconst)
! copy the current phase amounts and constituitions to be restored
! trying to fix problems with saving invariants
! compared to reoutines above here abnorm is also saved ...
! rw=0 if save, 1 if restore
! NOTE different ceq may be used for save and restore!
   implicit none
   integer rw
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision, allocatable, dimension(:) :: copyofconst
```

8.16.13 Some auxillary subroutines

The first subroutine is used to handle the EEC to prevent that a solid phase with higher entropy than the liquid becomes stable even if that would lower the Gibbs energy for the system. The calc_mqmqa subroutines calculates the excess parameters of the MQMQA model, the entropy is calculated separately. The MQMQA model normally has a much simpler excess model than a phase modelled with CEF.

The tabder routine is used in the “CALC PHASE” command for interactive calculation of a phase by specifying explicitly the constitution. Sometimes that is useful to handle converge problems by setting a constitution of a phase before a “CALC NO_GLOBAL” command.

```
 subroutine calc_eec_gibbsenergy(phres,ceq)
! calculate an ideal Gibbs energy with just configurational entropy
! phres is pointer to phase_varres record for the phase
! for a solid phase with higher entropy than the liquid
! G = RT \sum_s a_s \sum_i y_si \ln(y_si)
! dG/dy_si = RT a_s (1+ln(y_si))
! d2G/dy_si^2 = RT a_s/y_si            all other 2nd derivatives zero
   implicit none
   TYPE(gtp_phase_varres), pointer :: phres
   TYPE(gtp_equilibrium_data), pointer :: ceq
 =======================
 subroutine calc_mqmqa(lokph,phres,mqf,ceq)
! Called from calcg_internal, calculates G for the mqmqa phase
! separate subroutine for the entropy which calculates all data in phres%mqf
   integer lokph
   type(gtp_phase_varres), pointer :: phres
   type(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_mqmqa_var), pointer :: mqf
 =======================
 subroutine setendmemarr(lokph,ceq)
! stores the pointers to all ordered and disordered endmemners in arrays
! intended to allow parallel calculation of parameters
! UNUSED ??
   implicit none
   integer lokph
   TYPE(gtp_equilibrium_data), pointer :: ceq
 =======================
 subroutine tabder(iph,ics,times,ceq)
! tabulate derivatives of phase iph with current constitution and T and P
   implicit none
   integer iph,ics,times
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.16.14 The Toop/Kohler excess model

This is an unfinished tentative routine to handle Toop/Kohler excess models.

```
 subroutine calc_toop(lokph,lokpty,moded,vals,dvals,d2vals,gz,toopx,ceq)
```

```
! Handle a binary interaction that is in a Toop or Kohler model
! toop is the link to the kohler-Toop record
! We come here only if the parameter is composition dependent using RK series
   implicit none
   integer moded,lokph
   TYPE(gtp_property), pointer :: lokpty
   TYPE(gtp_parcalc) :: gz
   double precision vals(6),dvals(3,gz%nofc)
   double precision d2vals(gz%nofc*(gz%nofc+1)/2)
   TYPE(gtp_equilibrium_data), pointer :: ceq
   TYPE(gtp_tooprec), pointer :: toopx
```

8.17 Grid minimizer, gtp3Y

When there is only mass balance conditions it is possible to use a global minimization technique to find start points for a more standard Newton-Raphson technique. The latter has the disadvantage that it can only find local minimas.

Tn the global minimization all solution phases are divided into a grid with fixed compositions and the Gibbs energy for each gridpoint is calculated. These gridpoints are then searched until one finds a set that encloses the given overall composition which represent the chemical potentials for the minimal Gibbs energy. It is thus necessary to know the overall composition. In principle one or more conditions could be fixed chemical potentials but in such a canse one may find it is impossible to use one or more of the chemical potentials because they are above any tangent plane.

The phases the gridpoints in the solution are then identified and possibly one can merge gridpoints in the same phase unless they are separated by a miscibility gap.

Some care should be taken with ordered phases as they have several identical sublattices and one thus can significantly reduce the number of gridpoints needed if this is taken into account. There are special subroutine to generate grids for the I2SL model, for crystalline phases with ions, for FCC and BCC phases with 4 sublattices for ordering and with permutation F or B set. Clement Introïni and coworkers at CEA, Cadarache have worked on improving the grid minimizer as well as the speed of convergence as they work with very large system with 15 or more components.

8.17.1 Global Gridminimizer

This is the main grid minimizing subroutine. It can be called also after an equilibrium calculation, see section 8.17.8, to check if there are any phases with a gridpoint below the calculated G hypersurface. That is useful if the conditions does not allow, for example if T is not known, that the gridminimizer is called before the equilibrium calculation.

```
 subroutine global_gridmin(what,tp,xknown,nvsph,iphl,icsl,aphl,&
      nyphl,cmu,ceq)
!
! Starting rewriting 2017-02-01
```

```
!
! finds a set of phases that is a global start point for an equilibrium
! calculation at T and P values in tp and known mole fraction in xknown
! It is intentional that this routine is independent of current conditions
! It returns: nvsph stable phases, list of phases in iphl, amounts in aphl,
! nyphl(i) is redundant, cmu are element chemical potentials of solution
! WHAT determine what to do with the results, 0=just return solution,
! 1=enter stable set and constitution of all phases in gtp datastructure
! and create composition sets if necessary (and allowed)
! what=-1 will check if any gridpoint is below current calculated equilibrium
! ?? removed what -1 170428/BoS
   implicit none
! nyphl(j) is the start position of the constitiuent fractions of phase j in
   integer, dimension(*) :: iphl,nyphl,icsl
   integer what,nvsph

   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision, dimension(2) :: tp
! cmu(1..nrel) is the chemical potentials of the solution
   double precision, dimension(*) :: xknown,aphl,cmu
```

8.17.2 Generate grid for I2SL, ordering option F and B etc.

There are several different grid generators depending on the type of phase for example ionic or with order/disorder transformation. It is also a possibility to generate a denser grid.

The ordered fcc, bcc and hcp with 4 sublattice model can have significantly reduced number of gridpoints. This is not yet implemented. For the ordered 4 sublattice model we can exclude all permutations of the endmembers, for example only include A:A:A:B but not A:A:B:A, A:B:A:A and B:A:A:A. This reduces the number of gridpoints that must be calculated.

The same when a phase have charged constituents with many anemembers that have net charge. This is implemented but not optimized.

```
 subroutine generate_grid(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! Different action depending of the value of mode,
! for mode<0:  (will no longer be used ... )
!    return the number of gridpoints that will be generated for phase iph in ngg
! for mode=0:
!    on entry ngg is dimension of garr
!    on exit ngg is number of generated gridpoints ...
!    return garr(i) gibbs energy and xarr(1,i) the compositions of gridpoint i
! for mode>0:
!    return site fractions of gridpoint mode in yarr, number of fractions in ny
!    iph is phase number, ngg is number of gridpoints, nrel number of elements,
! if mode=0:
```

```
!    return xarr mole fractions of gridpoints, garr Gibbs energy of gridpoints,
!    ngg is dimension of garr, gmax maximum G (start value for chem.pot)
! if mode>0:
!   "mode" is a gridpoint of this phase in solution, return number of
!   constituent fractions in ny and fractions in yarr for this gridpoint
! The current constitution is restored at the end of the subroutine
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
---------------
 subroutine generic_grid_generator(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! This generates grid for any phase
! mode=0 generate grid for phase iph and mole fraction and G for all points
!    ngg on entry is max number of gridpoints, on exit number of gridpoints
!    nrel is number of elements
!    xarr(1..nrel,gp) is composition of gripoint gp, garr(iv) its G
!    ny,yarr,gmax not used
! mode>0 return constitution for gridpoint number mode in yarr
!    iph is returned as phase index for gripoint mode
!    xarr(1..,nrel) the composition at the gripoint, garr not nused
!    ny is number of constituent fractions
!    yarr are the constituent fractions
!    gmax not used ??
!
   implicit none
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine generate_dense_grid(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! generates more gridpoints than default generate_grid
! Different action depending of the value of mode,
! for mode<0:
!    return the number of gridpoints that will be generated for phase iph in ngg
! for mode=0:
!    return garr(i) gibbs energy and xarr(1,i) the compositions of gridpoint i
! for mode>0:
!    return site fractions of gridpoint mode in yarr, number of fractions in ny
!    iph is phase number, ngg is number of gridpoints, nrel number of elements,
! if mode=0:
!    return xarr mole fractions of gridpoints, garr Gibbs energy of gridpoints,
!    ngg is dimension of garr
! if mode>0:
```

```
!   "mode" is a gridpoint of this phase in solution, return number of
!   constituent fractions in ny and fractions in yarr for this gridpoint
! The current constitution is restored at the end of the subroutine
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
---------------
 subroutine generate_ionliq_grid(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! generates gridpoints for ionic liquid (also dense)
! Different action depending of the value of mode,
! for mode<0:
!    return the number of gridpoints that will be generated for phase iph in ngg
! for mode=0:
!    return garr(i) gibbs energy and xarr(1,i) the compositions of gridpoint i
! for mode>0:
!    return site fractions of gridpoint mode in yarr, number of fractions in ny
!    iph is phase number, ngg is number of gridpoints, nrel number of elements,
! if mode=0:
!    return xarr mole fractions of gridpoints, garr Gibbs energy of gridpoints,
!    ngg is dimension of garr
! if mode>0:
!   "mode" is a gridpoint of this phase in solution, return number of
!   constituent fractions in ny and fractions in yarr for this gridpoint
! The current constitution is restored at the end of the subroutine
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
---------------
 subroutine generate_fccord_grid(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! This generates grid for a phase with 4 sublattice fcc/bcc/hcp ordering
! NO LONGER USED: mode<0 just number of gridpoints in ngg, for allocations
! mode=0 calculate mole fraction and G for all gridpoints
! mode>0 return constitution for gridpoint mode in yarr
   implicit none
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine generate_charged_grid(mode,iph,ngg,nrel,xarr,garr,ny,yarr,gmax,ceq)
! This generates grid for a phase with charged constituents
! mode<0 just number of gridpoints in ngg, needed for allocations
! mode=0 calculate mole fraction and G for all gridpoints
! mode>0 return constitution for gridpoint mode in yarr
   implicit none
   integer mode,iph,ngg,nrel,ny
   real xarr(nrel,*),garr(*)
   double precision yarr(*),gmax
   type(gtp_equilibrium_data), pointer :: ceq
```

8.17.3 Calculate a gridpoint

The Gibbs energy at given T, P is calculated for a gridpoint when its constitution is provided in yfra. The composition is also calculated and returned. Depending on the models many gridpoints may overlapp even for different constitutions.

```
 subroutine calc_gridpoint(iph,yfra,nrel,xarr,gval,ceq)
! called by global minimization routine
! Not adopted to charged crystalline phases as gridpoints have net charge
! but charged gripoints have high energy, better to look for neutral ones ...
   implicit none
   real xarr(*),gval
   integer iph,nrel
   double precision yfra(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.17.4 Calculate endmember

These subroutines ares used by the calc_gridpoint routine and also to calculate a user defined reference state..

```
 subroutine calcg_endmember(iphx,endmember,gval,ceq)
! calculates G for one mole of real atoms for a single end member
! used for reference states. Restores current composition (but not G or deriv)
! endmember contains indices in the constituent array, not species index
! one for each sublattice
   implicit none
   integer iphx
   double precision gval
   integer endmember(maxsubl)
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine calcg_endmemberx(iphx,endmember,gval,ceq)
! calculates G for single end member with current number of atoms
! used for reference states. Restores current composition (but not G or deriv)
! endmember contains indices in the constituent array, not species index
! one for each sublattice
   implicit none
   integer iphx
   double precision gval
   integer endmember(maxsubl)
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine calcg_endmember6(iph,endmember,gval,ceq)
! calculates G AND ALL DERIVATEVS wrt T and P for one mole of real atoms
! for a single end member, used for reference states.
! Restores current composition and G (but not deriv)
! endmember contains indices in the constituent array, not species index
! one for each sublattice
   implicit none
   integer iph
   double precision gval(6)
   integer endmember(maxsubl)
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.17.5 Calculate minimum of grid

The search starts from the lowest G value for the pure elements. (This seems obvious but makes it impossible to have a grid for only fcc with carbon dissolved as fcc does not exist for pure carbon). The selected gridpoints always represent a hyperplane (number of dimensions equal to that of the number of components) of Gibbs energy over the whole composition range. Then the whole grid is searched for the gridpoint that has the most negative deviation from this hyperplane. Then one gridpoint in the existing hyperplane is replaced by this in such a way that the overall mass balance is inside the points forming the plane. This is simply done by solving the mass balance equations replacing one grid point at a time with the new gridpoint. If any gridpoint has a negative amount that is not a correct set.

Then a new search is made and a new point is replaced until there are no grid points below the hyperplane. The points forming this plane represent the solution. There will always be as many points as components but several grid points may be in the same phase.

In order to speed up the search it was attempted to exclude gridpoints that were above the plane but that created trouble. There is obviously some higher geometry involved.

```
 subroutine find_gridmin(kp,nrel,xarr,garr,xknown,jgrid,phfrac,cmu,trace)
! there are kp gridpoints, nrel is number of components
! composition of each gridpoint in xarr, G in garr
! xknown is the known overall composition
! return the gridpoints of the solution in jgrid, the phase fraction in phfrac
! cmu are the final chemical potentials
   implicit none
   integer, parameter :: jerr=50
   integer kp,nrel
   integer, dimension(*) :: jgrid
   real xarr(nrel,*),garr(*)
   double precision xknown(*),phfrac(*),cmu(nrel)
   logical trace
```

8.17.6 Merge gridpoints in same phase

This subroutine tries to check if the number of gridpoints can be reduced by merging gridpoints in the same phase. Care must be taken that there can be miscibility gaps between points.

This subroutine may automatically create new composition sets if necessary (unless the user has set the appropriate bit to prevent that).

```
 subroutine merge_gridpoints(nv,iphl,aphl,nyphl,yphl,trace,nrel,xsol,cmu,ceq)
!
! BEWARE not adopted for parallel processing
!
! if the same phase has several gridpoints check if they are really separate
! (miscibility gaps) or if they can be murged.  Compare them two by two
! nv is the number of phases, iphl(i) is the index of phase i, aphl(i) is the
! amount of phase i, nyphl is the number of site fractions for phase i,
! and yphl is the site fractions packed together
!
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer nv,nrel
   integer, dimension(*) :: iphl,nyphl
   double precision, dimension(*) :: aphl,yphl,cmu
   logical trace
   real xsol(maxel,*)
```

8.17.7 Set constitution of metastable phases

Phases not part of the final solution will have their constitution set to a gridpoint that is closest to the final hyperplane. This is an important step because the gridminimizer will normally not find the most stable set of phases.

```
 subroutine set_metastable_constitutions2(pph,nrel,nphl,iphx,xarr,garr,&
      nr,iphl,cmu,ceq)
! this subroutine goes through all the metastable phases
! after a global minimization and sets the constituion to the most
! favourable one.  Later care should be taken that exiting higher composition
! sets are not set equal to the stable
! pph   number of phases for which a grid has been calculated
! nrel  number of components
! nphl(p) is last gridpoint for phase(p), nphl(0)=0, p=1,pph
! iphx(p) phase number of phase(p) (skipping dormant and suspended phases)
! xarr(1..nrel,i)  composition of gridpoint i
! garr(i)  Gibbs energy/RT for gridpoint i
! nr    is the number of stable phases in the solution
! iphl(s) the phase number of the stable phases s (not ordered)
! cmu   are the chemical potentials/RT of the solution
! ceq   equilibrium record
! called by global_gridmin
   implicit none
   integer pph,nrel,nr
   integer, dimension(0:*) :: nphl
   integer, dimension(*) :: iphl,iphx
   double precision, dimension(*) :: cmu
   real garr(*),xarr(nrel,*)
   type(gtp_equilibrium_data), pointer :: ceq
```

8.17.8 Check of equilibrium using grid minimizer

These subroutines can be used after an equilibrium calculation when the conditions does not allow the grid minimizer to be used (if T, P are not known or all conditions are not mass balance). It calculates the same grid as the global gridmin and then checks if any gripoint is below the current hyperplane definde by the calculated chemical potentials. If so it reports the phase and composition. It does not correct the equilibrium.

```
 logical function global_equil_check1(mode,addtuple,yfr,ceq)
! subroutine global_equil_check(ceq,newceq)
!
! This subroutine checks there are any gridpoints below the calculated solution
! if not it is taken as a correct global equilibrium
! This avoids creating any new composition sets but may fail in some cases
! to detect that the equilibrium is not global.
! mode=1 means try to recalculate equilibrium if not global (not implemented)
! if a gridpoint below is found addtuple and yfr returned with this
   implicit none
   integer mode,addtuple
   double precision, allocatable, dimension(:) :: yfr
   TYPE(gtp_equilibrium_data), pointer :: ceq
==========================
 subroutine check_all_phases(mode,ceq)
!
! This function check for all phases if there are any gridpoints
! closer (or below) to the current calculated solution
! if so it changes the composition of the phase
! If a gridpoint is BELOW the current plane an error code is returned
! phase should be stable with another composition an error code is returned
! It does not creating any new composition sets
! It can be usd during STEP/MAP to update compositions of metastable
! phases which have become stuck in a local minimium
! if error 4365 or 4364 is set mode will return index in meqrec%phr
! of the phase that should be stable
   implicit none
   integer mode
   TYPE(gtp_equilibrium_data), pointer :: ceq
 ======================
 subroutine check_phase_grid(iph,jcs,pceq,ceq)
!
! This function check for A SINGLE PHASE if there are any gridpoints
! closer (or below) to the current calculated solution if so it
! changes the composition of the phase If a gridpoint is below the
! phase should be stable with another composition an error code is
! returned It does not creating any new composition sets but may fail
! It can be usd during STEP/MAP to update compositions of metastable
! phases which have become stuck in a local minimium
! NOTE pceq is a pointer to a copy of the real equilibrium record
! ceq is a pointer to the real equilibrium record
! jcs is returned as the composition set that should be stable (if any)
   implicit none
   integer iph,jcs
   TYPE(gtp_equilibrium_data), pointer :: ceq,pceq
```

8.17.9 Separate composition set constitions that has merged

During step and map the constitution of two composition sets may merge above the top of the miscibility gap. This routine will check for that and separate them if that has happend.

```
 subroutine separate_constitutions(ceq)
! This is called during step/map
! Go through all entered phases and if there are two composition sets
! that have similar constitutions then separate them
! Used during mapping of for example Fe-Cr to detect the miscibility gap
    implicit none
    TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.17.10 Check if two phases are allotropes

Check if two phases are allotropes, i.e. have identical models and composition.

The subroutine fixedcomposition is needed to handle the case when a stoichiometric phase changes to another stochiometric phase with exactly the same composition. The minimizer cannot handle a system with two phases with identical composition and the same Gibbs energy.

```
  logical function allotropes(irem,iadd,iter,ceq)
! This function return TRUE if the phases indicated by IREM and IADD both have
! fixed and identical composition, i.e. they are componds and allotropes
! Such a transition can cause problems during a STEP command.
    implicit none
    TYPE(gtp_equilibrium_data), pointer :: ceq
    integer iadd,irem,iter
---------------
 logical function same_stoik(iph,jph)
! MAYBE IDENTICAL TO ALLOTROPES ?
! return TRUE if phase iph and jph are both stoichiometric and have the
! same composition  Used to check when adding a phase during equilibrium
! calculation as it normally fails to have two such phases stable
   implicit none
   integer iph,jph
---------------
 logical function fixedcomposition(iph)
! returns TRUE if phase cannot vary its composition
   integer iph
```

8.18 Miscellaneous things

Things that does not fit anywhere else.

8.18.1 Phase record location

```
 integer function phvarlok(lokph)
! return index of the first phase_varres record for phase with location lokph
! needed for external routines as phlista is private
   implicit none
   integer lokph
```

8.18.2 Numbers an interaction tree for permutations

For permutations it turned out to be necessary to keep track of the individual interaction records to know the permutation. This subroutine indexes the interaction records for each endmember of a phase.

```
 subroutine palmtree(lokph)
! Initiates a numbering of all interaction trees of an endmember of a phase
```

```
   implicit none
   integer lokph
```

8.18.3 Sorting constituent fractions

```
 subroutine sortinphtup(n,m,xx)
! subroutine to sort the values in xx which are in phase and compset order
! in phase tuple order.  This is needed by the TQ interface
! The number of values belonging to the phase is m (for example composition)
! argument ceq added as new composition sets can be created ...
   integer n,m
!   double precision xx(n*m)
   double precision xx(*)
!   type(gtp_equilibrium_data), pointer :: ceq
```

8.18.4 Get model parameter identifier index

Extract the model parameter index.

```
  integer function get_mpi_index(mpi)
! Return the index of a model parameter identifier
    character mpi*(*)
---------------
  integer function getmqindex()
! This is necessary because mqindex is private, replaced by getmpiindex ...
```

8.18.5 Check that certain things are allowed

To have a uniform check if something is allowed. One must not enter a phase before there are any elements for example and a user is not allowed to enter a new element when there is already a phase entered.

```
 logical function allowenter(mode)
! Check if certain commands are allowed
! mode=1 means entering an element or species
! mode=2 means entering a phase
! mode=3 means entering an equilibrium
! returns TRUE if command can be executed
   implicit none
   integer mode
```

8.18.6 Check proper symbol

There are many symbols and names in the GTP package. In general a symbol must start with a letter A-Z. All symbols are also case insensitive, i.e. upper and lower case are treated as the same. Most symbols may contain digits as second and later character. Some symbols and names may contain special characters like "_" and others.

This subroutine checks this.

```
 logical function proper_symbol_name(name,typ)
! checks that name is a proper name for a symbol
! A proper name must start with a letter A-Z
! for typ=0 it must contain only letters, digits and underscore
! for typ=1 it may contain also +, - maybe ?
! It must not be equal to a state variable
   implicit none
   integer typ
   character name*(*)
```

8.18.7 List things for debugging

List constitutions in some order or other.

```
 subroutine compmassbug(ceq)
! debug subroutine
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine list_free_lists(lut)
! for debugging the free lists and routines using them
   implicit none
   integer lut
```

8.18.8 The amount of a phase is set to a value

I am not sure when this is used or needed.

```
 subroutine set_phase_amounts(jph,ics,val,ceq)
! set the amount formula units of a phase. Called from user i/f
! iph can be -1 meaning all phases, all composition sets
   implicit none
   integer jph,ics
   double precision val
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.18.9 Set the default constitution of a phase

I am not sure when this is used and if it is needed.

```
 subroutine set_default_constitution(iph,ics,ceq)
! the current constitution of (iph#ics) is set to its default constitution
!  (if any), otherwise a random value.  The amount of the phase not changed
   implicit none
   integer iph,ics
   TYPE(gtp_equilibrium_data), pointer :: ceq
```

8.18.10 Subroutine to prepare for an equilibrium calculation

This is useful but unfinished.

```
 subroutine todo_before(mode,ceq)
! this could be called before an equilibrium calculation
! It should remove any phase amounts and clears CSSTABLE
! DUMMY
!
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer mode
```

8.18.11 Subroutine to clean up after an equilibrium calculation

This now checks if there are any composition sets with the AUTO bit set meaning that they have been created by the grid minimizer in this equilibrium calculation. If so it tries to move the stable phases to the lowest possible composition set, taking care that user defined composition sets with a given default constitution is honored, i.e a fcc carbide is not set as composition set 1 if the user has defined a metallic fcc phase with low carbon as the first.

It then removes unstable composition sets with the AUTO bit set and any stable composition sets with the AUTO bit set have this bit cleared.

```
 subroutine todo_after_found_equilibrium(mode,addtuple,ceq)
! this is called after an equilibrium calculation by calceq2 and calceq3
! It marks stable phase (set CSSTABLE and remove any CSAUTO)
! It removes redundant unstable composition sets created automatically
! (CSAUTO set).  It will also shift stable composition sets to loweest
! possible (it will take into account if there are default constituent
! fractions, CSDEFCON set).
! mode determine some of the actions, at present only >0 or <0 matters
!
! >>>>>>>>>>> THIS IS DANGEROUS IN PARALLEL PROCESSING
! It should work in step and map as a composition set that once been stable
! will never be removed except if one does global minimization during the
! step and map. The function global_equil_check works on a copy of the
! ceq record and creates only a grid, it does not create any composition sets.
! NOTE that automatically entered metallic-FCC and MC-carbides may shift
! composition sets. Such shifts can be avoided by manual entering composition
! sets with default constitutions, but that does not always work as comparing
! a stable constitution with several defaults is not trivial ...
!
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   integer mode,addtuple
```

8.18.12 Select composition set for stable phase

After an equilibrium calculation there may be automatically created composition sets by the grid-minimizer that are not needed. These routines tries to remove unneeded sets and also shift the used ones to the lowest composition set. It tries to take into account the default constitutions for user defined composition sets for example an fcc#1 with a small amount of C and fcc#2 with a large amount of C. Then a carbide rich fcc phase should be fcc#2 and not fcc#1 even if fcc#1 is not stable.

```
 subroutine checkdefcon(lokics,lokjcs,fit,ceq)
! check if composition of lokics fits default constitution in lokjcs
! return TRUE if lokics fits default in lokjcs
! NOTE lokics and lokjcs can be the same!!
   integer lokics,lokjcs,fit
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine shiftcompsets(ceq)
! check phase with several composition sets if they should be shifted
! to fit the default constitution better
! IGNORE UNSTABLE COMP.SETS
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine switch_compsets2(lokph,ics1,ics2,ceq)
! copy constitution and results from ic2 to ic1 and vice versa
   integer lokph,ics1,ics2
   type(gtp_equilibrium_data), pointer :: ceq
```

8.18.13 Select composition set for stable phase, maybe not used

```
 subroutine select_composition_set(iph,ics,yarr,ceq)
! PROBABLY NOT USED but should be implemenented
! if phase iph wants to become stable and there are several user defined
! composition sets with default composition limits this subroutine tries to
! select the one that fits these limits best.
! For example if an FCC phase that could be an austenite (low carbon content)
! or a cubic carbo-nitride (high carbon or nitrogen content, low vacancy)
! Less easy to handle ordered phases like B2 or L1_2 as ordering can be
! in any sublatittice ... but with option B and F that is possible
   implicit none
   TYPE(gtp_equilibrium_data), pointer :: ceq
   double precision, dimension(*) :: yarr
   integer iph,ics
```

8.18.14 Check consistency of phase_varres

```
 subroutine verify_phase_varres_array(ieq,verbose)
! This subroutine checks that the phase varres array is consistent
! in equilibrium ieq.  For ieq=1 it also checks the free list
! UNFINISHED and not yet used BUT IMPORTANT
   implicit none
   integer ieq,verbose
```

8.18.15 Not documented

```
 subroutine set_emergency_startpoint(mode,phl,amfu,ceq)
! this is called if no previous equilibrium and if grid minimizer
! cannot be used.  Select for each element a phase with as much of that
! element as possible to set as stable. Set the remaining phases to a default
! composition.  It will never create any compositon sets
!
   implicit none
   integer mode,phl(*)
   double precision amfu(*)
   TYPE(gtp_equilibrium_data), pointer :: ceq
---------------
 logical function ocv()
! returns TRUE if GSVERBOSE bit is set
---------------
 integer function ceqsize(ceq)
! calculates the size in words (4 bytes) of an equilibrium record
   implicit none
   type(gtp_equilibrium_data), pointer :: ceq
---------------
 integer function vssize(varres)
! calculates the size in words (4 bytes) of a phase_varres record
   implicit none
   type(gtp_phase_varres) :: varres
```

```
--------------
  logical function inveq(phases,ceq)
! Only called for mapping tie-lines not in plane.  If tie-lines in plane
! then all nodes are invariants.
    integer phases
    type(gtp_equilibrium_data), pointer :: ceq
```

8.19 TP functions, gtp3Z

TP functions are a sub package inside GTP. These can store simple expressions depending on T and P including unary functions LN and EXP (and maybe more). They can return a function value and the first and second derivative with respect to T and P. There are three datatypes associated with these functions described in section 7.

8.19.1 Initiate the TP fun package

```
 subroutine tpfun_init(nf,tpres)
! allocate tpfuns and create a free list inside the tpfuns
   implicit none
   integer nf
! use tpres declared externally for parallel processing
  TYPE(tpfun_parres), dimension(:), allocatable :: tpres
```

8.19.2 Utilities

This is needed because freetpfun is private.

```
 integer function notpf()
! number of tpfunctions because freetpfun is private
   implicit none
```

8.19.3 Find a TP function

```
 subroutine find_tpfun_by_name(name,lrot)
! returns the location of a TP function
! if lrot>0 then start after lrot, this is to allow finding with wildcard *
   implicit none
   integer lrot
   character name*(*)
---------------
 subroutine find_tpfun_by_name_exact(name,lrot,notent)
! returns the location of a TP function, notent TRUE if not entered
    implicit none
    integer lrot
    logical notent
    character name*(*)
```

8.19.4 Evaluate a TP function

Calculate the value and first and second derivatives.

```
 subroutine eval_tpfun(lrot,tpval,result,tpres)
!    subroutine eval_tpfun(lrot,tpval,symval,result)
! evaluate a TP function with several T ranges
   implicit none
   integer lrot
   double precision tpval(2),result(6),xxx
! changes to avoid memory leak in valgrind
   TYPE(tpfun_parres), dimension(*) :: tpres
```

8.19.5 List the expression of a TP function

```
 subroutine list_tpfun(lrot,nosym,str)
! lists a TP symbols with several ranges into string str
! lrot is index of function, if nosym=0 the function name is copied to str
   implicit none
   character str*(*)
   integer nosym,lrot
---------------
 subroutine list_all_funs(lut)
! list all functions except those starting with _ (parameters)
   implicit none
   integer lut
```

8.19.6 List unentered TP functions

For use when reading from a TDB file as functions may call each other. The TDB file is rewound until all functions found or are missing in the TDB file.

```
 subroutine list_unentered_funs(lut,nr)
! counts and list functions with TPNOTENT bit set if lut>0
   implicit none
   integer lut,nr
```

8.19.7 Compiles a text string as a TP function

Very messy, requires total rewriting as the whole TP package.

```
 subroutine ct1xfn(string,ip,nc,coeff,koder,fromtdb)
!...compiles an expression in string from position ip
!     it can refer to T and P or symbols in fnsym
!     compiled expression returned in coeff and koder
!
! >>> this is very messy
!
!...algorithm for function extraction
! 10*T**2 -5*T*LOG(T) +4*EXP(-5*T**(-1))
!
! AT LABEL 100 start of expression or after (
! sign=1
! -, sign=-1                            goto 200
! +, skip
!
! AT LABEL 200 after sign
! if A-Z                                goto 300
! if 0-9, extract number                goto 400
! (                                     goto 100
! ;                                     END or ERROR
! empty                                 END or ERROR
! anything else                         ERROR
!
! AT LABEL 300 symbol
! if T or P, extract power if any incl () goto 400
! unary fkn? extract (                  goto 100
! symbol                                goto 400
!
! AT LABEL 400 after factor
! -, sign=-1                            goto 200
! sign=1
! +, skip                               goto 200
! )                                     goto 400
! ** or ^ extract and store power incl () goto 400
! *                                     goto 200
! empty                                 goto 900
!
! for TDB compatibility skip #
!
! DO NOT allow unary functions ABOVE(TB) and BELOW(TB)
! check consistency
   implicit none
   integer ip,nc,koder(5,*)
   character string*(*)
   double precision coeff(*)
   logical fromtdb
```

```
---------------
 subroutine ct1getsym(string,ip,symbol)
!...extracts an symbol
!   implicit double precision (a-h,o-z)
   implicit none
   integer ip
   character string*(*),symbol*(*)
---------------
 subroutine ct1power(string,ip,ipower)
!...extracts an integer power possibly surrounded by ( )
   implicit none
   integer ip,ipower
   character string*(*)
---------------
 subroutine ct1mfn(symbol,nranges,tlimits,lokexpr,lrot)
!...creates a root record with name symbol and temperature ranges
! highest T limit is in tlimits(nranges+1)
!   implicit double precision (a-h,o-z)
   implicit none
   integer nranges,lrot
   character*(*) symbol
   TYPE(tpfun_expression), dimension(*) :: lokexpr
   real tlimits(*)
---------------
 subroutine ct2mfn(symbol,nranges,tlimits,lokexpr,lrot)
!...stores a TPfun in an existing lrot record with name symbol
! and temperature ranges, highest T limit is in tlimits(nranges+1)
   implicit none
   integer nranges,lrot
   character*(*) symbol
   TYPE(tpfun_expression), dimension(*) :: lokexpr
   real tlimits(*)
---------------
 subroutine ct1mexpr(nc,coeff,koder,lrot)
!...makes a datastructure of an expression. root is returned in lrot
!   implicit double precision (a-h,o-z)
   implicit none
   integer nc,koder(5,*)
!   TYPE(tpfun_expression), pointer :: lrot
   TYPE(tpfun_expression) :: lrot
!   TYPE(tpfun_expression), pointer :: noexpr
   double precision coeff(*)
---------------
 subroutine ct1efn(inrot,tpval,val,tpres)
!...evaluates a datastructure of an expression. Value returned in val
!     inrot is root expression tpfunction record
!     tpval is valuse of T and P, symval is values of symbols
! first and second derivatives of T and P also calculated and returned
! in order F, F.T, F.P, F.T.T, F.T.P, F.P.P
!
! if function already calculated one should never enter this subroutine
!
! It can call "itself" by reference to another TP function and for
! that case one must store results in levels.
   implicit none
   double precision val(6),tpval(*)
   TYPE(tpfun_expression), pointer :: inrot
   TYPE(tpfun_parres), dimension(*) :: tpres
```

8.19.8 Calculate the Einstein function

This is useful for endmembers describing the vibrational heat capacity as a series of Einstein functions.

```
 subroutine tpfun_geinstein(tpval,gg,ff,dfdt,dfdp,d2fdt2,d2fdtdp,d2fdp2)
! evaluates the integrated Einstein function (including 1.5*R) INTEIN/GEIN
! gg is the value of the Einstein THETA
! ff is a constant factor which should be multiplied with all terms
! ff is overwritten with the Einstein function (multiplied with ff in)
! the other parameters are derivatives of the integrated Einstein function
   implicit none
   double precision tpval(*)
   double precision gg,ff,dfdt,dfdp,d2fdt2,d2fdtdp,d2fdp2
```

8.19.9 Utility to list a TP function

```
 subroutine ct1wfn(exprot,tps,string,ip)
!...writes an expression into string starting at ip
!     lrot is an index to an tpexpr record
!   implicit double precision (a-h,o-z)
   implicit none
   character tps(2)*(*)
   character string*(*)
---------------
 subroutine ct1wpow(string,ip,tps,mult,npow)
!...writes "ips" with a power if needed and a * before or after
!   implicit double precision (a-h,o-z)
   implicit none
   integer ip,mult,npow
   character string*(*),tps*(*)
```

8.19.10 Enter a TP function interactively

```
 subroutine enter_tpfun_interactivly(cline,ip,longline,jp)
! interactive input of a TP expression, whole function returned in longline
!   implicit double precision (a-h,o-z)
   implicit none
   integer ip,jp
   character cline*(*),longline*(*)
```

8.19.11 Deallocate TP function

Deallocates a TP function with result arrays in all equilibria.

```
 subroutine tpfun_deallocate
! deallocates all arrays associated with a TP function
```

8.19.12 Enter a dummy TP function

```
 subroutine enter_tpfun_dummy(symbol)
! creates a dummy entry for a TP function called symbol, used when entering
! TPfuns from a TDB file where they are not in order
   implicit none
   character*(*) symbol
------------------
 subroutine store_tpfun_dummy(symbol)
! creates a dummy entry for a TP function called symbol, used when entering
! TPfuns from a TDB file where they are not in order
   implicit none
   character*(*) symbol
```

8.19.13 Some more utilities for TP function

One handles nested TP function, other abbreviated names.

```
 subroutine nested_tpfun(lrot,tpval,nyrot)
! called from ct1efn when a it calls another TP function that must be
! evaluated.  nyrot is the link to the ct1efn in the correct range
!   implicit double precision (a-h,o-z)
   implicit none
   integer lrot
   double precision tpval(2)
   TYPE(tpfun_expression), pointer :: nyrot
! use lowest range for all T values lower than first upper limit
! and highest range for all T values higher than the next highest limit
! one should signal if T is lower than lowest limit or higher than highest
! used  saved reults if same T and P
```

8.19.14 Routines for optimizing coefficients and some other things

They are utility routines for the assessment program to manipulate the coefficients in the models.

```
 subroutine enter_optvars(firstindex)
! enter variables for optimization A00-A99
   implicit none
   integer firstindex
---------------
 subroutine find_tpsymbol(name,type,value)
! enter variables
   implicit none
! type=0 if function, 1 if variable, 2 if optimizing variable
   integer type
   character name*(lenfnsym)
   double precision value
---------------
 subroutine store_tpconstant(symbol,value)
! enter variables
   implicit none
   character symbol*(lenfnsym)
   double precision value
---------------
 subroutine change_optcoeff(lrot,value)
! change value of optimizing coefficient.  lrot is index
! -1 means just force recalculate
   implicit none
   integer lrot
   double precision value
---------------
 subroutine force_recalculate_tpfuns
! force recalculation of all tpfuns by incrementing an integer in tpfuns
---------------
 subroutine get_value_of_constant_name(symbol,lrot,value)
! get value (and index) of a TP constant.  lrot is index
   implicit none
   integer lrot
   character symbol*(*)
   double precision value
---------------
 subroutine get_value_of_constant_index(lrot,value)
! get value of a TP constant at known lrot
```

```
   implicit none
   integer lrot
   double precision value
---------------
 subroutine get_all_opt_coeff(values)
! get values of all optimizing coefficients
   implicit none
   double precision values(*)
---------------
 subroutine delete_all_tpfuns
! delete all TPFUNs.  No error if some are already deleted ...
! note: tpres is deallocated when deleting equilibrium record
```

8.19.15 Saving and reading unformatted TP funs

These are used to save and read TP functions from an unformatted file.

```
 subroutine save0tpfun(lfun,iws,jfun)
! save one tpfun (or parameter) with index jfun in workspace iws
!   implicit double precision (a-h,o-z)
   implicit none
   integer lfun,iws(*),jfun
---------------
 subroutine read0tpfun(lfun,iws,jfun)
! read one TPfun from workspace
   implicit none
   integer lfun,jfun,iws(*)
```

8.19.16 Routines used during assessments and debugging

Used to list functions using a coefficient and some other things.

```
 subroutine makeoptvname(name,indx)
    implicit none
    character name*(*)
    integer indx
---------------
 subroutine findtpused(lfun,string)
! this routine finds which other TPFUNS (including parameters) that
! use the TPFUN lfun.  It is used when listing optimizing coefficients
   implicit none
   integer lfun
   character string*(*)
---------------
```

```
 subroutine list_tpfun_details(lfun)
! listing the internal datastructure of all tpfuns
! converts all TP functions to arrays of coefficients with powers of T
   implicit none
   integer lfun
```

8.19.17 Routines used writing SOLGASMIX files

In FactSag [27] e and various software based on SOLGASMIX it is not possible to enter functions which can call each other and be used as model parameters. The routines below were added to write a DAT file with only coefficients.

This section is no longer supported and it contains bugs.

```
 subroutine tpfun2coef(ctpf,ntpf,npows,text)
! called by saveadatformat in gtp3C to generate SOLGASMIX DAT files
! converts all TP functions to arrays of coefficients with powers of T
   implicit none
   integer ntpf,npows
   type(gtp_tpfun2dat) :: ctpf(*)
   character text*(*)
---------------
 subroutine list_tpascoef(lut,text,paratyp,i1,npows,factor,ctpf)
! writes a parameter in DAT format
! text contains the stoichiometries written with the format 1x,F11.6
! it can be very long if there are many coefficients.
   implicit none
   integer lut,i1,npows,paratyp
   character text*(*)
! this is a factor that may be multiplied with all coefficients for
! phases like sigma which has only a disordered part. Also ionic liquid
   double precision factor
   type(gtp_tpfun2dat) :: ctpf(*)
---------------
 subroutine tpf2c(ctpf,lfun,done)
! convert TPfun lfun to an array of coefficients with powers of T
! if this TP function already converted just return
! if this TP function calls another TP function not converted return error
!
   implicit none
   integer lfun
   logical done
   type(gtp_tpfun2dat) :: ctpf(*)
---------------
 subroutine tpf2cx(ctpf,lfun,nrange,cfun1)
! convert TPfun lfun to an array of coefficients with powers of T
! if this TP function already converted just return
! if this TP function calls another TP function not converted return error
   implicit none
   integer lfun,nrange
   type(gtp_tpfun_as_coeff) :: cfun1
   type(gtp_tpfun2dat) :: ctpf(*)
!   type(gtp_equilibrium_data), pointer :: ceq
---------------
 subroutine adjust1range(lfun,nr1,nrange,krange,ctp1,nr2,ctp2)
! check if ctp1 range nr1 must be split in more ranges due to tbreaks in ctp2
! nrange is the total number of ranges of ctp1
! There are 10 ranges allocated for all, nr1 and nr2 are the used ranges
   implicit none
   integer lfun,nr1,nr2,krange,nrange
   type(gtp_tpfun_as_coeff) :: ctp1,ctp2
---------------
 subroutine adjustranges(nr1,ctp1,nr2,ctp2)
! add ctp2 to ctp1 which to have the same ranges and breakpoints
! nr1 and nr2 give the number of T-ranges and breakpoints
! add coefficients of ctp1 and ctp2 for each range
! NOTE these already multiplied with the coefficents!!
! There are 10 coefficients allocated for all functions.
! the added function is returned as ctp1
   implicit none
   integer nr1,nr2
   type(gtp_tpfun_as_coeff) :: ctp1,ctp2
---------------
 subroutine add1tpcoeffs(i1,ctp1,i2,ctp2,i3,ctp3)
! ctp3 is created with added coefficents from range i1 in ctp1
! and range i2 in cp2 with same tpower.  Normally i3=1
   implicit none
   integer i1,i2,i3
   type(gtp_tpfun_as_coeff) :: ctp1,ctp2,ctp3
---------------
 subroutine checkpowers(nc1,lfun,tpow1,npow,usedpow)
! check powers used in TP functions
! There can be several terms with same power ...
! nc1 is the maximal number of coefficients for each range (maxnc in fact)
   implicit none
   integer tpow1(*),nc1,lfun,usedpow(*),npow
---------------
 subroutine sortcoeffs(nc1,lfun,coeff1,tpow1)
! sort the coefficients in order power: 0 1 TlnT 2 3 -1; 7 -9 -2 other1 other2
!                                       1 2  3   4 5  6; 7  8  9   10    11
! other powers are 3, 4, -8 (meaning sqrt(t)) and maybe more
! tpowi is array giving the T power for coeffi,  101 means T*ln(T)
! ANY CHANGE OF POWERS MUST BE MADE ALSO IN ... CHECKPOWERS
! There can be several terms with same power ...
! nc1 is the maximal number of coefficients for each range (maxnc in fact)
   implicit none
   integer tpow1(*),nc1,lfun
   double precision coeff1(*)
```

9 Summary

Thats all! Be happy YOU did not have to write all this.

References

[1] http://www.opencalphad.org

[2] sundmanbo/opencalphad repository at https://www.github.com

[3] Ebert Alvares, Paul Jerabek, Yuanyuang Shang, Archa Santhosh, Claudio Pistidda, Tae Wook Hea, Bo Sundman, Martin Dornheim, *Modeling the thermodynamics of the FeTi hydrogenation under para-equilibrium: An ab-initio and experimental study*, Calphad **77** (2022) 102426

[4] J Herrnring, B Sundman, P Staron and B Klusemann, *Modeling precipitation kinetics for multi-phase and multi-component systems using particle size distributions via a moving grid technique*, Acta Materialia, (2021) **215** 117053

[5] Bo Sundman, Nathalie Dupin and Bengt Hallstedt, *Algorithms useful for calculating multi-component equilibria, phase diagrams and other kinds of diagrams*, Calphad **75** (2021) 102330

[6] B Sundman, U R Kattner, M Hillert, M Selleby, J Ågren, S Bigdeli, Q Chen, A Dinsdale, B Hallstedt A Khvan, H Mao, R Otis, *A method for handling the extrapolation of solid crystalline phases to temperatures far above their melting point*, Caphad **68**, (2020), 101737

[7] C. Introïni, J. Sercombe and B. Sundman, *Development of a robust, accurate and efficient coupling between PLEIADES/ALCYONE 2.1 fuel performance code and the OpenCalphad thermo-chemical solver*, Nucl Eng and Design **369**, (2020) 110818

[8] Karl Samuelsson, Jean-Christophe Dumas, Bo Sundman and Marc Lainet, *An improved method to evaluate the "Joint Oxide-Gaine" formation in (U,Pu)O_2 irradiated fuels using GERMINAL V2 code coupled to Calphad thermodynamic computations*, EPJ Nuclear Sci. Technol. **6** (2020) 47 https://doi.org/10.1051/epjn/2020008

[9] Jan Herrning, Bo Sundman and Benjamin Klusemann, *Diffusion-driven microstructure evolution in OpenCalphad*, Computational Materials Science, **171**, (2020), 109236

[10] Karl Samuelsson, Jean-Christophe Dumas, Bo Sundman, Jerome Lamontane and Christine Guéneau, *Simulation of the chemical state of high burnup (U,Pu)O_2 fuel in fast reactors bases on thermodynamic calculations,* J of Nuclear Materials, **532** (2020) 151969

[11] M Enoki, B Sundman, M H E Sluiter, M Selleby and H Othtani, *Calphad Modeling of LRO and SRO using ab initio Data*, Metals (2020), 10, 998, doi:10.3390/met10080998

[12] J Li, B Sundman, J G M Winkelman, A I Vakis and F Picchioni *Implementation of the UNUQUAC model in the OpenCalphad software*, Fluid Phase Equil, **907** (2020) 112398

[13] B Sundman, U R Kattner, C Sigli, M Stratmann, Romain Le Tellier, M Palumbo and S G Fries, *The OpenCalphad thermodynamic software interface*, Computational Matetrials Science, **125** (2016) 188–196

[14] HMS software package (2022), part of OpenCalphad documentation.

[15] B Sundman, X-G Lu and H Ohtani, *The implementation of an algorithm to calculate thermodynamic equilibria for multicomponent systems with non-ideal phases in a free software*, Computational Materials Science, **101** (2015) 127-137

[16] B Sundman, U Kattner, M Palumbo and S G Fries, *OpenCalphad - a free thermodynamic software*, Integrating Materials and Manufacturing Innovation, **4**:1 (2015), open access

[17] G Lambotte and P Chartrand, J Chem. Thermodynamics, **43** (2011) 1678–1699

[18] W Xiong, H Zhang, L Vitos and M Selleby, Magnetic phase diagram of the Fe-Ni system Acta Materialia **59** (2011) 521-530

[19] M Hillert, M Selleby and B Sundman, *An attempt to correct the quasichemical model*, Acta Mater. **57** (2009) 5237-5244

[20] Q Chen and B Sundman, *Modeling of Thermodynamic Properties for BCC, FCC, Liquid and Amorphous Iron*, J. of Phase Equil. **22** (2001)

[21] A D Pelton, *A General "Geometric" Thermodynamic Model for Multicomponent Solutions*, Calphad, **25**, (2001), 319–328

[22] A D Pelton and P Chartrand, Met. Mat. Trans. A **32A** (2001) 1355–1380

[23] P Chartrand and A D Pelton, Met. Mat. Trans. A **32A** (2001) 1397–1407

[24] A D Pelton, P Chartrand, G Eriksson, Met. Mat. Trans. A **32A** (2001) 1409–1416

[25] P Chartrand and A D Pelton *On the Choice of "Geometric" Thermodynamic Models*, J Phase Equil, **21**, (2000), 141-147

[26] B Sundman and J Ågren, *A Regular solution model for phases with several compoents and sublattices, suitable for computer applications*, J Phys. Solids, **42**, (1981), 297–301

[27] C. W. Bale, E. Bélisle, P. Chartrand, S. A. Decterov, G. Eriksson, A.E. Gheribi, K. Hack, I. H. Jung, Y. B. Kang, J. Melançon, A. D. Pelton, S. Petersen, C. Robelin. J. Sangster, P. Spencer and M-A. Van Ende, *FactSage Thermochemical Software and Databases - 2010 - 2016*, Calphad, **54**, (2016) pp 35–53

[28] https://factsage.com/

[29] M Hillert, *Phase Equilibria, Phase Diagrams and Phase Transformations*, Cambr. univ. press (2008)

[30] H L Lukas, S G Fries and B Sundman, *Computational Thermodynamics, the Calphad method*, Cambr. univ. press (2007)

[31] F Kongoli, Y Dessureault and A D Pelton, *Thermodynamic Modeling of Liquid Fe-Ni-Cu-Co-S Mattes* Met. Mat. Trans. B **29B** (1998) pp 591–601

[32] Mats Hillert, Bo Jansson, Bo Sundman and John Ågren, *A Two-Sublattice Model for Molten Solutions with Different Tendency for Ionization,* in Met. Trans. A, **16A** (1985) 261-266.

[33] S Hertzman and B Sundman, *A Thermodynamic analysis of the Fe-Cr system* Calphad **6** (1982) 67–80

[34] R Kikuchi, *A theory of cooperative phenomena*, Phys Rev B **81** (1951) 988

[35] M Temkin, *Mixtures of Fused Salts as Ionic Solutions* Acta Physiochimica U.R.S.S **20**, (1945), 411–420

Appendix A GTP version, dimensioning and constants

Most arrays are dimensioned in the GTP package using constants. If one need to change them it is usually only in one place. Most of these constants are used in the init_gtp subroutine but also when temporary arrays are needed inside some other subroutines.

These declarations are now in the separate module models/ocparam.F90

```
!-----------------------------------------------------------------------
! Version numbers
!-----------------------------------------------------------------------
! version number of GTP (not OC)
character*8, parameter :: gtpversion='GTP-3.30'
! THIS MUST BE CHANGED WHENEVER THE UNFORMATTED FILE FORMAT CHANGES!!!
character*8, parameter :: savefile='OCF-3.20'
!
!-----------------------------------------------------------------------
!
! Parameters defining the size of arrays etc.
! max elements, species, phases, sublattices, constituents (ideal phase)
! NOTE increasing maxph to 600 and maxtpf to 80*maxph made the equilibrium
! record very big and created problems storing equilibria at STEP/MAP!!!
integer, parameter :: maxel=100,maxsp=1000,maxph=600,maxsubl=10,maxconst=1000
! maximum number of constituents in non-ideal phase
integer, parameter :: maxcons2=300
! maximum number of elements in a species
integer, parameter :: maxspel=10
! maximum number of references
integer, parameter :: maxrefs=1000
! maximum number of equilibria
integer, parameter :: maxeq=900
! some dp values, default precision of Y and default minimum value of Y
! zero and one set in tpfun
double precision, parameter :: YPRECD=1.0D-6,YMIND=1.0D-30
! dimension for push/pop in calcg, max composition dependent interaction
integer, parameter :: maxpp=1000,maxinter=3
! max number of TP symbols, TOO BIG VALUE MAKES SAVE AT STEP/MAP DIFFICULT
integer, parameter :: maxtpf=20*maxph
!  integer, private, parameter :: maxtpf=80*maxph
! max number of properties (G, TC, BMAG MQ%(...) etc)
integer, parameter :: maxprop=50
! max number of state variable functions
integer, parameter :: maxsvfun=500

double precision, parameter :: zero=0.0D0,one=1.0D0,two=2.0D0,ten=1.D1
```

```
!------------------------------------------------------------------------
! Numerical parameters
!------------------------------------------------------------------------
integer, parameter :: default_splitsolver = 0
! 1 to allow to split the linear system when conditions leads to square matrix
integer, parameter :: default_precondsolver = 0
! 1 to allow to use a Jacobi preconditionner for solving the linear system
!
!------------------------------------------------------------------------
! convergence criteria used in matsmin.F90
!------------------------------------------------------------------------
!
!!!!!!!
!! meq_phaseset subroutine
!!!!!!!
integer, parameter :: default_nochange = 4
! minimum number of iterations between a change of the set of stable phases
! should not be smaller than default_minadd/default_minrem
integer, parameter :: default_minadd=4
integer, parameter :: default_minrem=4
!
double precision, parameter :: default_addchargedphase = 1.D-2
!Used to verify: charge(phase) > 1.0D-2
!That is, checking that the phase to be added does not have a net charge
!
!!!!!!!
!! meq_sameset subroutine
!!!!!!!
integer, parameter :: default_typechangephaseamount = 0
! By default, 0 leads to default_scalechangephaseamount=1.0
! 1 leads to default_scalechangephaseamount=sum of prescribed conditions -/+ 1
! 2 leads to default_scalechangephaseamount=max of (1, max of prescribed conditions)

double precision, parameter :: default_scalechangephaseamount = 1.0
! scale all changes in phase amount with total number of atoms. By default,
! assume this is unity.

double precision, parameter :: default_ylow = 1.D-3
! parameter added to avoid too drastic jumps in small site fractions
! normalizing factor, if y < ylow ....
!
double precision, parameter :: default_ymin = 1.D-12
! parameter added to avoid too drastic jumps in small site fractions
!
double precision, parameter :: default_ymingas = 1.D-30
! parameter added for gases, since the phase one must allow smaller constituent fractions
! normalizing factor, if y < critymingas then y = cirtymingas
!
double precision, parameter :: default_ionliqyfact = 3.D-1
! this is an emergecy fix to improve convergence for ionic liquid
! correction to site fractions in ionic liquids
!
double precision, parameter :: default_deltaTycond=2.5D1
! this is set each time the set of phases changes, controls change in T
! when there is a condition on y
!
integer, parameter :: default_nophasechange = 100
!Criterion on the maximum number of iterations that should go by with no change in the set of p
! That is, the system should have at least one phase change every default_nophasechange iterati
!
double precision, parameter :: default_maxphaseamountchange = 1.0E-10
!Criterion on the minimum of amount of phase change (DeltaN) vis-a-vis slow convergence
! That is, if the set of stable phases doesn't change, and the change in stable phases is lower
! default_maxphaseamountchange, then this is considered a 'slow convergence case'
!
double precision, parameter :: default_bigvalues = 1.0D+50
!Criterion on the maximum element of smat matrix
! Most probably, if something in smat is bigger than default_bigvalues, a calculation error has
!
double precision, parameter :: default_minimalchangesT = 1.0D-2
! minimal change in Temperature allowed when Temperature is variable
!
double precision, parameter :: default_limitchangesT = 0.2D0
double precision, parameter :: default_deltaT = 1.0D1
!modified xconv criterion CHECK
!Used to verify:  DeltaT > default_delatT*%xconv (converged = 8)
!Case where T is variable
!
double precision, parameter :: default_limitchangesP = 0.2D0
double precision, parameter :: default_deltaP = 1.0D4
!modified xconv criterion CHECK
!Used to verify:   DeltaP > default_deltaP*%xconv (converged = 8)
!Case where P is variable
!
double precision, parameter :: default_minimalchangesP = 1.0D-2
! minimal change in Pressure allowed when Pressure is variable
!
double precision, parameter :: default_chargefact = 1.0
! term added to the correction in site fraction due to electric charge
!
integer, parameter :: default_noremove=3
!Criterion on the minimum number of iterations with
```

```
! N-DeltaN<0  Before removing the phase in question
! That is, the phase must have a negative quantity for default_noremove iterations
! before being removed
!
double precision, parameter :: default_yvar1 = 1.0D-4
! first limitation to change in site fraction
!
double precision, parameter :: default_yvar2 = 1.0D-13
! second limitation to change in site fraction
!
double precision, parameter :: default_upperyvar1 = 1.0D-3
! limitation to change in site fraction
! normalizing factor, if yvar1 > default_upperyvar1 ....
!     then yvar1 =  default_upperyvar1
!
double precision, parameter :: default_upperyvar2 = 1.0D-13
! limitation to change in site fraction
! normalizing factor, if yvar2 > default_upperyvar2 ....
!             then yvar2 =  default_upperyvar2
!
double precision, parameter :: default_correctionfactorYS = 1.0D1
!multiplier in a criterion
!Used to verify:
!|Delta y(phase, constituent)(recursion =k)| > default_correctionfactorYS*|Delta y(phase, const
!
double precision, parameter :: default_correctionfactorXCONV = 1.0D2
!multiplier in a criterion CHECK
!Used to verify: In an unstable phase:
!   Delta y(phase, constituent) > default_correctionfactorXCONV*%xconv(converged = 4)
!
double precision, parameter :: default_correctionfactorDGM = 1.0
!default_correctionfactorDGM criterion
!Used to verify:
!   dgm(recursion=k) - dgm(recursion=k-1) > default_correctionfactorDGM *gdconv(1) (converged =
!Case where more than 10 constituents in the phases are present, (apparently) warranting a bigg
!
double precision, parameter :: default_upperycormax2 = 1.0D-4
! check on max stepsize, determining whether or not it is too small
!
integer, parameter :: default_minimaliterations = 4
!Criterion on the minimum number of iterations for the code as a whole
!
!!!!!!!
!! userif/pmon6.F90, gtp3A.F90 Fortran files
!!!!!!!
double precision, parameter :: default_maxiter = 500
```

```
! default maximum number of iteration
!
double precision, parameter :: default_xconv = 1.D-6
double precision, parameter :: default_minxconv = 1.D-30
! default and minimal values for ceq%xconv criterion
!
double precision, parameter :: default_mingdconv = 1.D-5
double precision, parameter :: default_gdconv1 = 4.D-3
double precision, parameter :: default_gdconv2 = 0.D0
! default and  minimal value for ceq%gdconv(1) criterion
!
double precision, parameter :: default_mingridmin = -1.D-2
! minimal value for ceq%gmindif criterion
!

!----------------------------------------------------------------------
! Physical parameters
!----------------------------------------------------------------------
double precision, parameter :: PI = 3.141592653589793D0
```

Appendix B Declaration of model parameter identifiers

This is the declaration of the model parameter identifiers in the source code. See also Table 4. It is used in subroutine init_gtp.

```
! Model parameter identifiers entered in gtp3A.F90 and used mainly in gtp3H
! to calculate additions.  Used also in gtp3B when entering parameters
! Index Name Used in addition/model
!  1     G    Gibbs energy for endmembers or interactions
!  2     TC   Curie T in Inden-Hillert magnetic model
!  3 BMAG  - -                                    1 Average Bohr magneton numb
!  4 CTA   - P                                    2 Curie temperature
!  5 NTA   - P                                    2 Neel temperature
!  6 IBM   - P &<constituent#sublattice>;        12 Individual Bohr magneton num
!  7 LNTH  - P                                    2 ln(Einstein temperature)
!  8 V0    - -                                    1 Volume at T0, P0
!  9 VA    T -                                    4 Thermal expansion
! 10 VB    T P                                    0 Bulk modulus
! 11 VC    T P                                    0 Alternative volume parameter
! 12 VS    T P                                    0 Diffusion volume parameter
! 13 MQ    T P &<constituent#sublattice>;        10 Mobility activation energy
! 14 MF    T P &<constituent#sublattice>;        10 RT*ln(mobility freq.fact.)
! 15 MG    T P &<constituent#sublattice>;        10 Magnetic mobility factor
! 16 G2    T P                                    0 Liquid two state parameter
! 17 THT2  - P                                    2 Smooth step function Tcrit
! 18 DCP2  - P                                    2 Smooth step function increm.
! 19 LPX   T P                                    0 Lattice param X axis
! 20 LPY   T P                                    0 Lattice param Y axis
! 21 LPZ   T P                                    0 Lattice param Z axis
! 22 LPTH  T P                                    0 Lattice angle TH
! 23 EC11  T P                                    0 Elastic const C11
! 24 EC12  T P                                    0 Elastic const C12
! 25 EC44  T P                                    0 Elastic const C44
! 26 UQT   T P &<constituent#sublattice>;        10 UNIQUAC residual parameter
! 27 RHO   T P                                    0 Electric resistivity
! 28 VISC  T P                                    0 Viscosity
! 29 LAMB  T P                                    0 Thermal conductivity
! 30 HMVA  T P                                    0 Enthalpy of vacancy form.
! 31 TSCH  - P                                    2 Schottky anomaly T
! 32 CSCH  - P                                    2 Schottky anomaly Cp/R.
! 33 QCZ   - -                                    1 MQMQA cluster coord factor
! DO NOT CHANGE THE ORDER in gtp3A, that would require changes elsewhere too
! The table below is the current definition of model parameters!!
  character (len=4), dimension(40) :: MODPARID=&
```

```
       ['G   ','TC  ','BMAG','CTA ','NTA ','IBM ','LNTH','V0  ','VA  ','VB  ',&
        'VC  ','VS  ','MQ  ','MF  ','MG  ','G2  ','THT2','DCP2','LPX ','LPY ',&
        'LPZ ','LPTH','EC11','EC12','EC44','UQT ','RHO ','VISC','LAMB','HMVA',&
        'TSCH','CSCH','QCZ ','    ','    ','    ','    ','    ','    ','    ']
!         1      2      3      4      5      6      7      8      9      10
! The meaning of the model parameters is entered in init_gtp in gtp3A.F90
```

Appendix C Iso-C declaration of OC data structures for C++

This is the file OC-isoC.h that can be used in C++ program to access data structures inside the OC software.

I would like to have some explanaton of this because I do not understand C++.

```
#if !defined __OCASI__
#define __OCASI__

/* Modification history
160829 Bo Sundman Update
2015-2016 Matthias Stratmann and Cristophe Sigli Modifications
2014 Teslos? First version

This contains the structure of TYPE variables in OC needed for the OC/TQ OCASI interface

NOTE there is also a c_gtp_equilibrium_data structure defined in liboctqisoc.F90 */

typedef struct {
  int forcenewcalc;
  double tpused[2];
  double results[6];
} tpfun_parres;

typedef struct {
  int splink, phlink, status;
  char refstate[16];
  int *endmember;
  double tpref[2];
  double chempot[2];
  double mass, molat;
} gtp_components;

typedef struct {
  int lokph, compset, ixphase, lokvares, nextcs;
} gtp_phasetuple;

typedef struct {
  int statevarid, norm, unit, phref, argtyp;
  int phase, compset, component, constituent;
  double coeff;
  int oldstv;
} gtp_state_variable;
```

```
typedef struct {
  int latd, ndd, tnoofxfr, tnoofyfr, varreslink, totdis;
  char id;
  double *dsites;
  int *nooffr;
  int *splink;
  int *y2x;
  double *dxidyj;
  double fsites;
} gtp_fraction_set;

//struct gtp_fraction_set;

typedef struct {
  int nextfree, phlink, status2, phstate,phtupx;
  double abnorm[3];
  char prefix[4], suffix[4];
  int *constat;
  double *yfr;
  double *mmyfr;
  double *sites;
  double *dpqdy;
  double *d2pqdvay;
  //struct gtp_fraction_set disfra;
  double amfu, netcharge, dgm;
  int nprop;
  int *listprop;
  double **gval;
  double ***dgval;
  double **d2gval;
  double curlat[3][3];
  double **cinvy;
  double *cxmol;
  double **cdxmol;
  double *addg;
} gtp_phase_varres;

typedef struct gtp_condition {
  int noofterms, statev, active, iunit, nid, iref, seqz, experimenttype;
  int symlink1, symlink2;
  int **indices;
  double *condcoeff;
  double *prescribed, current, uncertainity;
  // should this be a struct ??
  gtp_state_variable *statvar;
  struct gtp_condition *next, *previous;
} gtp_condition;

typedef struct {
  int status, multiuse, eqno, next;
  char eqname[24], comment[72];
  double tpval[2], rtn;
  double weight;
  double *svfunres;
  gtp_condition *lastcondition, *lastexperiment;
  gtp_components *complist;
  double **compstoi, **invcompstoi;
  gtp_phase_varres *phase_varres;
  tpfun_parres *eq_tpres;
  double *cmuval;
  double xconv;
  double gmindif;
  int maxiter;
  char eqextra[80];
  int sysmatdim, nfixmu, nfixph;
  int *fixmu;
  int *fixph;
  double **savesysmat;
} gtp_equilibrium_data;

#endif
```

Appendix D Implementation of some ternary extrapolation methods

In OC the Redlich-Kister extrapolation method is used for all binary interactions and by default the Muggianu extrapolation method for ternary and higher order extrapolations.

It is possible to set other ternary extrapolations methods, specific for each ternary, as explained in this appendix. It is fragile.

The command to specify a Kohler or Toop extrapolation method is explained in section 2.3.2.5. In a TBD file this command is not yet implemented.

D.1 The Kohler-Toop extrapolation methods in OC

For the Toop and Kohler extrapolation methods we must calculate a reduced set of constituent fraction variables and below we explain how this is done for ξ_{ij}, ξ_{ji} and a quotient σ_{ij} are obtained using eqs. (18 and (19 (repeated here):

$$\xi_{ij} = y_i + \sum_{k \in ijk} y_k \quad \text{(D1)}$$

$$\sigma_{ij} = \sigma_{\text{ji}} = 1 - \sum_{m \in ijk} y_m \quad \text{(D2)}$$

where the summation is over k in all in ternary subsystems A with constituents $i - j - k$ in which j is an asymmetric (i.e. Toop) constituent and the summation over m is for all subsystems where $i - j$ is a Kohler extraplation form m.

The reduced fractions ξ_{ij} are used to calculate conribution to the Gibbs energy for the binary interaction between $i - j$ in the the symmetric Redlisch-Kister series as:

$$^{E}G_{M}^{i-j \text{ bin}} = y_i y_j \sum_{\nu=0}^{n} (\frac{\xi_{ij} - \xi_{ji}}{\sigma_{ij}})^{\nu} \cdot \, ^{\nu}L_{ij} \quad \text{(D3)}$$

Note that the composition independent term, $^{0}L_{ij}$ is independent of the extrapolation method. For a Muggianu extrapolation $\xi_{ij} = y_i, \ \xi_{ji} = y_j$ and $\sigma = 1$ which does not require any extra data structures.

D.2 Using other binary extrapolation methods than Muggianu

Each interaction record has **gtp_tooprec** pointer and at present it is used only for Toop/Kohler ternary extrapolations. If this pointer is unassigned the composition dependence of the binary parameter is calculated according to a Muggianu model, i.e. using the multicomponent fractions of the two constituents indicated by the binary interaction.

If the pointer is associated the fractions used to calculate the composition dependence of the binary parameter L_{ij} will be calculated according to eqs. 18-19 using a list of gtp_tooprec records linked from the binary $i-j$ interaction. The values of xi_{ij}, ξ_{ji} and σ_{ij} calculated using the list of gtp_tooprec records are then used in eq. D3 to calculate the contribution to the excess Gibbs energy from the $i-j$ interaction for the phase.

To calculate ξ_{ij} for a particular binary a list of records with all ternaries with a Kohler or Toop extrapolation must be created and linked from the binary interaction records involved. Calculating first and second derivatives with respect to the proper fraction variables must also be included.

D.3 Creating the extrapolation datastructure

A new record type **TYPE gtp_tooprec** is created by the command AMEND PHASE < *name* > TERNARY_EXTRAPOL as explained in section 2.3.2.5. This record will be linked from the 3 binary interaction parameter records indicated by the 3 constituents A, B and C specified in the command, A-B, A-C and B-C. If one or more of these binary interaction parameters are zero or missing no link will be created for this binary interaction. If a missing binary interaction is added after the AMEND command is given, the calculation will give wrong results. Thus the AMEND $\cdots$ TERNARY_EXTRAPOL command must be given after all non-zero binary interactions has been entered.

A binary interaction will normally be involved in several ternary extrapolations and each such AMEND command will add a new record to the list for the binary systems involved. The algorithm to add a new record for an A-B-C ternary extrapolation involve the steps described below. (Note that a significant part of the algorithm is to handle user input errors.) It also attempts to prevent entering duplicates of the ternary extrapolation method for the same A-B-C system. The algorithm can be used even if one or two of the binaries involved does not have any binary interaction.

1. The first step is to order the constitents in alphabetical order, A-B-C and set an indicator which is the Toop constituent (if any).
   (a) Loop to find an end member with the constituent A. The end members are always in alphabetical order and any binary interaction involving A will be linked from this endmember. If the endmember is found a pointer is set to the interaction record of A. If this pointer is associated (i.e. points to a record) continune with step 2.
   (b) Continue to find an endmember for B. If it is found set a pointer to the interaction record of B but if it is not associated exit with an error code.
2. Here we have an interaction list which may contain a binary A-B, A-C or B-C parameter. First create a new method record and insert the constituent indices for A, B and C, set the possible Toop constituent index and nullify the next12, next13 and next23 pointers (these are links needed to sum up fractions involved to calculate the fractions to be used for each parameter).
3. Loop the interaction records until we find an interaction with B or C (the interaction records are not ordered alphabetically). Check if this record already has a ternary method list and if so goto step 4. If not link the new method record from the interaction parameter and if we have not found all interactions for A go to step 2, otherwise to step 1b.

4. Check if any method record in this list has the constituents A-B-C, i.e. if the method is a duplicate. If so give an error message, delete the method record and exit. Otherwise add the new method record at the end of the list linking it from the last previous method record using the appropriate nextij link. If we have not found all interactions for A go to step 2, otherwise to step 1b.

5. Loop the interaction records for B until we find an interaction with C, otherwise exit.

This is coded in the subroutine add_ternary_etrapol_method in the gtp3H.F90 file. An extrapolation for a specific ternary can only be given once, if it is given a second time the results will be unpredictable.

D.4 Calculating fractions using the extrapolation datastructure

When a binary record ij has an associated tooprec pointer the algorithm below, based on eqs. 18-19:

$$\xi_{ij} = y_i + \sum_{k \in ijk} y_k \tag{D4}$$

$$\sigma_{ij} = \sigma_{\mathrm{ji}} = 1 - \sum_{m \in ijk} y_m \tag{D5}$$

is used to calculate ξ_{ij} and σ_{ij}. The rule is:
- y_k is added if j is a Toop element in $i - j - k$ and
- y_m is subtracted if $i - j$ has Kohler method in $i - j - m$.
An elaborate example is listed in Table D1.

1. Set $\xi_{ij} = y_i$; $\xi_{ji} = y_j$ and $\sigma_{ij} = 1$. Set pointer next to first gtp_tooprec record.

2. The current binary $(i-j)$ is a subset of the ternary $(k-l-m)$ in the gtp_tooprec record with a possible Toop constituent τ. The constituents $k < l < m$ are always in ascending order.

3. If no Toop constituent ($\tau \equiv 0$) then
   $\sigma_{ij} = \sigma_{ij} - y_{m \neq (i,j)}$; go to step 5.

4. The record is part of 3 lists for the $k - l, k - m$ and $l - m$ binaries. We have to select the next record accordingly
   If $j \equiv \tau$ then $\xi_{ij} = \xi_{ij} + y_i$;
   . else if $i \equiv \tau$ then $\xi_{ji} = \xi_{ji} + y_j$;
   . else $\sigma_{ij} = \sigma_{ij} - y_\tau$; endif

5. Select next gtp_tooprec:
   If $i \equiv k$ then
   . if $j \equiv l$ then set pointer next=>next12;
   . else set pointer next=>next13; endif
   else set pointer next=>next23; endif
   if next is associated go to step 2.

6. We have now calculated the fractions ξ_{ij}, ξ_{ji} and σ_{ij} needed to to calculate the interaction $i-j$ in equation eq. D3. Note the first and second partial derivatives with respect to the relevant y fractions must be calculated also!

Here is an example how the ξ_{ij} should be calculated based on the Table D1 below.

Table D1: A case with Toop element 2 in (1-2-3), 1 in (1-2-4), (1-3-4) and (1-4-5), element 5 in (2-3-5) and element 3 in (3-4-5). In this example there are two Toop elements in (2-3-4) which has not been implemented in OC. If the ternary has a Toop element that is given inside parenthesis, a symmetric ternary is denoted (-). This table is based on an example presented in [21]. The algorithm to calculate ξ_{ij} etc. is described in section D.4.

i-j	ternary	ternary	ternary	ξ_{ij}	ξ_{ji}	σ_{ij}
1-2	1-2-3(2)	1-2-4(1)	1-2-5(-)	$\xi_{12} = y_1 + y_3$	$\xi_{21} = y_2 + y_4$	$1 - y_5$
1-3	1-2-3(2)	1-3-4(1)	1-3-5(-)	$\xi_{13} = y_1$	$\xi_{31} = y_3 + y_4$	1
1-4	1-2-4(1)	1-3-4(1)	1-4-5(1)	$\xi_{14} = y_1$	$\xi_{41} = y_4 + y_2 + y_3 + y_5$	1
1-5	1-2-5(-)	1-3-5(-)	1-4-5(1)	$\xi_{15} = y_1$	$\xi_{51} = y_5 + y_4$	$1 - y_2$
2-3	1-2-3(2)	2-3-4(3,4)	2-3-5(5)		not implemented	
2-4	1-2-4(1)	2-3-4(3,4)	2-4-5(-)		not implemented	
2-5	1-2-5(-)	2-3-5(5)	2-4-5(-)	$\xi_{25} = y_2$	$\xi_{52} = y_5$	$1 - y_1$
3-4	1-3-4(1)	2-3-4(3,4)	3-4-5(3)		not implemented	
3-5	1-3-5(-)	2-3-5(5)	3-4-5(3)	$\xi_{35} = y_3 + y_2$??	$\xi_{53} = y_5 + y_4$	1
4-5	1-4-5(1)	2-4-5(-)	3-4-5(3)	$\xi_{45} = y_4$	$\xi_{54} = y_5$	$1 - y_1 - y_2$

The items below is just for check of Table D1.

1. 1-2 binary:
   (a) $\xi_{12} = y_1$ always as first term, $\sigma_{ij} = 1$.
      i. In ternary 1-2-3: Toop 2 is second index of ξ_{12}, add y_3
      ii. In ternary 1-2-4: ignore Toop 1 as first index of ξ_{12}
      iii. In ternary 1-2-5: no Toop constituent

      $\xi_{12} = y_1 + y_2$ is final reduced fraction, OK.
   (b) $\xi_{21} = y_2$ always as first term
      i. In ternary 1-2-3 ignore Toop 2 as first index of ξ_{21}
      ii. In ternary 1-2-4 Toop 1 is second index of ξ_{21}, add y_4
      iii. In ternary 1-2-5 no Toop constituent

      $\xi_{21} = y_2 + y_4$ is final reduced fraction, OK.
2. 1-4 binary:
   (a) $\xi_{14} = y_1$ always as first term, $\sigma_{ij} =$?.
      i. In ternary 1-2-4 ignore Toop 1 as first index of ξ_{14}

ii. In ternary 1-3-4 ignore Toop 1 as first index of ξ_{14}

iii. In ternary 1-4-5 ignore Toop 1 as first index of ξ_{14}

$\xi_{14} = y_1$ is final reduced fraction, OK

(b) $\xi_{41} = y_4$ always as first term

i. In ternary 1-2-4 Toop 1 is second index of ξ_{41}, add y_2

ii. In ternary 1-3-4 Toop 1 is second index of ξ_{41}, add y_3

iii. In ternary 1-4-5 Toop 1 is second index of ξ_{41}, add y_5

$\xi_{41} = y_4 + y_2 + y_3 + y_5$ is final reduced fraction, OK

3. 2-5 binary:

(a) $\xi_{25} = y_2$ always as first term, $\sigma_{ij} =?$.

i. In ternary 1-2-5 no Toop constituent

ii. In ternary 2-3-5 Toop 5

iii. In ternary 2-4-5

$\xi_{24} = y_2 + y_3$ is final reduced fraction, OK

(b) $\xi_{42} = y_4$ always as first term

i. In ternary 1-2-4 Toop 1 is not in ξ_{24}, add y_4

ii. In ternary 2-3-4 no Toop constituent

iii. In ternary 2-4-5 no Toop constituent

$\xi_{42} = y_4$ is final reduced fraction.

4. 3-4 binary:

(a) $\xi_{34} = y_4$ always as first term, $\sigma_{ij} =?$.

i. In ternary 1-3-4 ignore Toop 1 as not an index of ξ_{34}

ii. In ternary 2-3-4 ignore Toop 4 as not an index of ξ_{34}

iii. In ternary 3-4-5 ignore Toop 3 as first index of ξ_{34}

$\xi_{34} = y_3$ is final reduced fraction, OK

(b) $\xi_{43} = y_4$ always as first term

i. In ternary 1-3-4 ignore Toop 1 as not an index of ξ_{34}

ii. In ternary 2-3-4 no Toop constituent

iii. In ternary 3-4-5 Toop 3 as second index of ξ_{43}, add y_5

$\xi_{43} = y_4 + y_5$ is final reduced fraction, OK

5. 3-5 binary:

(a) $\xi_{35} = y_3$ always as first term, $\sigma_{ij} =?$.

i. In ternary 1-3-5 nothing

ii. In ternary 2-3-5 nothing

iii. In ternary 3-4-5 nothing

$\xi_{35} = y_3$, , error in Table D1??

(b) $\xi_{53} = y_5$ always as first term

i. In ternary 1-3-5 nothing

ii. In ternary 2-3-5 nothing

iii. In ternary 3-4-5 constituent 3 is a Toop element, add y_4

$\xi_{53} = y_5 + y_4$

6. and so on ....

D.5 Calculating derivatives of G using Toop-Kohler excess models

The minimization algorithm requires first and second derivatives of the Gibbs energy and whereas that is straightforward using the RKM excess parameters described in section 2.3.2.1, it becomes more complicated with a Toop or Kohler excess parameter because such a parameter depend on other fractions than used for the binary parameter itself.

For a binary parameter L_{ij} with a Kohler model in a multicomponent system the contribution to the Gibbs energy from

$$L_{ij} = \sum_{\nu=0}^{n} \left(\frac{y_i - y_j}{\sigma_{ij}}\right)^{\nu} \cdot \ {}^{\nu}L_{ij} \quad = \quad \sum_{\nu=0}^{n} \left(\frac{y_i - y_j}{\sum_k y_k}\right)^{\nu} \cdot \ {}^{\nu}L_{ij} = \sum_{\nu=0}^{n} z^{\nu} \cdot \ {}^{\nu}L_{ij} \tag{D6}$$

where

$$z \quad = \quad \frac{y_i - y_j}{\sigma_{ij}} = \frac{y_i - y_j}{\sum_k y_k} \tag{D7}$$

where the summation over k are for all elements **not** included in σ in eq. D5. Note that for a simple $i - j - k$ ternary Kohler method we have $\sigma_{ij} = 1 - y_k = y_i + y_j$. Eqs. D6 and D7 are valid also for Toop methods if y_i is replaced by ξ from eq. D4.

The first derivatives of z with respect to the different fractions are:

$$\frac{\partial}{\partial y_i} z^n \quad = \quad n\frac{z^{n-1}}{\sigma_{ij}} \tag{D8}$$

$$\frac{\partial}{\partial y_j} z^n \quad = \quad -n\frac{z^{n-1}}{\sigma_{ij}} \tag{D9}$$

$$\frac{\partial}{\partial y_k} z^n \quad = \quad -n\frac{z^{n}}{\sigma_{ij}} \tag{D10}$$

As a check consider calculating the derivative of $z_1 = \frac{y_i - y_j}{\sigma_{ij}}$ with $\sigma_{ij} = y_i + y_j$ by combining eqs. D8 and D10:

$$\frac{\partial z_1^n}{\partial y_i} \quad = \quad n\frac{z_1^{n-1}}{\sigma_{ij}} - n\frac{z_1^{n}}{\sigma_{ij}} = n\frac{z_1^{n-1}}{\sigma_{ij}}(1 - (y_i - y_j)) = n\frac{z_1^{n-1}}{\sigma_{ij}}(y_i + y_j - y_i + y_j) = 2ny_j\frac{z_1^{n-1}}{\sigma_{ij}}$$

which is the correct result. The second derivatives are:

$$\frac{\partial^2}{\partial y_i^2} z^n = n(n-1)\frac{z^{n-2}}{\sigma_{ij}^2} \tag{D11}$$

$$\frac{\partial^2}{\partial y_i y_j} z^n = -n(n-1)\frac{z^{n-2}}{\sigma_{ij}^2} \tag{D12}$$

$$\frac{\partial^2}{\partial y_i y_k} z^n = -n^2\frac{z^{n-1}}{\sigma_{ij}^2} \tag{D13}$$

$$\frac{\partial^2}{\partial y_j^2} z^n = n(n-1)\frac{z^{n-2}}{\sigma_{ij}^2} \tag{D14}$$

$$\frac{\partial^2}{\partial y_j y_k} z^n = n^2\frac{z^{n-1}}{\sigma_{ij}^2} \tag{D15}$$

$$\frac{\partial^2}{\partial y_{k_1} y_{K_2}} z^n = n(n+1)\frac{z^{n}}{\sigma_{ij}^2} \tag{D16}$$

which are implemented in the specific subroutine calc_toop called from cgint in the gtp3X.F90 file decribed in section 8.16.3.

Appendix E The quasichemical configurational entropy with modifications

CVM considers mixing of clusters of two or more atoms in the configurational entropy and a general method to calculate the configurational entropy for clusters in all kinds of crystalline structures was derived by Kikuchi [34]. Such clusters share surfaces, edges and points (where the atoms are situated) and in the configurational entropy all clusters must agree on the atom placed in each point.

In CVM both LRO and SRO are described using clusters but in order to describe liquids, where there is no LRO, some modifications are needed.

E.1 The crystalline quasichemical model with LRO

Starting with the classical quasichemical model for a binary A-B system in a crystalline BCC lattice we have 4 clusters with bond pairs AA, AB, BA and BB. The clusters AB and BA represent the two cases when A is on the first sublattice (denoted ') and B on the second (denoted ") and vice versa. When we have LRO they will be different. The configurational entropy is from Kikuchi [34]:

$$\begin{aligned} {}^{qch}S_m/R &= -\frac{z}{2}(p_{\mathrm{AA}}\ln(p_{\mathrm{AA}}) + p_{\mathrm{AB}}\ln(p_{\mathrm{AB}}) + p_{\mathrm{BA}}\ln(p_{\mathrm{BA}}) + p_{\mathrm{BB}}\ln(p_{\mathrm{BB}})) + \\ & (z-1)(x_{\mathrm{A}}\ln(x_{\mathrm{A}}) + x_{\mathrm{A}}\ln(x_{\mathrm{A}})) \end{aligned} \quad \text{(E1)}$$

where $z = 8$ is the number of bonds between the sublattices, p_{AB} represent the probability of a cluster AB with A on the first sublattice and B on the second, p_{BA} the probability of a cluster with B on the first sublattice and A on the second. We must treat the two clusters p_{AB} and p_{BA} as different because when we have LRO the fraction of A and B on the two sublattices will be different. The sublattice fractions can be calculated from the cluster probablilities:

$$\begin{aligned} y'_{\mathrm{A}} &= p_{\mathrm{AA}} + p_{\mathrm{AB}} \\ y'_{\mathrm{B}} &= p_{\mathrm{BB}} + p_{\mathrm{BA}} \\ y''_{\mathrm{A}} &= p_{\mathrm{AA}} + p_{\mathrm{BA}} \\ y''_{\mathrm{B}} &= p_{\mathrm{BB}} + p_{\mathrm{AB}} \end{aligned} \quad \text{(E2)}$$

where y'_{A} and y''_{B} are the constituent fractions of A on the first sublattice and second sublattice respectively. Introducing ϵ as a measure of SRO we have also:

$$\begin{aligned} p_{\mathrm{AA}} &= y'_{\mathrm{A}}y''_{\mathrm{A}} - \epsilon \\ p_{\mathrm{AB}} &= y'_{\mathrm{A}}y''_{\mathrm{B}} + \epsilon \\ p_{\mathrm{BA}} &= y'_{\mathrm{B}}y''_{\mathrm{A}} + \epsilon \\ p_{\mathrm{BB}} &= y'_{\mathrm{B}}y''_{\mathrm{B}} - \epsilon \end{aligned} \quad \text{(E3)}$$

where a nonzero ϵ represent SRO. Finally the mole fractions, x_{A} and x_{B} are:

$$\begin{aligned} x_{\mathrm{A}} &= y'_{\mathrm{A}} + y''_{\mathrm{A}} = 2p_{\mathrm{AA}} + p_{\mathrm{AB}} + p_{\mathrm{BA}} \quad \text{(E4)} \\ x_{\mathrm{B}} &= y'_{\mathrm{B}} + y''_{\mathrm{B}} = 2p_{\mathrm{BB}} + p_{\mathrm{AB}} + p_{\mathrm{BA}} \end{aligned}$$

This model has the endmember parameters for $^\circ G_{\mathrm{AB}} = {}^\circ G_{\mathrm{BA}}$ which must be negative to promote ordering. A positive value would promote phase separation, i.e. a miscibility gap.

Without SRO, i.e. $\epsilon = 0$, and inserting eq. E3 in eq. E1 gives the correct LRO configurational entropy in the BCC lattice:

$$^{LRO}S_m/R \quad = \quad -(y'_{\mathrm{A}}\ln(y'_{\mathrm{A}}) + y'_{\mathrm{B}}\ln(y'_{\mathrm{B}}) + y''_{\mathrm{A}}\ln(y''_{\mathrm{A}}) + y''_{\mathrm{A}}\ln(y''_{\mathrm{A}})) \qquad \text{(E5)}$$

because we have one site on each sublattice and this is identical to a CEF model. The value of ϵ has very little contribution to the entropy when we have LRO.

If $^\circ G_{\mathrm{AB}}$ is slightly negative we will have SRO i.e. $\epsilon \neq 0$ but no LRO. This means $p_{\mathrm{AB}} = p_{\mathrm{BA}}$ and $y'_{\mathrm{A}} = y''_{\mathrm{A}}$ and such a non-zero ϵ cannot be included in eq. E5 for the CEF model.

With more negative $^\circ G_{\mathrm{AB}}$ there will be an order/disorder transformation and in such a case we will have $p_{\mathrm{AB}} \neq p_{\mathrm{BA}}$ and also $y'_{\mathrm{A}} \neq y''_{\mathrm{A}}$.

If we ignore the LRO transition eq. E1 is valid only for small SRO, i.e. small ϵ as discussed by Hillert et al. [19]. In fact eq. E1 will give negative configurational entropies for strong SRO unless $z = 2$ which represent a linear chain where each atom has just 2 bonds.

Note also that the crystalline quasichemical model can only describe ordering at the equiatomic composition, to describe ordering at other compositions one must use larger clusters, for example tetrahedrons.

E.2 The quasichemical model without LRO

In order to describe short range order (SRO) in liquids where there is no LRO, Pelton et al. [22, 23, 24] have explored a version of the quasichemical model using only two bonds/atom, $z = 2$. Although it is quite unphysical to assume 2 bonds/atom it has been shown the model can describe experimental data and it provides reasonable extrapolations to higher order systems.

The MQMQA model includes a long range ordering (LRO) contribution, (called FNN) using a two-sublattice model where the cations occupies one sublattice and anions the other, similar to the Temkin model for molten salts [35] and the ionic two-sublattice model (I2SL) [32].

In this documentation we will just describe how the MQMQA model is implemented in Open-Calphad (OC) [1]. For a theoretical derivation see the original papers.

E.3 Some modeling concepts

In MQMQA there are two levels of ordering, one is called FNN (First Nearest Neigbor) between cations and anions (on different sublattices). This is usually very strong and can lead to Long Range Ordering (LRO) as described by sublattices in crystalline phases and in molten salts as a Temkin model. The Second Nearest Neighbor (SNN) ordering is between cations or anions on the same sublattice and is weaker. This can be summarized by model:

$$(A, B)(X, Y) \qquad \text{(E6)}$$

where A and B are cations and X and Y are anions on separate sites in the liquid. These sites are not defined in space but simply means that an anion may not occupy a place of a cation when considering the configurational entropy. Depending on the valency of the ions they can form several compunds, called quadrupoles, with different stoichiometric ratios. SNN means that two cations can share bonds with the same anion in a quadrupole and vice versa.

In the original papers the endmembers of a system such as eq. E6 are denoted A_2/X_2, B_2/X_2, B_2/X_2 and B_2/Y_2 which represent FNN (or LRO). In order handle SNN additional configurations such as AB/X_2, AB/Y_2, A_2/XY, B_2/XY and all reciprocal combinations AB/XY are also included. This notation is slightly confusing as the numbers used in the subscripts are not related to the actual stoichiometries. In this paper the notation AA/XX to AB/XY will be used.

The stoichiometric ratios of the constituents are determined requiring electroneutrality for the constituent representing FNN and by the composition for maximum SRO for the SNN. The MQMQA model has a parameter representing the coordination number for each element A in the liquid, called Z_{A} which varies with the composition of the liquid.

For a quadrupol constituent such as AB/XY there are 4 such coordination numbers, $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$, $Z^{\mathrm{B}}_{\mathrm{AB/XY}}$, $Z^{\mathrm{X}}_{\mathrm{AB/XY}}$ and $Z^{\mathrm{Y}}_{\mathrm{AB/XY}}$ for the 4 elements in the quadrupol. These coordination numbers are the inverse of the stoichiometry of the element in the quadropol. In OC these quadrupoles are entered as constituents and they mix on a single set of sites in the liquid. It is stated in Pelton et al [24] that one can formally treat the quadrupoles as real species. Implementing the MQM model in Thermo-Calc some 20 years ago species such as $\mathrm{Cu}_{2/1}$, $\mathrm{S}_{2/2}$ and $\mathrm{Cu}_{1/1}\mathrm{S}_{1/2}$ were used for the Cu-S system. But there is a problem with this discussed in section E.8.

Comparing with the I2SL model each cation and anion has an explicit charge and the electroneutrality is maintained by varying the number of sites on the sublattices for cations and anions. There can be neutral species on the anion sublattice and in the case with no cations only neutral constituents are allowed on anion sublattice. Vacancies with a composition dependent charge in the anion sublattice takes care of an excess of cations.

E.4 The constituent fraction

A summary of symbols used in this paper is given in table E1 together with the corresponding symbol in Pelton et al. [24].

In the original paper several sets of variables are used to denote amounts or fractions of different fractions, most notably n_{A} for the total number of moles of element A. The main equations in the paper by Pelton et al. [24] are summarized in Appendx E.10. There are several fraction variables needed to handle the configurational entropy expression eq. E33 and all of them are defined refering to the total amounts of the elements, n_{A}. However, in OC it is necessary to have a single set of constituent fraction variables which can be used to calculate the derivatives of the Gibbs energy depending on a single set of fraction variables from which the others can be obtained by simple relations. The fraction of the quadrupols, denoted $X_{\mathrm{AB/XY}}$ in [24], will be used for this and it will be denoted:

$$p_{\mathrm{AB/XY}} = p_{\mathrm{BA/XY}} = p_{\mathrm{AB/YX}} = p_{\mathrm{BA/YX}} \tag{E7}$$

where A and B are on the first sublattice and X and Y on the second but the order of the elements

Table E1: Symbols used here and in Pelton et al. [24]

Symbol		
OC	Pelton	Meaning
M_A	n_A	Moles of element A in the liquid
ϑ_A	X_A	Site fraction of element A on first sublattice.
ϑ_X	X_X	Site fraction of element X on second sublattice.
Z_A	Z_A	Composition dependant coordination number (bonds) for element A in the liquid.
$Z^\mathrm{A}_\mathrm{AB/XY}$	$Z^\mathrm{A}_\mathrm{AB/XY}$	Coordination number of element A in the quadrupole AB/XY. This is a constant.
$p_\mathrm{AB/XY}$, y_i	$X_\mathrm{AB/XY}$	Fraction of quadrupole AB/XY or i.
$\xi_{A/X}$	$X_\mathrm{A/X}$	Fraction of the endmember A/X.
w_A	Y_A	Coordination-equivalent site fraction for element A.

in each sublattice is irrelevant.

The sum of all quadrupol fractions is unity:

$$\sum_\mathrm{A}\sum_\mathrm{B\geq A}(\sum_\mathrm{X}\sum_\mathrm{Y\geq X} p_\mathrm{AB/XY}) = \sum_i y_i \quad = \quad 1 \tag{E8}$$

When the specific species are irrelevant the quadrupol fraction may be denoted as y_i in agreement with the fraction variables in eq. 1. One of the sublattices can have a single element and maybe empty if there is no SRO.

The constituents AB/XY represent a substitutional set and the fact that they are related to a sublattice model is mainly through one part of the expression of the configurational entropy in eq. E33. It means the set of constituents is similar to an associated model, see section 2.3.1.2, but in the configurational entropy there are factors which compensates for the exaggerated contribution to the entropy from an associated model. A complete model for a system (A,B)(X,Y) will contain 9 quadrupole fractions:

$$p_\mathrm{AA/XX},\ p_\mathrm{AA/XY},\ p_\mathrm{AA/YY},\ p_\mathrm{AB/XX},\ p_\mathrm{AB/XY},\ p_\mathrm{AB/YY},\ p_\mathrm{BB/XX},\ p_\mathrm{BB/XY},\ p_\mathrm{BB/YY} \tag{E9}$$

where

$$p_\mathrm{AA/XX} + p_\mathrm{AA/XY} + p_\mathrm{AA/YY} + p_\mathrm{AB/XX} + p_\mathrm{AB/XY} + p_\mathrm{AB/YY} + p_\mathrm{BB/XX} + p_\mathrm{BB/XY} + p_\mathrm{BB/YY} = 1 \tag{E10}$$

as they also represent probabilities.

The stoichiometry of a quadrupole is related to the charge balance and it should be neutral. This is discussed in the original paper by Pelton et al [24].

For each quadrupole there is a Gibbs energy of formation, $^\circ G_\mathrm{AB/XY}$ relative to the reference state of the elements. In Pelton et al. [24] these are explained in relation to chemical reactions between different quadrupoles in the liquid which makes them very interconnected, it seems easier to refer these parameters directly to data for the pure elemenets, even if one consider the quadrupoles as a hypothetical model tool. There can also be interaction parameters between the quadrupoles. A quadrupole can contain 2 to 4 different species, A, B, X and Y and some of these species may contain more than one element. The vacancy species can be one of these species and thus a quadrupole can represent a single element.

E.5 The MQMQA configurational entropy

Using the quadrupol fractions, $p_{AB/XY}$ as the basic constituent variable we can follow the paper by Pelton et al [24] to express other variables used in the MQMQA model.

The amount of A according to eq. E31 per mole formula unit of the liquid can be expressed using the quadrupole fractions $p_{\mathrm{AB/XY}}$ in eq. E7 with the coordination numbers, $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$:

$$\begin{aligned} M_{\mathrm{A}} &= \sum_{\mathrm{X}}\sum_{Y\geq X}\left(\frac{p_{\mathrm{AA/XY}}}{Z^{\mathrm{A}}_{\mathrm{AA/XY}}}+\sum_{\mathrm{B}}\frac{p_{\mathrm{AB/XY}}}{Z^{\mathrm{A}}_{\mathrm{AB/XY}}}\right) \qquad \text{(E11)}\\ M_{\mathrm{X}} &= \sum_{\mathrm{A}}\sum_{B\geq A}\left(\frac{p_{\mathrm{AB/XX}}}{Z^{\mathrm{X}}_{\mathrm{AB/XX}}}+\sum_{\mathrm{Y}}\frac{p_{\mathrm{AB/XY}}}{Z^{\mathrm{X}}_{\mathrm{AB/XY}}}\right)\end{aligned}$$

However, using the stoichiometry of the constituents as entered in Table E2 where a factor 2 is included whenever an element appear twice in any sublattice, we can use eq. 2:

$$\begin{aligned} M_{\mathrm{A}} &= \sum_i b_{i\mathrm{A}}y_i \qquad \text{(E12)}\\ M_{\mathrm{X}} &= \sum_i b_{i\mathrm{X}}y_i\end{aligned}$$

where y_i is the fraction of quadrupole i and $b_{i\mathrm{A}}$ and $b_{i\mathrm{X}}$ are the stoichiometric factor for element A and X respectively. If A or X is not included in i then $b_{i\mathrm{X}} = b_{i\mathrm{A}} = 0$.

From eq. E12 we obtain the site fractions of components A and X as:

$$\begin{aligned} \vartheta_{\mathrm{A}} &= \frac{M_{\mathrm{A}}}{\sum_{\mathrm{A}} M_{\mathrm{A}}} \qquad \text{(E13)}\\ \vartheta_{\mathrm{X}} &= \frac{M_{\mathrm{X}}}{\sum_{\mathrm{X}} M_{\mathrm{X}}}\\ \sum_{\mathrm{A}}\vartheta_{\mathrm{A}} &= 1 \qquad \text{(E14)}\\ \sum_{\mathrm{X}}\vartheta_{\mathrm{X}} &= 1\end{aligned}$$

The endmember fractions, here denoted $\xi_{\mathrm{A/X}}$, can be calculated from the quadrupole fractions $p_{\mathrm{AB/XY}}$ using eq. E32 in the Appendix:

$$\begin{aligned} \xi_{\mathrm{A/X}} &= 0.25(p_{\mathrm{AA/XX}}+\sum_{\mathrm{Y}}p_{\mathrm{AA/XY}}+\sum_{\mathrm{B}}p_{\mathrm{AB/XX}}+\sum_{\mathrm{B}}\sum_{\mathrm{Y}}p_{\mathrm{AB/XY}}) \qquad \text{(E15)}\\ &= p_{\mathrm{AA/XX}}+0.5\sum_{\mathrm{Y>X}}p_{\mathrm{AA/XY}}+0.5\sum_{\mathrm{B>A}}p_{\mathrm{AB/XX}}+0.25\sum_{\mathrm{B>A}}\sum_{\mathrm{Y>X}}p_{\mathrm{AB/XY}} \qquad \text{(E16)}\\ \sum_{\mathrm{A}}\sum_{\mathrm{X}}\xi_{\mathrm{A/X}} &= 1 \qquad \text{(E17)}\end{aligned}$$

The summations in eq. E16 for B and Y are for all elements in first and second sublattice respectively, thus the $p_{\mathrm{AA/XX}}$ will occur 4 times in the summation. See the original paper by Pelton et al. [24] for the derivation.

Finally the "coordination equivalent site fraction", w_{A}:

$$\begin{aligned} w_{\mathrm{A}} &= 0.5(\sum_{\mathrm{X}}\sum_{\mathrm{Y\geq X}} p_{\mathrm{AA/XY}} + \sum_{\mathrm{B}}\sum_{\mathrm{X}}\sum_{\mathrm{Y\geq X}} p_{\mathrm{AB/XY}}) \qquad \text{(E18)} \\ &= \sum_{\mathrm{X}}\sum_{\mathrm{Y\geq X}} p_{\mathrm{AA/XY}} + 0.5\sum_{\mathrm{B>A}}\sum_{\mathrm{X}}\sum_{\mathrm{Y\geq X}} p_{\mathrm{AB/XY}} \qquad \text{(E19)} \end{aligned}$$

from eq. 11 in [24]. In the first line all fractions $p_{\mathrm{AA/XY}}$ with 2 AA will occur twice in the summation.

In OC it may be simpler to check the stoichiometry of the quadrupoles when summing eqs. E16 and E19 but there is a possible problem with a factor 2 which cause a deviation between the $Z^{\mathrm{A}}_{AB/XY}$ and the stochiometry as explaned in section E.8.

We can now write the configurational entropy for one mole formula unit of the liquid equivalent to eq. E33 using the quadrupol frations or variables that are expressed in these:

$$\begin{aligned} -S_M/R = & \sum_{\mathrm{A}} \vartheta_{\mathrm{A}} \ln(\vartheta_{\mathrm{A}}) + \sum_{\mathrm{X}} \vartheta_{\mathrm{X}} \ln(\vartheta_{\mathrm{X}}) + \\ & \sum_{\mathrm{A}}\sum_{\mathrm{X}} \xi_{\mathrm{A/X}} \ln(\frac{\xi_{\mathrm{A/X}}}{w_{\mathrm{A}}w_{\mathrm{X}}}) + \\ & \sum_{\mathrm{A}}\sum_{\mathrm{X}} p_{\mathrm{AA/XX}} \ln\left(\frac{p_{\mathrm{AA/XX}}}{\frac{\xi^4_{\mathrm{A/X}}}{w^2_{\mathrm{A}}w^2_{\mathrm{X}}}}\right) + \\ & \sum_{\mathrm{A}}\sum_{\mathrm{B>A}}\left[\sum_{\mathrm{X}} p_{\mathrm{AB/XX}} \ln\left(\frac{p_{\mathrm{AB/XX}}}{\frac{2\xi^2_{\mathrm{A/X}}\xi^2_{\mathrm{B/X}}}{w_{\mathrm{A}}w_{\mathrm{B}}w^2_{\mathrm{X}}}}\right)\right] + \\ & \sum_{\mathrm{A}}\left[\sum_{\mathrm{X}}\sum_{\mathrm{Y>X}} p_{\mathrm{AA/XY}} \ln\left(\frac{p_{\mathrm{AA/XY}}}{\frac{2\xi^2_{\mathrm{A/X}}\xi^2_{\mathrm{A/Y}}}{w^2_{\mathrm{A}}w_{\mathrm{X}}w_{\mathrm{Y}}}}\right)\right] + \\ & \sum_{\mathrm{A}}\sum_{\mathrm{B>A}}\left[\sum_{\mathrm{X}}\sum_{\mathrm{Y>X}} p_{\mathrm{AB/XY}} \ln\left(\frac{p_{\mathrm{AB/XY}}}{\frac{4\xi_{\mathrm{A/X}}\xi_{\mathrm{A/Y}}\xi_{\mathrm{B/X}}\xi_{\mathrm{B/Y}}}{w_{\mathrm{A}}w_{\mathrm{B}}w_{\mathrm{X}}w_{\mathrm{Y}}}}\right)\right] \qquad \text{(E20)} \end{aligned}$$

The configurational entropy depend on several fraction variables such as quadrupoles, $p_{\mathrm{AB/XY}}$, endmembers, $\xi_{\mathrm{A/X}}$, coordination equivalent site fractions, w_{A} which are all the coordination equivalent fraction, ϑ_{A} which are all obtained by different summations of the quadrupole fractions. Note that the variable coordination number Z_{A} does not appear explicitly.

For a derivation of this formula see the original paper. It is only approximative and recently some modifications has bee proposed, see Lambotte and Chartrand [17].

E.6 New OC datastructures and code

The MQMQA phase is entered as any other phase specifying the model as MQMQA. It has a single set of constituents which represent the different quadrupoles, AB/XY. A quadrupol is a species with 2-4 species, A, B, X and Y each with a stoichiometry $1/Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ or $2/Z^{\mathrm{A}}_{\mathrm{AA/XY}}$ if the element is alone in the sublattice.

A new data structure **mqmq_data** has been added. It has an allocateble array **contyp(:,:)** where the second index is the constituent number and first index index has several uses. There is also a double array **constoi** with the coordination numbers for the quadrupol. When calculating the entropy each quadrupol has to have indices to its constituent endmembers and the endmembers to its constituent elements.

First index	Use
1..4	Specifies for the 2, 3 or 4 species: 2 if alone in first sublattice, 1 if there are two in the sublattice, -2 if the species is alone in second sublattice, -1 if there are two. Its value is zero if not needed
5	endmember index, zero if not endmember
6,7	for endmember quadrupoles the index in splista record array
6,7,8,9	for other quadrupols the endmember indices in this array (2 or 4)
10	The fraction index in the phase constituent array

In the first attempt to implement this model a subroutine **mqmqa_constituents** was called when entering the constituents of the MQMQA phase. These has to be entered in a special way using a slash, "/" to separate species in the sublattices and a comma, "," between species in the same sublattice. For example FE+2/O-2 means Fe+2 in first sublattice and O-2 in second. This must be followed driectly by two reals specifying the coordination numbers. A more complex quadrupole would be FE+2,AL+3/O-2,VA which must be followed by 4 coordination numbers in the order of the species in the quadrupole. The contyp array and an addition constoi array store the species indices and their coordination numbers. A second subroutine, **mqmqa_rearrange** is called when all quadrupoles have been entered and this performs some checks for example that all endmember quadrupoles are present. It then replaces links to species in non-endmember quadrupoles to links to the related endmembers in the contyp array. It also sets a link to the phase fraction variable for the quadrupole.

In gtp3X a new subroutine **config_entropy_mqmqa** will calculate the entropy of the phase. This requires several summations to obtain the fractions on the different sublattices, χ_{A}, the fractions of endmembers, $\xi_{\mathrm{A/X}}$ and the coordination-equivalent site fraction, w_{A}.

In the subroutine the first major loop for all constituents i will calculate all ϑ, ξ and w from the $y_i = p_{\mathrm{AB/XT}}$ fractions using an internal loop extracting the elements and comparing these with the array convay(1..4,i) to know which sublattice they belong to and if they are alone in that sublattice. The constituent fraction, y_i, will be multiplied with the stoichiometric factor for the element is then added to the fraction of the sublattice species. The factor n in $n/Z^{\mathrm{A}}_{\mathrm{AA/X}}$ is part of the stoichiometry of the quadrupole.

A second loop over all constituents will calculate the entropy according to eq. E20 using these variables.

E.7 Entering constituents

The way the constituents are used in the MQMQA model requires a special method to enter them. In the Na-Si-O-F system assessed by Lambotte and Chartrand [17] there are 9 quadrupoles and the values selected for $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ are listed in Table E2. Just entering the species as substitutional set of species with the constitution according to Table E2 is not possible because one has to idenify which part of a species is a cation and which is an anion. However, the complex expression for Z_{A} from eq. E30 is not needed if the all quadrupoles are enterd in the CONSTITUENT keyword and their stoichiometry include the factor 2 for species with are alone in a sublattice.

Table E2: Values of $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ and species stoichiometry for the Na-Si-O-F system taken from [17]. Each quadrupole must be neutral and the fraction of the species with 3 or 4 elements should reflect the composition of maximum SRO.

Quadrupole	$Z^{\mathrm{A}}_{\mathrm{NaSi:OF}}$				Species	Element stoichiometry				Total
	Na	Si	F	O	name in OC	Na	Si	F	O	Elements
NaNa/OO	3	-	-	6	NA/O-Qij	2/3	-	-	1/3	1
SiSi/OO	-	6	-	3	SI/O-Qij	-	1/3	-	2/3	1
NaSi/OO	3	12	-	2	NASI/O-Qij	1/3	1/12	-	1/2	11/12
NaNa/FF	6	-	6	-	NA/F-Qij	1/3	-	1/3	-	2/3
SiSi/FF	-	6	1.5	-	SI/F-Qij	-	1/3	4/3	-	5/3
NaSi/FF	3	6	2	-	NASI/F-Qij	1/3	1/6	1	-	3/2
NaNa/FO	4	-	6	6	NA/FO-Qij	1/2	-	1/6	1/6	5/6
SiSi/FO	-	6	1.5	3	SI/FO-Qij	-	1/3	2/3	1/3	4/3
NaSi/FO	2	7.9204	12.4375	2.1630	NASI/FO-Qij	1/2	0.13..	0.08..	0.46..	1.17..

Table E3: Values of $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ and species stoichiometry for the Na-Si-O-F system taken from [17]. Each quadrupole must be neutral and the fraction of the species with 3 or 4 elements should reflect the composition of maximum SRO.

Quadrupole	$Z^{\mathrm{A}}_{\mathrm{NaSi:OF}}$				Species	Element stoichiometry				SRO
	Na	Si	F	O	name in OC	Na	Si	F	O	ratio
NaNa/OO	3	-	-	6	NA/O-Qij	2/3	-	-	1/3	-
SiSi/OO	-	6	-	3	SI/O-Qij	-	1/3	-	2/3	-
NaSi/OO	3	12	-	2	NASI/O-Qij	1/3	1/12	-	1/2	2/3
NaNa/FF	6	-	6	-	NA/F-Qij	1/3	-	1/3	-	-
SiSi/FF	-	6	1.5	-	SI/F-Qij	-	1/3	4/3	-	-
NaSi/FF	3	6	2	-	NASI/F-Qij	1/3	1/6	1	-	1/3
NaNa/FO	4	-	6	6	NA/FO-Qij	1/2	-	1/6	1/6	2/5
SiSi/FO	-	6	1.5	3	SI/FO-Qij	-	1/3	2/3	1/3	1/2
NaSi/FO	2	7.9204	12.4375	2.1630	NASI/FO-Qij	1/2	0.13..	0.08..	0.46..	-

The values in the last columns in table E3 for SRO is calculated using the quotiends of the relevant values of $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$.

$$x^{\mathrm{SRO}}_{\mathrm{AB/XX}} = \qquad \text{(E21)}$$

Table E4: Values of $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ and species stoichiometry for the Na-Si-O-F system. Each quadrupole must be neutral and the fraction of the constituents should reflect the composition of maximum SRO.

	$Z^{\mathrm{A}}_{\mathrm{NaSi:OF}}$				Species	
Quadrupole	Na	Si	O	F	Name	Stoichiometry
SiSi/OO	-	6	3	-	SI/O-Qij	$Si_{2/6}O_{2/3}$, **note the factor of 2!**
NaNa/OO	3	-	6	-	NA/O-Qij	$Na_{2/3}O_{2/6}$
NaSi/OO	3	12	1.5	-	NASI/O-Qij	$Na_{1/3}Si_{1/12}O_{2/3}$
SiSi/FF	-	6	-	1.5	Si/F-Qij	$Si_{2/6}F_{2/1.5}$
NaNa/FF	6	-	-	6	Na/F-Qij	$Na_{2/6}F_{2/6}$
NaSi/FF	3	6	-	2	NASI/F-Qij	$Na_{1/3}Si_{1/6}F_{2/2}$
SiSi/FO	-	6	3	1.5	Si/FO-Qij	$Si_{2/6}O_{1/3}F_{1/1.5}$
NaNa/FO	4	-	6	6	NA/FO-Qij	$Na_{2/4}O_{1/6}F_{1/6}$
NaSi/FO	2	7.9204	2.1630	12.4375	NASIFO-Qij	$Na_{1/2}Si_{1/7.9204}O_{1/2.1630}F_{1/12.4375}$

The proposed way to enter the constituents of the MQMQA phase for the Na-Si-O-F system from a TDB file is:

```
CONSTITUENT LIQUID:Q :NA/O 3.0 6.0 NA/F 6.0 6.0 SI,NA/O 12.0 3.0  6.0
    NA,SI/O,F 2 7.9204 2.1630 12.4375 ... : !
```

where the original slash, "/" between constituents on the different sublattices will be retained in the name of the quadrupole but not the comma, "," separating the species on the same sublatice. The values after the quadrupoles are the coordination factors, $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$ and must be in the same order as the species in the quadrupole.

All the species in the quadrupoles must already be entered as species, and it is the name of the species is used in CONSTITUENT, not the stoichiometry. The software will generate the name of the quadrupole by combining the species names in alphabetical order for each sublattice and the slash will be kept. OC will add a suffix "-Qij", where "ij" is an arbitrary sequential number, to ensure the name is unique and not an abbreviation of another species. Thus the quadrupole SI,NA/O will have the name NASI/O-Qij. The order of the coordination numbers will also be adjusted for the alphabetical order and a factor 2 is used for species alone in a sublattice when OC generate the stoichiometry of the quadrupole.

A parameter for a quadrupole in the database must use the alphabetically ordered quadrupole name, including a "-Q", but not any numbers "ij" after the Q because they are arbitrary and irreleveant. A model parameter for a quadrupole NASI/O in the TDB file should be:

```
PARAMETER G(LIQUID,NASI/O-Q) tlow expression; tmax N bibref !
```

When reading the CONSTITUENT line the software will also generate the contyp array described in section E.6 which is needed to calculate sublattice site fractions and the variables $\vartheta_{\mathrm{A}}, \xi_{\mathrm{A/X}}$ and w_{A} explained in section E.5. The mass balance equation is thus maintained by the quadrupole stoichiometry and the other fraction variables are used without any referebce to their actual stoichiometry.

The list of constituents in a TDB file can contain species with elements that are not selected, only those containing elements selected will be entered when reading the TDB file.

The fractions of the quadrupoles, $p_{\mathrm{AB/XY}}$ in the MQM model add up to unity and are used to calculate all other fractions in the model, for the mass balance, for the Gibbs energy and entropy and for all derivatives in order to calculate the equilibrium, all chemical potentials and other thermodynamic properies of the system.

E.8 Stoichiometric factors and coordiation numbers

As noted in Pelton et al. [24] the quadrupols can formally be viewed as molcules or associates in the liquid with a stoichiometry related to the $Z^{\mathrm{A}}_{\mathrm{AB/XY}}$. This stoichiometry must be taken into account when entering these species in OC. One must also take into account that the species must not very considerably in size, on the average a quadrupole should contain between 1 and 2 atoms. In Pelton et al. [24] a system K-Mg-Cl the quadrupol KMg/ClCl has the stoichiometry

Quadrupole	K	Mg	Cl	Cl	Species
KMg/ClCl	3	6	3	3	$K_{1/3}Mg_{1/6}Cl_{2/3}$

where the species $K_{1/3}Mg_{1/6}Cl_{2/3}$ or $(KCl)_2MgCl_2$ is electrically neutral using the normal valencies of the elements. As Cl occur twice in the second sublattice its stoichiometry is multiplied with 2 relative to the value of $Z^{\mathrm{Cl}}_{\mathrm{KMg/ClCl}}$. This is the factor 2 in the quasichemical model.

E.9 Derivatives

OC requires exact first derivatives with respect to the quadrupole fractions, y_i, of the entropy and at least approximate second derivatives. We must start from the derivatives of the variables $\mathrm{M_A}$, $\vartheta_{\mathrm{A}}, \xi_{\mathrm{A/X}}$ and w_{A} according to eqs E12, E13, E16 and E19. The Kronecker-delta, $\delta_{AB} = 1$ if A and B the same, 0 otherwise, is used below

Derivative	symbol	expression
$\frac{\partial M_{\mathrm{A}}}{\partial y_i} = \frac{\partial \mathrm{M_A}}{\partial p_{\mathrm{AB/XY}}}$	$\mathrm{M}_{i\mathrm{A}}$	$\sum_i b_{i,\mathrm{A}}$
$\frac{\partial \vartheta_{\mathrm{A}}}{\partial y_i} = \frac{\partial \vartheta_{\mathrm{A}}}{\partial p_{\mathrm{BC/XY}}}$	$\vartheta_{i,\mathrm{A}}$	$\delta_{\mathrm{AB}} \frac{\sum_i b_{i\mathrm{A}} - \vartheta_{\mathrm{A}} \sum_{\mathrm{A}} \sum_i b_{i\mathrm{A}}}{\sum_{\mathrm{A}} M_{\mathrm{A}}} = \delta_{\mathrm{AB}} \frac{\mathrm{M}_{i\mathrm{A}} - \vartheta_{\mathrm{A}} \sum_{\mathrm{A}} \mathrm{M}_{i,\mathrm{A}}}{\sum_{\mathrm{A}} \mathrm{M_A}}$
$\frac{\partial w_{\mathrm{A}}}{\partial y_i} = \frac{\partial w_{\mathrm{A}}}{\partial p_{\mathrm{BC/XY}}}$	$\mathrm{w}_{i,\mathrm{A}}$	$0.5(\delta_{\mathrm{AB}} + \delta_{\mathrm{AC}})$
$\frac{\partial \xi_{\mathrm{A/X}}}{\partial y_i} = \frac{\partial \xi_{\mathrm{A/X}}}{\partial p_{\mathrm{BC/YZ}}}$	$\xi_{i,\mathrm{A/X}}$	$0.25\delta_{\mathrm{AB}}\delta_{XY}(1 + \delta_{\mathrm{AC}} + \delta_{\mathrm{XZ}} + \delta_{\mathrm{AC}}\delta_{\mathrm{XZ}})$

(E22)

(E23)

E.10 The MQM model in the original Pelton notation

This appendix repeat the equations from the paper by Pelton et al. [24] which are referenced above. All symbols explained in Table E5 using the same notation as in the paper.

Table E5: Symbols used in Pelton et al. [24]

Symbol	Meaning
n_A	Moles of element A in the liquid
X_A	Site fraction of element A on first sublattice.
X_X	Site fraction of element X on second sublattice.
Z_A	Average coordination number (bonds) for element A in the liquid. Varies with the composition.
$n_\mathrm{AB/XY}$	Moles of a quadrupole (cluster) consisting of A and B in first sublattice and elements X and Y and second sublattice.
$X_\mathrm{AB/XY}$	Fraction of cluster AB/XY.
$n_\mathrm{A/X}$	Number of moles of an endmember with A in first and X in second sublattice.
$X_\mathrm{A/X}$	Fraction of the endmember A/X.
$n_{ij/kl}$	Any cluster with elements ij in first and kl in second sublattice
Y_A	Coordination-equivalent site fraction for element A.
$Z^\mathrm{A}_\mathrm{AB/XY}$	Coordinaton number of element A in the cluster AB/XY. This is a constant.

Elements A, B etc. are in first sublattice, X, Y etc. in second sublattce. Note X, Y and Z are used for other tings. Definition of the average coordination number Z_A for element A:

$$Z_\mathrm{A} n_\mathrm{A} \;=\; 2n_{\mathrm{A_2/X_2}} + 2n_{\mathrm{A_2/Y_2}} + 2n_{\mathrm{A_2/XY}} + n_{\mathrm{AB/X_2}} + \cdot \tag{E24}$$

with the factor 2 is for all clusters $n_\mathrm{AB/XY}$ with the element A twice. We have similar equations for element B, X etc. This is equivalent to eqs. 4 and 5 in [24].

Definition of site fractions, endmember fractions and quadrupol (cluster) fractions (equivalent to eqs. 6-8 in [24]):

$$X_\mathrm{A} \;=\; \frac{n_\mathrm{A}}{\sum_A n_\mathrm{A}} \tag{E25}$$

$$X_\mathrm{A/X} \;=\; \frac{n_\mathrm{A/X}}{\sum_A \sum_\mathrm{X} n_\mathrm{A/X}} \tag{E26}$$

$$X_\mathrm{AB/XY} \;=\; \frac{n_\mathrm{AB/XY}}{\sum_A \sum_\mathrm{B\geq A}(\sum_\mathrm{X} \sum_\mathrm{Y\geq X} n_\mathrm{AB/XY})} \tag{E27}$$

where A, B and X, Y are used both for the specific fraction and in the summation for all fractions.

Definition of "coordination-equivalent site fractions" (equivalent to eq.10 in [24]):

$$Y_{\mathrm{A}} = \frac{Z_{\mathrm{A}} n_{\mathrm{A}}}{\sum_{\mathrm{A}} Z_{\mathrm{A}} n_{\mathrm{A}}} \tag{E28}$$

which is needed to correct the configurational entropy expression, eq. E33. We have similar equations for B, X etc. This is was transformed by Pelton et al [24] to:

$$Y_{\mathrm{A}} = 0.5 \sum_{\mathrm{X}} \sum_{\mathrm{Y} \geq \mathrm{X}} (p_{\mathrm{AA/XY}} + \sum_{\mathrm{B}} p_{\mathrm{AB/XY}}) \tag{E29}$$

Equation for the composition dependendence of Z_{A} (equivalent to eqs 13-14 in [24]):

$$\frac{1}{Z_{\mathrm{A}}} = \frac{\frac{2n_{\mathrm{A_2/X_2}}}{Z^{\mathrm{A}}_{\mathrm{A_2/X_2}}} + 2\frac{n_{\mathrm{A_2/Y_2}}}{Z^{\mathrm{A}}_{\mathrm{A_2/Y_2}}} + 2\frac{n_{\mathrm{A_2/XY}}}{Z^{\mathrm{A}}_{\mathrm{A_2/XY}}} + \frac{n_{\mathrm{AB/X_2}}}{Z^{\mathrm{A}}_{\mathrm{AB/X_2}}} + \cdots}{2n_{\mathrm{A_2/X_2}} + 2n_{\mathrm{A_2/Y_2}} + 2n_{\mathrm{A_2/XY}} + n_{\mathrm{AB/X_2}} + \cdots} \tag{E30}$$

and similarly for B, X etc. This equation is used to obtain a new equation for the total number of moles of element A (equivalent to eq. 15-16 in [24] relative to the quadrupol (cluster) fractions):

$$\begin{aligned} n_{\mathrm{A}} &= 2\sum_{\mathrm{X}} \sum_{\mathrm{Y} \geq \mathrm{X}} \frac{n_{\mathrm{A_2/XY}}}{Z^{\mathrm{A}}_{\mathrm{A_2/XY}}} + \sum_{\mathrm{B} \neq \mathrm{A}} \sum_{\mathrm{X}} \sum_{\mathrm{Y} \geq \mathrm{X}} \frac{n_{\mathrm{AB/XY}}}{Z^{\mathrm{A}}_{\mathrm{AB/XY}}} \\ &= \sum_{\mathrm{X}} \sum_{\mathrm{Y} \geq \mathrm{X}} (\frac{n_{\mathrm{A_2/XY}}}{Z^{\mathrm{A}}_{\mathrm{A_2/XY}}} + \sum_{\mathrm{B}} \frac{n_{\mathrm{AB/XY}}}{Z^{\mathrm{A}}_{\mathrm{AB/XY}}}) \end{aligned} \tag{E31}$$

and similarly of B, X etc.

Using eq. 21 in [24] we can express the endmember fractions as:

$$X_{\mathrm{A/X}} = 0.25(X_{\mathrm{AA/XX}} + \sum_{\mathrm{Y}} X_{\mathrm{AA/XY}} + \sum_{\mathrm{B}} X_{\mathrm{AB/XX}} + \sum_{\mathrm{B}} \sum_{\mathrm{Y}} X_{\mathrm{AB/XY}}) \tag{E32}$$

where the summations over B and Y start with A and X and thus the cluster $X_{\mathrm{AA/XX}}$ appear in all 3 summations and terms such as $X_{\mathrm{AA/XY}}$ and $X_{\mathrm{AB/XX}}$ in two of them.

Using these variables we can write the configurational entropy for $(\sum_{\mathrm{A}} n_{\mathrm{A}} + \sum_{\mathrm{X}} n_{\mathrm{X}})$ moles of liquid (eq. 39 in [24]):

$$\begin{aligned} -S/R &= \sum_{\mathrm{A}} n_{\mathrm{A}} \ln(X_{\mathrm{A}}) + \sum_{\mathrm{X}} n_{\mathrm{X}} \ln(X_{\mathrm{X}}) + \\ &\sum_{\mathrm{A}} \sum_{\mathrm{X}} n_{\mathrm{A/X}} \ln(\frac{X_{\mathrm{A/X}}}{Y_{\mathrm{A}} Y_{\mathrm{X}}}) + \\ &\sum_{\mathrm{A}} \sum_{\mathrm{X}} n_{\mathrm{A_2/X_2}} \ln\left(\frac{X_{\mathrm{A_2/X_2}}}{\frac{X^4_{\mathrm{A/X}}}{Y^2_{\mathrm{A}} Y^2_{\mathrm{X}}}}\right) + \\ &\sum_{\mathrm{A}} \sum_{\mathrm{B}>\mathrm{A}} (\sum_{\mathrm{X}} n_{\mathrm{AB/X_2}} \ln\left(\frac{X_{\mathrm{AB/X_2}}}{\frac{2X^2_{\mathrm{A/X}} X^2_{\mathrm{B/X}}}{Y_{\mathrm{A}} Y_{\mathrm{B}} Y^2_{\mathrm{X}}}}\right)) + \end{aligned}$$

$$\sum_{\mathrm{A}}(\sum_{\mathrm{X}}\sum_{\mathrm{Y>X}} n_{\mathrm{A_2/XY}} \ln\left(\frac{X_{\mathrm{A_2/XY}}}{\frac{2X^2_{\mathrm{A/X}}X^2_{\mathrm{A/Y}}}{Y^2_{\mathrm{A}}Y_{\mathrm{X}}Y_{\mathrm{Y}}}}\right)) +$$

$$\sum_{\mathrm{A}}\sum_{\mathrm{B>A}}(\sum_{\mathrm{X}}\sum_{\mathrm{Y>X}} n_{\mathrm{AB/XY}} \ln\left(\frac{X_{\mathrm{AB/XY}}}{\frac{4X_{\mathrm{A/X}}X_{\mathrm{A/Y}}X_{\mathrm{B/X}}X_{\mathrm{B/Y}}}{Y_{\mathrm{A}}Y_{\mathrm{B}}Y_{\mathrm{X}}Y_{\mathrm{Y}}}}\right)) \qquad \text{(E33)}$$

where the first line is the configurational entropy from ideal mixing the of elements on the two sublattices, the second is the contribution from mixing of the endmembers and the last 4 lines the contribution from mixing of the quadrupols (clusters). The expressions inside the logarithms are an approximate reduction of the entropy due to sharing of surfaces, edges and points of the different clusters.

In a later paper by Lambotte and Chartrand [17] some of the factors in these equations has been modified. Those will be discussed separately.

Appendix F Table with all 419 functions and subroutines

In alphabetical order (including the type of function) indicating the file where the routine occurs.

Name	File
double precision function evaluate-svfun-old	gtp3F.F90
double precision function mass-of	gtp3A.F90
integer function ceqrecsize	gtp3E.F90
integer function ceqsize	gtp3Y.F90
integer function degrees-of-freedom	gtp3C.F90
integer function find-phasetuple-by-indices	gtp3A.F90
integer function get-mpi-index	gtp3Y.F90
integer function get-phase-status	gtp3G.F90
integer function get-phtuplearray	gtp3A.F90
integer function getmqindex	gtp3Y.F90
integer function gettupix	gtp3A.F90
integer function ispdbkeyword	gtp3E.F90
integer function istdbkeyword	gtp3E.F90
integer function newhighcs	gtp3B.F90
integer function noconst	gtp3A.F90
integer function noel	gtp3A.F90
integer function noeq	gtp3A.F90
integer function nonsusphcs	gtp3A.F90
integer function noofcs	gtp3A.F90
integer function noofphasetuples	gtp3A.F90
integer function nooftup	gtp3A.F90
integer function noph	gtp3A.F90
integer function nosp	gtp3A.F90
integer function nosvf	gtp3A.F90
integer function notpf	gtp3Z.F90
integer function phvarlok	gtp3Y.F90
integer function test-phase-status	gtp3G.F90
integer function vssize	gtp3Y.F90
logical function allotropes	gtp3Y.F90
logical function allowenter	gtp3Y.F90
logical function check-minimal-ford	gtp3B.F90
logical function fixedcomposition	gtp3Y.F90
logical function global-equil-check1	gtp3Y.F90
logical function gtp-error-message	gtp3C.F90
logical function inveq	gtp3Y.F90
logical function iskeyword	gtp3E.F90
logical function notallowlisting	gtp3E.F90
logical function ocv	gtp3Y.F90

Name	File
logical function proper-symbol-name	gtp3Y.F90
logical function same-statevariable	gtp3D.F90
logical function same-stoik	gtp3Y.F90
logical function test-phase-status-bit	gtp3G.F90
logical function testelstat	gtp3G.F90
logical function testspstat	gtp3G.F90
subroutine add-addrecord	gtp3H.F90
subroutine add-fraction-set	gtp3G.F90
subroutine add-size-derivatives	gtp3H.F90
subroutine add-ternary-extrapol-method	gtp3H.F90
subroutine add-to-reference-phase	gtp3G.F90
subroutine add1tpcoeffs	gtp3Z.F90
subroutine addition-selector	gtp3H.F90
subroutine adjust-1range	gtp3Z.F90
subroutine adjustranges	gtp3Z.F90
subroutine alphaelorder	gtp3G.F90
subroutine alphaphorder	gtp3G.F90
subroutine alphasporder	gtp3G.F90
subroutine amend-components	gtp3A.F90
subroutine amend-global-data	gtp3D.F90
subroutine any-disordered-part	gtp3E.F90
subroutine apply-condition-value	gtp3D.F90
subroutine ask-default-constitution	gtp3D.F90
subroutine ask-phase-constitution	gtp3D.F90
subroutine ask-phase-new-constitution	gtp3D.F90
subroutine assessmenthead	gtp3A.F90
subroutine bccendmem	gtp3B.F90
subroutine bccint1	gtp3B.F90
subroutine bccint2	gtp3B.F90
subroutine bccpermuts	gtp3B.F90
subroutine biblioread	gtp3E.F90
subroutine bibliosave	gtp3E.F90
subroutine calc-debyecp	gtp3H.F90
subroutine calc-diffusion	gtp3H.F90
subroutine calc-disfrac	gtp3X.F90
subroutine calc-disfrac2	gtp3X.F90
subroutine calc-eec-gibbsenergy	gtp3X.F90
subroutine calc-einsteincp	gtp3H.F90
subroutine calc-elastica	gtp3H.F90
subroutine calc-gridpoint	gtp3Y.F90
subroutine calc-magnetic-inden	gtp3H.F90
subroutine calc-molmass	gtp3F.F90
subroutine calc-mqmqa	gtp3X.F90
subroutine calc-phase-mol	gtp3F.F90
subroutine calc-phase-molmass	gtp3F.F90

Name	File
subroutine calc-qf	gtp3F.F90
subroutine calc-qf-old	gtp3F.F90
subroutine calc-qf-otis	gtp3F.F90
subroutine calc-qf-sub	gtp3F.F90
subroutine calc-schottky-anomaly	gtp3H.F90
subroutine calc-secondeinstein	gtp3H.F90
subroutine calc-toop	gtp3X.F90
subroutine calc-twostate-model-endmember	gtp3H.F90
subroutine calc-twostate-model-john	gtp3H.F90
subroutine calc-twostate-model-old	gtp3H.F90
subroutine calc-twostate-model1	gtp3H.F90
subroutine calc-twostate-model2	gtp3H.F90
subroutine calc-volmod1	gtp3H.F90
subroutine calc-xiongmagnetic	gtp3H.F90
subroutine calcg	gtp3X.F90
subroutine calcg-endmember	gtp3Y.F90
subroutine calcg-endmember6	gtp3Y.F90
subroutine calcg-endmemberx	gtp3Y.F90
subroutine calcg-internal	gtp3X.F90
subroutine calculate-reference-state	gtp3F.F90
subroutine calculate-reference-state-old	gtp3F.F90
subroutine cgint	gtp3X.F90
subroutine change-element-status	gtp3G.F90
subroutine change-many-phase-status	gtp3G.F90
subroutine change-optcoeff	gtp3Z.F90
subroutine change-phase-status	gtp3G.F90
subroutine change-phtup-status	gtp3G.F90
subroutine change-species-status	gtp3G.F90
subroutine check-all-phases	gtp3Y.F90
subroutine check-alphaindex	gtp3G.F90
subroutine check-phase-grid	gtp3Y.F90
subroutine checkdb2	gtp3E.F90
subroutine checkdefcon	gtp3Y.F90
subroutine checkpowers	gtp3Z.F90
subroutine clear-phase-status-bit	gtp3G.F90
subroutine compmassbug	gtp3Y.F90
subroutine condition-value	gtp3D.F90
subroutine config-entropy	gtp3X.F90
subroutine config-entropy-cqc-classicqc	gtp3X.F90
subroutine config-entropy-cvmce	gtp3X.F90
subroutine config-entropy-i2sl	gtp3X.F90
subroutine config-entropy-mqmqa	gtp3X.F90
subroutine config-entropy-qchillert	gtp3X.F90
subroutine config-entropy-qcwithlro	gtp3X.F90
subroutine config-entropy-srot	gtp3X.F90

Name	File
subroutine config-entropy-ssro	gtp3X.F90
subroutine config-entropy-tisr provided by Ed Kremer	gtp3X.F90
subroutine copy-condition	gtp3B.F90
subroutine copy-equilibrium	gtp3B.F90
subroutine copy-equilibrium2	gtp3B.F90
subroutine copy-fracset-record	gtp3G.F90
subroutine copyfracs(fromeq,ceq)	gtp3G.F90
subroutine create-constitlist	gtp3G.F90
subroutine create-debyecp	gtp3H.F90
subroutine create-diffusion	gtp3H.F90
subroutine create-einsteincp	gtp3H.F90
subroutine create-elastic-model-a	gtp3H.F90
subroutine create-endmember	gtp3G.F90
subroutine create-interaction	gtp3G.F90
subroutine create-magrec-inden	gtp3H.F90
subroutine create-newtwostate-model1	gtp3H.F90
subroutine create-parrecords	gtp3G.F90
subroutine create-proprec	gtp3G.F90
subroutine create-schottky-anomaly	gtp3H.F90
subroutine create-secondeinstein	gtp3H.F90
subroutine create-twostate-model1	gtp3H.F90
subroutine create-volmod1	gtp3H.F90
subroutine create-xiongmagnetic	gtp3H.F90
subroutine ct1efn	gtp3Z.F90
subroutine ct1getsym	gtp3Z.F90
subroutine ct1mexpr	gtp3Z.F90
subroutine ct1mfn	gtp3Z.F90
subroutine ct1power	gtp3Z.F90
subroutine ct1wfn	gtp3Z.F90
subroutine ct1wpow	gtp3Z.F90
subroutine ct1xfn	gtp3Z.F90
subroutine ct2mfn	gtp3Z.F90
subroutine deallocate-gtp	gtp3A.F90
subroutine decode-constarr	gtp3C.F90
subroutine decode-state-variable	gtp3F.F90
subroutine decode-state-variable3	gtp3F.F90
subroutine decode-stoik	gtp3C.F90
subroutine delete-all-conditions	gtp3B.F90
subroutine delete-all-tpfuns	gtp3Z.F90
subroutine delete-equilibria	gtp3B.F90
subroutine delete-unstable-compsets	gtp3B.F90
subroutine delphase	gtp3A.F90
subroutine diffusion-onoff	gtp3H.F90
subroutine disordery	gtp3X.F90
subroutine disordery2	gtp3X.F90

Name	File
subroutine elements2components1	gtp3G.F90
subroutine encode-constarr	gtp3C.F90
subroutine encode-state-variable	gtp3F.F90
subroutine encode-state-variable-record	gtp3F.F90
subroutine encode-state-variable3	gtp3F.F90
subroutine encode-stoik	gtp3C.F90
subroutine enter-bibliography-interactivly	gtp3D.F90
subroutine enter-composition-set	gtp3B.F90
subroutine enter-default-constitution	gtp3D.F90
subroutine enter-equilibrium	gtp3B.F90
subroutine enter-experiment	gtp3D.F90
subroutine enter-many-equil	gtp3B.F90
subroutine enter-material	gtp3B.F90
subroutine enter-optvars	gtp3Z.F90
subroutine enter-parameter	gtp3B.F90
subroutine enter-parameter-interactivly	gtp3D.F90
subroutine enter-phase	gtp3B.F90
subroutine enter-species	gtp3B.F90
subroutine enter-species-property	gtp3B.F90
subroutine enter-svfun	gtp3F.F90
subroutine enter-tpfun-interactivly	gtp3Z.F90
subroutine enterphase	gtp3B.F90
subroutine eval-tpfun	gtp3Z.F90
subroutine evaluate-all-svfun-old	gtp3F.F90
subroutine expand-wildcards	gtp3E.F90
subroutine extend-proprec	gtp3G.F90
subroutine extract-massbalcond	gtp3X.F90
subroutine extract-stvr-of-condition	gtp3D.F90
subroutine fccint211	gtp3B.F90
subroutine fccint22	gtp3B.F90
subroutine fccint31	gtp3B.F90
subroutine fccip2A	gtp3B.F90
subroutine fccip2B	gtp3B.F90
subroutine fccpe1111	gtp3B.F90
subroutine fccpe211	gtp3B.F90
subroutine fccpermuts	gtp3B.F90
subroutine find-component-by-name	gtp3A.F90
subroutine find-constituent	gtp3A.F90
subroutine find-defined-property	gtp3C.F90
subroutine find-defined-property3	gtp3C.F90
subroutine find-element-by-name	gtp3A.F90
subroutine find-gridmin	gtp3Y.F90
subroutine find-phase-by-name	gtp3A.F90
subroutine find-phase-by-name-exact	gtp3A.F90
subroutine find-phasetuple-by-name	gtp3A.F90

Name	File
subroutine find-phasex-by-name	gtp3A.F90
subroutine find-species-by-name	gtp3A.F90
subroutine find-species-by-name-exact	gtp3A.F90
subroutine find-species-record	gtp3A.F90
subroutine find-species-record-exact	gtp3A.F90
subroutine find-species-record-noabbr	gtp3A.F90
subroutine find-svfun	gtp3F.F90
subroutine find-symbol-with-equilno	gtp3F.F90
subroutine find-tpfun-by-name	gtp3Z.F90
subroutine find-tpfun-by-name-exact	gtp3Z.F90
subroutine find-tpsymbol	gtp3Z.F90
subroutine findconst	gtp3A.F90
subroutine findeq	gtp3A.F90
subroutine findsublattice	gtp3F.F90
subroutine findtpused	gtp3Z.F90
subroutine force-recalculate-tpfuns	gtp3Z.F90
subroutine format-phase-composition	gtp3C.F90
subroutine geneqname	gtp3B.F90
subroutine generate-charged-grid	gtp3Y.F90
subroutine generate-dense-grid	gtp3Y.F90
subroutine generate-fccord-grid	gtp3Y.F90
subroutine generate-grid	gtp3Y.F90
subroutine generate-ionliq-grid	gtp3Y.F90
subroutine generic-grid-generator	gtp3Y.F90
subroutine get-all-conditions	gtp3C.F90
subroutine get-all-opt-coeff	gtp3Z.F90
subroutine get-component-name	gtp3A.F90
subroutine get-condition	gtp3D.F90
subroutine get-condition-expression	gtp3D.F90
subroutine get-condition2	gtp3D.F90
subroutine get-constituent-data	gtp3A.F90
subroutine get-constituent-location	gtp3A.F90
subroutine get-constituent-name	gtp3A.F90
subroutine get-diffusion-matrix	gtp3H.F90
subroutine get-element-data	gtp3A.F90
subroutine get-experiment-with-symbol	gtp3D.F90
subroutine get-many-svar	gtp3F.F90
subroutine get-one-condition	gtp3C.F90
subroutine get-one-experiment	gtp3C.F90
subroutine get-parameter-typty	gtp3D.F90
subroutine get-phase-compset	gtp3A.F90
subroutine get-phase-data	gtp3A.F90
subroutine get-phase-name	gtp3A.F90
subroutine get-phase-record	gtp3A.F90
subroutine get-phase-structure	gtp3A.F90

Name	File
subroutine get-phase-variance	gtp3A.F90
subroutine get-phasetup-name	gtp3A.F90
subroutine get-phasetup-record	gtp3A.F90
subroutine get-phasetuple-name	gtp3A.F90
subroutine get-species-component-data	gtp3A.F90
subroutine get-species-data	gtp3A.F90
subroutine get-species-location	gtp3A.F90
subroutine get-species-name	gtp3A.F90
subroutine get-stable-state-var-value	gtp3F.F90
subroutine get-state-var-value	gtp3F.F90
subroutine get-stoichiometry	gtp3A.F90
subroutine get-value-of-constant-index	gtp3Z.F90
subroutine get-value-of-constant-name	gtp3Z.F90
subroutine global-gridmin	gtp3Y.F90
subroutine gtpread	gtp3E.F90
subroutine gtpsave	gtp3E.F90
subroutine gtpsavedir	gtp3E.F90
subroutine gtpsavelatex	gtp3E.F90
subroutine gtpsavetdb	gtp3E.F90
subroutine gtpsavetm	gtp3E.F90
subroutine gtpsaveu	gtp3E.F90
subroutine incunique	gtp3E.F90
subroutine init-gtp	gtp3A.F90
subroutine initialize-default-global-parameters	gtp3A.F90
subroutine intsort	gtp3E.F90
subroutine line-with-phases-withdgm0	gtp3C.F90
subroutine list-TDB-format	gtp3C.F90
subroutine list-addition	gtp3H.F90
subroutine list-addition-values	gtp3H.F90
subroutine list-all-components	gtp3C.F90
subroutine list-all-elements	gtp3C.F90
subroutine list-all-elements2(unit)	gtp3C.F90
subroutine list-all-funs	gtp3Z.F90
subroutine list-all-phases	gtp3C.F90
subroutine list-all-species	gtp3C.F90
subroutine list-all-svfun	gtp3F.F90
subroutine list-bibliography	gtp3C.F90
subroutine list-components-result	gtp3C.F90
subroutine list-conditions	gtp3C.F90
subroutine list-defined-properties	gtp3C.F90
subroutine list-element-data	gtp3C.F90
subroutine list-equilibria-details	gtp3C.F90
subroutine list-free-lists	gtp3Y.F90
subroutine list-global-results	gtp3C.F90
subroutine list-many-formats	gtp3C.F90

Name	File
subroutine list-phase-data	gtp3C.F90
subroutine list-phase-data2	gtp3C.F90
subroutine list-phase-model	gtp3C.F90
subroutine list-phase-results	gtp3C.F90
subroutine list-phases-with-positive-dgm	gtp3C.F90
subroutine list-short-results	gtp3C.F90
subroutine list-sorted-phases	gtp3C.F90
subroutine list-species-data	gtp3C.F90
subroutine list-species-data2	gtp3C.F90
subroutine list-svfun	gtp3F.F90
subroutine list-tpascoef	gtp3Z.F90
subroutine list-tpfun	gtp3Z.F90
subroutine list-tpfun-details	gtp3Z.F90
subroutine list-unentered-funs	gtp3Z.F90
subroutine listoptcoeff	gtp3C.F90
subroutine locate-condition	gtp3D.F90
subroutine lower-case-species-name	gtp3E.F90
subroutine make-stvrec	gtp3F.F90
subroutine makeoptvname	gtp3Z.F90
subroutine mark-stable-phase	gtp3G.F90
subroutine merge-gridpoints	gtp3Y.F90
subroutine mqmqa-addquads	gtp3B.F90
subroutine mqmqa-constituents	gtp3B.F90
subroutine mqmqa-quadbonds	gtp3B.F90
subroutine mqmqa-rearrange	gtp3B.F90
subroutine need-propertyid	gtp3H.F90
subroutine nested-tpfun	gtp3Z.F90
subroutine new-element-data	gtp3A.F90
subroutine new-gtp	gtp3A.F90
subroutine palmtree	gtp3Y.F90
subroutine pop-pyval	gtp3X.F90
subroutine push-pyval	gtp3X.F90
subroutine read0tpfun	gtp3Z.F90
subroutine readash	gtp3E.F90
subroutine readendmem	gtp3E.F90
subroutine readequil	gtp3E.F90
subroutine readintrec	gtp3E.F90
subroutine readpdb	gtp3E.F90
subroutine readphases	gtp3E.F90
subroutine readproprec	gtp3E.F90
subroutine readtdb	gtp3E.F90
subroutine readtdbsilent	gtp3E.F90
subroutine remove-composition-set	gtp3B.F90
subroutine restore-constitutions	gtp3X.F90
subroutine restore-phases-implicitly-suspended	gtp3G.F90

Name	File
subroutine restore-species-implicitly-suspended	gtp3G.F90
subroutine save-constitutions	gtp3X.F90
subroutine save-datformat	gtp3E.F90
subroutine save-phase-constitutions	gtp3X.F90
subroutine save0tpfun	gtp3Z.F90
subroutine saveash	gtp3E.F90
subroutine saveequil	gtp3E.F90
subroutine savephases	gtp3E.F90
subroutine select-composition-set	gtp3Y.F90
subroutine selecteq	gtp3A.F90
subroutine separate-constitutions	gtp3Y.F90
subroutine set-cond-or-exp	gtp3D.F90
subroutine set-condition	gtp3D.F90
subroutine set-constituent-reference-state	gtp3G.F90
subroutine set-constitution	gtp3A.F90
subroutine set-default-constitution	gtp3Y.F90
subroutine set-driving-force	gtp3X.F90
subroutine set-emergency-startpoint	gtp3Y.F90
subroutine set-input-amounts	gtp3D.F90
subroutine set-lattice-parameters	gtp3H.F90
subroutine set-metastable-constitutions2	gtp3Y.F90
subroutine set-new-stoichiometry	gtp3A.F90
subroutine set-phase-amounts	gtp3Y.F90
subroutine set-phase-status-bit	gtp3G.F90
subroutine set-putfun-constant	gtp3F.F90
subroutine set-reference-state	gtp3A.F90
subroutine set-uniquac-species	gtp3B.F90
subroutine set-unit	gtp3G.F90
subroutine setendmemarr	gtp3X.F90
subroutine setpermolebit	gtp3H.F90
subroutine shiftcompsets	gtp3Y.F90
subroutine sort-ionliqconst	gtp3B.F90
subroutine sortcoeffs	gtp3Z.F90
subroutine sortinphtup	gtp3Y.F90
subroutine state-variable-val	gtp3F.F90
subroutine state-variable-val3	gtp3F.F90
subroutine store-element	gtp3B.F90
subroutine store-putfun	gtp3F.F90
subroutine store-putfun-old	gtp3F.F90
subroutine store-tpconstant	gtp3Z.F90
subroutine store-tpfun	gtp3Z.F90
subroutine store-tpfun-dummy	gtp3Z.F90
subroutine subrefstates	gtp3C.F90
subroutine sumprops	gtp3F.F90
subroutine suspend-composition-set	gtp3B.F90

Name	File
subroutine suspend-phases-implicitly	gtp3G.F90
subroutine suspend-somephases	gtp3B.F90
subroutine suspend-species-implicitly	gtp3G.F90
subroutine suspend-unstable-sets	gtp3B.F90
subroutine svfunread	gtp3E.F90
subroutine svfunsave	gtp3E.F90
subroutine switch-compsets2	gtp3Y.F90
subroutine tabder	gtp3X.F90
subroutine tdbrefs	gtp3B.F90
subroutine termterm	gtp3G.F90
subroutine todo-after-found-equilibrium	gtp3Y.F90
subroutine todo-before	gtp3Y.F90
subroutine tpf2c	gtp3Z.F90
subroutine tpf2cx	gtp3Z.F90
subroutine tpfun-deallocate	gtp3Z.F90
subroutine tpfun-geinstein	gtp3Z.F90
subroutine tpfun-init	gtp3Z.F90
subroutine tpfun2coef	gtp3Z.F90
subroutine uniquac-model	gtp3X.F90
subroutine verify-phase-varres-array	gtp3Y.F90
subroutine write-pdbformat	gtp3E.F90