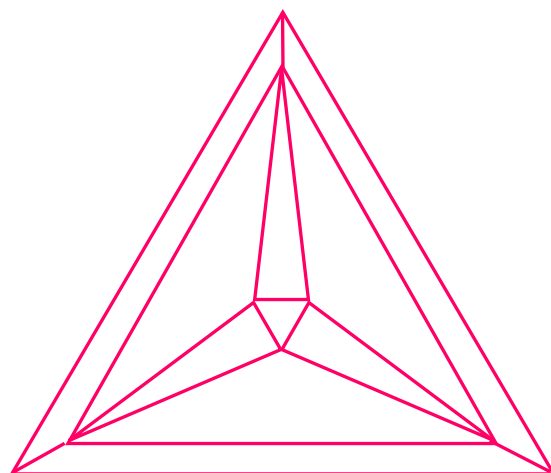


Thermodynamic Calculation Interface

TC-API

(Version 5.0)

Programmer's Guide and Examples



Thermo-Calc Software AB
Stockholm Technology Park, Björnåsvägen 21
SE-113 47 Stockholm, Sweden

May, 2008

Thermodynamic Calculation Interface

TC-API

(Version 5.0)

Programmer's Guide and Examples

Developer: Thermo-Calc Software AB
Stockholm Technology Park
Björnåsvägen 21
SE - 113 47 Stockholm, Sweden

Copyright © **1995-2008** Foundation of Computational Thermodynamics
Stockholm, Sweden

Copyright:

The Thermo-Calc and DICTRA software are the exclusive copyright properties of the STT Foundation (Foundation of Computational Thermodynamics, Stockholm, Sweden). All rights are reserved worldwide!

Thermo-Calc Software AB has the exclusive rights for further developing and marketing all kinds of versions of Thermo-Calc and DICTRA software/database/interface packages, worldwide.

This *TC-API Programmer's Guide and Examples*, as well as all other related documentation, is the copyright property of Thermo-Calc Software AB.

It is absolutely forbidden to make any illegal copies of the software, databases, interfaces, and their manuals (User's Guide and Examples Book) and other technical publications (Reference Book and Technical Information). Any unauthorized duplication of such copyrighted products, is a violation of international copyright law. Individuals or organizations (companies, research companies, governmental institutes, and universities) that make or permit to make unauthorized copies may be subject to prosecution.

The utilization of the Thermo-Calc and DICTRA software/database/interface packages and their manuals and other technical information are extensively and permanently governed by the *Thermo-Calc Software AB END USER LICENSE AGREEMENT (EULA)*, which is connected with the software.

Disclaimers:

Thermo-Calc Software AB and the STT Foundation reserve the rights to further developments of the Thermo-Calc and DICTRA software and related software/database/interface products, and to revisions of their manuals and other publications, with no obligation to notify any individual or organization of such developments and revisions. In no event shall Thermo-Calc Software AB and the STT Foundation be liable to any loss of profit or any other commercial damage, including but not limited to special, consequential or other damage.

Please visit the Thermo-Calc Software web site (www.thermocalc.se) for any modification and/or improvement that have been incorporated into the programs, or for any amendment that have made to the contents of the various User's Guides and to the FAQ lists and other technical information publications.

Acknowledgement of Copyright and Trademark Names:

Various third-party software names that are protected by copyright and/or trademarks are mentioned for descriptive purposes, within this User's Guide and other documents of the Thermo-Calc and DICTRA software/database/interface packages. Due acknowledgement is herein made of all such protections.

Contents

1	INTRODUCTION.....	5
1.1	STRUCTURE OF THE TC-API	5
1.2	THE DIFFERENCE BETWEEN THE TC-API AND TQ-I	6
2	HOW TO INSTALL AND RUN THE TC-API.....	8
3	DESCRIPTION OF FUNCTIONS IN THE TC-API	9
3.1	TC_ROOT	10
3.2	TC_DATABASE.....	10
3.3	TC_SYSTEM	12
3.4	TC_UTIL.....	15
3.5	TC_GES5	15
3.6	FUNCTIONS SPECIFIC FOR TCW.....	16
3.6.1	TC_SYSTEM.....	16
3.6.2	TC_POST.....	17
3.6.3	TC_BINARY.....	19
3.6.4	TC_TERNARY	19
3.6.5	TC_SCHEIL.....	20
4	PROGRAMMING EXAMPLES	22
4.1	EXAMPLE 1	22
4.2	EXAMPLE 2	26
4.3	EXAMPLE 3	27
4.4	EXAMPLE 4	30
4.5	EXAMPLE 5	33
4.6	EXAMPLE 6	36

1 Introduction

Thermo-Calc is a general software package for manipulation of thermodynamic quantities and calculation of phase equilibria in multicomponent systems. The software can be applied to many problems in the field of materials science. Thermo-Calc comes in two modifications: Thermo-Calc Classic (TCC) and Thermo-Calc Windows (TCW). In TCC the user communicates with the program via specific commands while in TCW the user is presented with a graphical user interface. Generally, TCC offers greater flexibility to the user compared to TCW. On the other hand TCW could be considered more user-friendly.

It is also possible to use Thermo-Calc in user-written application programs through the so-called TC-Interfaces. Currently, there are three TC-Interfaces available: TQ-I, TC-API and the TC MATLAB Toolbox. The basic idea behind these interfaces is that application programmers can retrieve thermodynamic data and calculate phase equilibria directly from within their programs without having to implement various sophisticated models or mathematical routines, or have extensive knowledge of Thermo-Calc. In this document the Thermo-Calc Application Programming Interface (TC-API) is described.

1.1 Structure of the TC-API

Since the Thermo-Calc software was initially written in FORTRAN the TC-API can be regarded as composed of two parts (Fig. 1): an intermediate layer written in FORTRAN followed by a part written in the C programming language. This is the layer seen by the application programmer. The FORTRAN layer consists of basic routines for handling the passing of e.g. strings and other data types between Thermo-Calc and the user-written program. The TC-API has been developed with the intention of making as many features as possible found in Thermo-Calc available also to the application programmer. The TC-API is most suitable for developing windows applications involving advanced thermodynamic and/or kinetic calculations and an example of this is the TCW software.

As can be seen in Fig. 1 the TC-API also forms an important part of the TC MATLAB Toolbox. The MATLAB software can communicate with programs written in e.g. C via so-called MEX-functions. The TQ-Interface (TQ-I) is independent of the TC-API and contains a large number of routines especially designed for time critical, computationally intensive application programs.

1.2 The difference between the TC-API and TQ-I

Since the TC-API is written in C, linking and communication with programming languages other than FORTRAN is more convenient compared to the TQ-I. Also, the TC-API includes more of the functionality found in Thermo-Calc. It offers access to most of the commands in the TDB, POLY-3 and POLY-3 Postprocessor modules and some important commands in the GES5 module.

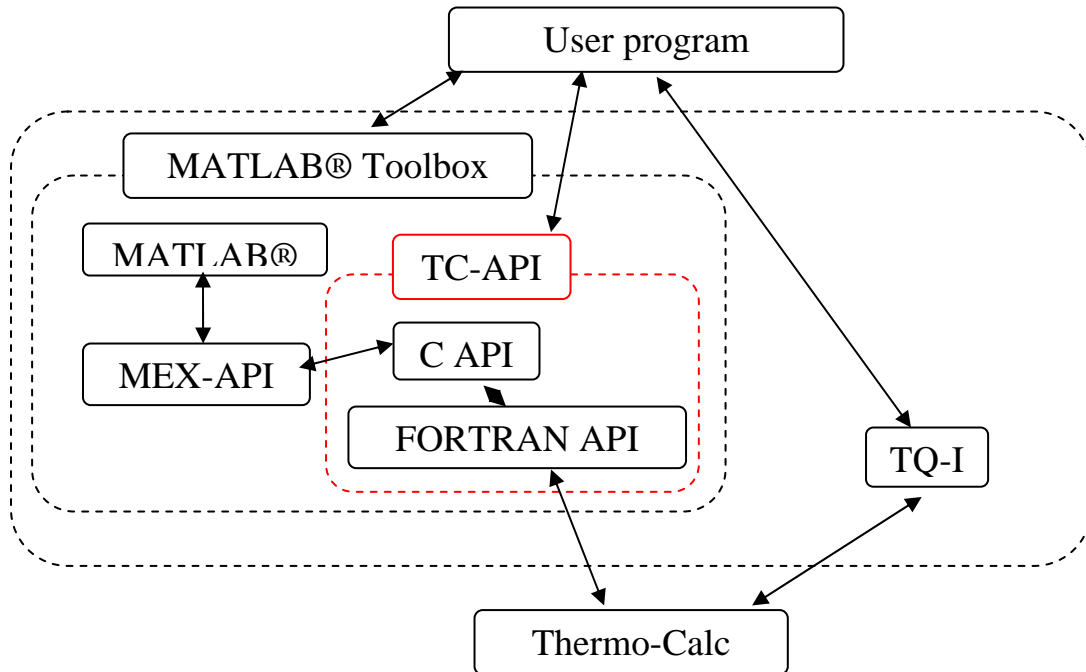


Fig. 1. Illustration of the different TC-Interfaces available for Thermo-Calc and their dependencies.

An example of the use of the programming interface is a simple application that calculates the solidus and liquidus temperatures in an Fe-Cr-Ni-C alloy. The application is a window based program that requires the composition in weight percent and returns the temperatures in degrees Celsius, see Fig. 2.

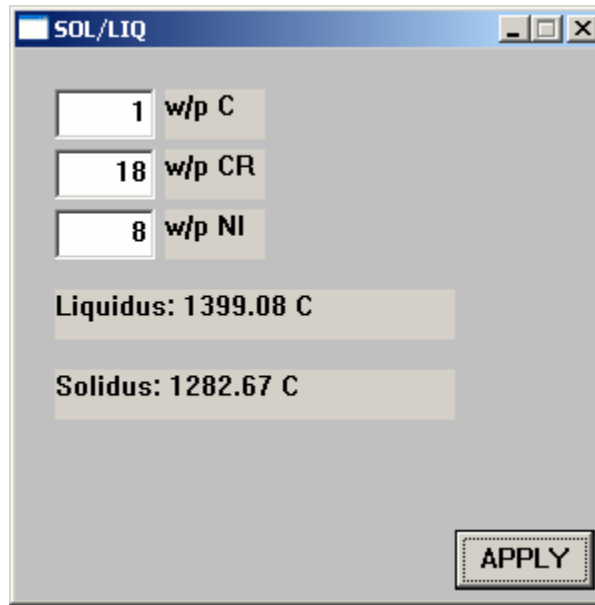


Fig. 2. An example of a window based application program created with the TC-API. The solidus and liquidus temperature is calculated for an Fe-Cr-Ni-C alloy.

The thermodynamic data is stored in a POLY-3 binary file which can be created from either TCC or TCW. The calculation procedure should not be regarded as the correct way of obtaining these temperatures, but should only be seen as a simple example of the use of the programming interface. The full source code for setting up the window, buttons and output fields is specific to the Windows API and can be found in section 4.6

2 How to install and run the TC-API

The TC-API is intended to be used for programming under PC-Linux and PC-Windows (XP/VISTA) environments. To be able to use the TC-API licensed versions of TCC or TCW together with suitable databases are required.

Installation of the TC-API is straightforward. Simply unpack the compressed file found on the installation CD to any location. This should leave a folder (named TC-API) on the chosen location containing the dynamic TC-API, version 5, tested with version S of TCC. A number of examples (1-6) are also included together with command files to compile and link all of the examples. All examples may be generated by running the “mkallexamples” command file which in turn will call the specific command files to generate the executables.

Before using the TC-API you need to run the *VFRUN66BI.exe* file, which will install the necessary Visual Fortran runtime libraries. The full source code of each example is found in section 4 in this guide

After installation a folder with the following files should be present:

<i>examples.txt</i>	text file with basically the same information as in this section
<i>README.lst</i>	=>examples.txt
<i>USER_TCAPI.pdf</i>	this guide
<i>VFRUN66BI.exe</i>	run this file to install necessary VF runtime libraries
<i>tcapi.h</i>	header file needed to compile examples
<i>tcdlls.dll</i>	dynamic version library
<i>tcdlls.lib</i>	library headers
<i>exampleX.c</i>	X=1, 2, 3, 4, 5, 6. Source code for the attached examples
<i>mkexampleX.bat</i>	X=1, 2, 3, 4, 5, 6. Files to generate a dynamic version of each example.
<i>mkallexamples.bat</i>	command file to generate all examples
<i>clean.bat</i>	command file to clean up the directory

3 Description of functions in the TC-API

The functions available in the TC-API have been divided into eight groups depending on their purposes. The groups are:

TC_ROOT	refers to general information and miscellaneous commands
TC_DATABASE	refers to information and commands in the database module
TC_SYSTEM	refers to information and commands in the POLY-3 module
TC_POST	refers to information and commands in the POST module
TC_BINARY	refers to commands for calculations of binary diagrams
TC_UTIL	refers to various commands e.g. path information
TC_GES	refers to information and commands in the GES5 module
TC_SCHEIL	refers to commands in the SCHEIL module

Some groups of functions are specific for use by the TCW (Thermo-Calc Windows) program and should not be of interest for the normal user of the TC-API e.g. functions in TC_POST, TC_BINARY and TC_SCHEIL. These functions will be described in section 3.6.

In order to avoid conflict with reserved names in C and C++ every function name begins with “tc_” e.g. tc_poly3. Furthermore, the following conventions, regarding variable types, have been used:

void	nothing
integer	word integer (32 bit)
string	character array
string array	array of string
double	double word floating point value
logical	logical value (word integer)

3.1 TC_ROOT

Type	Name	Arguments	Description
void	tc_init_tc_root	none	Initializes the Thermo-Calc system, must be called prior to anything else.
void	tc_stop	none	Terminates the application in a brutal way
integer	tc_database	address of string array: names_of_databases integer: length of string in array	Returns the number of databases in the system and their names in "names_of_databases"
void	tc_version	address of string: version_name integer: length of string	Returns the version of Thermo-Calc in "version_name"
void	tc_open_database	string: name_of_database	Opens the named database "name_of_database"
void	tc_append_database	string: name_of_database	Appends the named database "name_of_database"
void	tc_poly3	address of string: command	Sends a command to POLY-3 module as defined in the argument "command"
void	tc_poly3t	address of string: command	Sends an unterminated command to POLY-3 as defined in the argument "command"

3.2 TC_DATABASE

Type	Name	Arguments	Description
integer	tc_nr_of_databases	none	Returns the number of databases defined in the system.
string	tc_database_name	none	Returns the full name of the currently selected database
string	tc_database_description	none	Returns the description of the currently selected database
string	tc_database_filename	none	Returns the full filename of the currently selected database
void	tc_database_info	address of string: info integer: length of string	Returns the information on the selected database in "info". NOTE: the records are terminated with the "" character.
integer	tc_element	address of string array: elements integer: length of string array	Returns the number of elements in the database and their names in "elements"
integer	tc_nr_of_elements	none	Returns the number of elements of the currently selected database
void	tc_element_select	address of string: element name	Selects "element name" in the currently selected database.
void	tc_element_reject	address of string: element name	Rejects "element name" in the currently selected database.
logical	tc_is_element_selected	address of string: element name	Returns true if the element name is selected in the database.
void	tc_reject_system	none	Rejects the currently defined system in the database and reinitiates the system.
integer	tc_phase	address of string array: phases integer: : length of string in phases	Returns the number of phases in the system with the selected elements. NOTE: the routine returns the number of

			all available phases.
integer	tc_nr_of_phases	none	Returns the number of phases in the system with the selected elements. NOTE: the routines always returns the same number, which are the available phases with the current element selection.
void	tc_phase_select	address of string: phase_name	Selects the phase in "phase_name"
void	tc_phase_reject	address of string: phase_name	Rejects the phase in "phase_name"
logical	tc_is_phase_selected	address of string: phase_name	Returns true if the phase in "phase_name" is selected
logical	tc_is_phase-included	address of string: phase_name	Returns true if the phase in "phase_name" is included in the current database.
void	tc_get_data	none	Executes the database command "GET_DATA"
void	tc_phasename_to_index	address of string: phase_name integer: phase_index	Returns the index of "phase_name" in "phase_index". NOTE: the get_data command must be called prior to using this routine.
void	tc_index_to_phasename	integer: phase_index address of string: phase_name length of string in phases_name	Returns the name of "phase_index" in "phase_name". NOTE: the get_data command must be called prior to using this routine.
integer	tc_phase_constituents	address of string: phase_name address of integer array: constituents address of string array: names of constituents integer: length of string in names of constituents address of double array: number of sites	Returns the number of sublattices in the phase, the number of constituents on each sublattice in "constituents", the name of the selected species on each sublattice one after each other in "species names" and the "number of sites" on each sublattice.
integer	tc_phase_all_constituents	address of string: phase_name address of integer array: constituents address of string array: names of constituents integer: length of string in names of constituents address of double array: number of sites	Returns the number of sublattices in the phase (including phases with the status SUSPENDED), the number of constituents on each sublattice in "constituents", the name of the selected species on each sublattice one after each other in "species names" and the "number of sites" on each sublattice.
integer	tc_get_references	address of string array: references integer: length of string in references	Returns the number of references in "references", the array must be of a size of at least 210 elements and with a string of 400 characters.
void	tc_reject_constituent	address of string: phase_name integer: sublattice address of string: constituents	Rejects the constituent "constituent" on sublattice "sublattice" from phase "phase_name".
void	tc_restore_constituent	address of string: phase_name integer: sublattice address of string: constituents	Restores the constituent "constituent" on sublattice "sublattice" from phase "phase_name".

3.3 TC_SYSTEM

Type	Name	Arguments	Description
integer	tc_list_phase	address of string array: phase_name integer: length of string in array	Returns the number of phases in the system and their names in "phase_name".
string	tc_phase_status	address of string: phase_name	Returns the status of "phase_name" where status may be one of "FIXED", "SUSPENDED" or "ENTERED"
void	tc_set_phase_status	address of string: phase_name address of string: status double: value	Sets the status of "phase_name" to "status" to one of "FIXED", "SUSPENDED", DORMANT" or "ENTERED".
integer	tc_list_species	address of string: species name integer: length of string in array	Returns the number of species in the system and their names in "species_name".
string	tc_species_status	address of string: species name	Returns the status of "species_name" where status may be one of "ENTERED" or "SUSPENDED"
void	tc_set_species_status	address of string: species name address of string: status	Sets the status of "species_name" to "status" to one of "ENTERED" or "SUSPENDED"
integer	tc_list_component	address of string: component name integer: length of string in array	Returns the number of components in the system and their names in "component_name".
string	tc_component_status	address of string: component_name	Returns the status of "component_name" where status may be one of "ENTERED" or "SUSPENDED"
void	tc_set_component_status	address of string: component_name address of string: status	Sets the status of "component_name" to "status" to one of "ENTERED" or "SUSPENDED"
void	tc_define_components	address of string array: component_name integer: length of string in array integer: number of components	Redefines the components in the system to the components in "component_name".
void	tc_set_reference_state	address of string: component_name address of string: phase_name double: temperature double: pressure	Sets the reference state for component in "component_name" to the phase in "phase_name" with temperature "temperature" and pressure "pressure", temperature may be set to be less than zero, resulting in "*" as reference temperature.
void	tc_get_reference_state	address of string: component_name address of string: phase_name integer: length of phase_name address of double: temperature address of double: pressure	Retrieves the reference state for component in "component_name" to the phase in "phase_name" with temperature "temperature" and pressure "pressure", temperature may be set to be less than zero, resulting in "*" as reference temperature.
integer	tc_list_conditions	address of string array: conditions integer: length of string in array	Returns the number of conditions set and their values in "conditions"
void	tc_set_condition	address of string: condition double: value	Sets a condition for the expression in "condition" to value in "value".
void	tc_delete_condition	address of string: condition	Deletes the condition for the expression in "condition".
void	tc_set_phase_addition	address of string: phase_name	Sets the addition "addition" to the Gibbs

		double: addition	energy function for phase "phase name".
integer	tc_degrees_of_freedom	none	Returns the degrees of freedom in the system. This must be zero in order to perform an equilibrium calculation.
void	tc_enter_symbol	address of string: symbol name address of string: type of symbol integer: argument type integer: integer argument double: double argument address of string: string argument	Enters a symbol in the system, the symbol type may be one of "CONSTANT", "VARIABLE", "FUNCTION" or "TABLE", "argument type" defines which of the following arguments will be used, 1 indicates the integer argument, 2 the double argument and 3 the string argument.
void	tc_delete_symbol	address of string: symbol name	Deletes a symbol in the system.
integer	tc_list_symbols	address of string array: symbols integer: array: length of string in array address of integer array: type of symbol	Returns the number of defined symbols in the system with their expression and value in "symbols" and their corresponding type in "type of symbol", where the type may be one of 1="CONSTANT", 2="VARIABLE" 3="FUNCTION" 4="TABLE"
void	tc_set_axis_variable	integer: axis number address of string: condition double: min value double: max value double: step integer: type of axis	Sets the stepping axis "axis number" to condition "condition" with min, max and step. Type of axis may be one of 0=linear or 1=logarithmic
integer	tc_list_axis	address of string array: condition integer: length of string in array address of double array: min address of double array: max address of double array: step address of integer array: number	Returns the number of defined axis in the system and their conditions in "conditions", their range, increment and number in "min", "max", "step" and "number" respectively.
integer	tc_list_initial_equilibrium	address of string array: initial_equilibrium integer: length of string in array	Returns the number of stored initial equilibria in the system with their description in "initial equilibrium".
void	tc_add_initial_equilibrium	integer: direction	Adds the current equilibrium to the list of initial equilibria with starting stepping direction in "direction" where direction may take values from -2 to 2, a value of zero will generate the option default
void	tc_compute_equilibrium	none	Computes the equilibrium in POLY-3 using the currently set conditions
void	tc_compute_equilibrium_try_hard	none	Same as tc_compute_equilibrium, but can be used when having trouble finding a stable phase set
void	tc_save_poly3_file	address of string: filename	Stores/overwrites the current workspace in POLY-3 on the file "filename".
void	tc_read_poly3_file	address of string: filename	Loads the workspace from file "filename" in POLY-3.
integer	Tc_check_poly3_file	address of string: filename	Checks the workspace on file "filename" in POLY-3, returns 0=file does not exist, 1=file exists, 2=file exists and has mapped info.
logical	tc_error	address of integer: error number address of string: error message integer: string length of error message	Returns true if an error has been set, returning the error number in "error number" and its corresponding message in "error message"
void	tc_reset_error	none	Resets the error if an error has been set
double	tc_get_value	address of string: symbol/state variable	Retrieves the symbol or state variable value from the POLY-3 module.

integer	tc_stable_phases	address of string array: phases integer: length of string in array	Returns the number of stable phases in an equilibrium and their names in "phases".
integer	tc_phase_structure	address of string: phase name address of integer array: constituents address of string array: species names integer: length of string species names address of double array: number of sites	Returns the number of sublattices in the phase, the number of constituents on each sublattice in "constituents", the name of the species on each sublattice one after each other in "species names" and the number of sites in "number of sites".
integer	tc_get_table_values	address of string: table name address of integer: status address of string: symbol names length of string in array integer: address of double array: values	Returns the number of values retrieved from a table "table name", first value is returned if "status"=0, next value if "status"=1, "staus" is updated to -1 if no more values are available. The symbol names and their values are returned in "symbol names" and "values".
integer	tc_nr_of_consituents_in_phase	address of string: phase name	Returns the number of constituents in a phase
void	tc_create_new_equilibrium	integer: equilibrium number	Creates a new equilibrium in POLY-3 with number "equilibrium number".
void	tc_select_equilibrium	integer: equilibrium number	Selects an equilibrium in POLY-3 with number "equilibrium number".
void	tc_set_start_value	string: state variable double: start value	Sets a starting value for the "state variable" to "start value".
void	tc_para_equilibrium	string: first phase string: second phase	Computes the para-equilibrium between "first phase" and "second phase", the command requires that the degrees of freedom are zero prior to executing this command and that carbon "C" is present in the system.
void	tc_get_minimization_option	address of integer: global_flag address of integer: max_gridpoints address of integer: restart_flag address of integer: hessian_flag address of integer: frequency	Returns the current settings for global minimization
void	tc_set_minimization_option	address of integer: global_flag address of integer: max_gridpoints address of integer: frequency address of integer: mesh_flag	Sets parameters for global minimization

3.4 TC_UTIL

Type	Name	Arguments	Description
void	tc_delete_file	Address of string: filename	Deletes a filename "filename"
void	tc_get_temp_filename	Address of string: filename integer: length of string	Returns a unique filename in "filename"
void	tc_where_am_i	Address of string: path	Returns the path of the running program in "path"
logical	tc_check_license	Address of string: license key address of string: error message integer: length of string error message	Returns true if the checked license is available and valid, currently accepts: TC_DLL, TC_GUI and TC_TC4U
void	tc_get_environment_variable	Address of string: name address of string: value integer: length of value string	Returns the value of "name" in string "value"

3.5 TC_GES5

Type	Name	Arguments	Description
void	tc_ges5c	address of string: command	Sends a command to the GES5 module as defined in the argument "command"
void	tc_get_ges5_parameter	address of string: parameter name address of string: expression integer: length of string	Retrieves the expression of a parameter name
void	tc_enter_ges5_parameter	address of string: expression address of string: parameter name	Enters a parameter expression
void	tc_get_derivatives	address of string: phase name address of array of double: arr1 address of array of double: arr2	Returns Gm and the first derivatives with respect to site-fractions in "arr1" and the second derivatives in "arr2" as GM.Y1.Y1, GM.Y1.Y2, GM.Y2.Y2, GM.Y1.Y3, GM.Y2.Y3 ... GM.YN.YN

3.6 Functions specific for TCW

3.6.1 TC_SYSTEM

void	tc_map	none	starts mapping
void	tc_map_reentrant	address of integer: kode address of integer number of phases address of string: phases integer: string length of phases	maps reentrant, kode must be 0 at start and returns a value less than 0 when finished, returns a list of mapped phases in "phases". Use the returned value of "kode" for each subsequent call.
void	tc_step	integer: option	start stepping with option, where option may be: 0="no argument", 1="NORMAL", 2="INITIAL_EQUILIBRIA" 3="EVALUATE_VARIABLES" 4="SEPARATE_PHASES"
void	tc_step_reentrant0	address of integer: kode address of integer: number of phases address of string: phases integer: string length of phases	step reentrant, "kode" must be 0 at start and returns a value less than 0 when finished, returns a "number of phases". Use the returned value of "kode" for each subsequent call.
void	tc_step_reentrant	integer: option address of integer: kode address of integer: number of phases address of string: phases integer: string length of phases address of string: variables address of string: interstitial components integer: 0 for allowing bcc to fcc transformation and 1 for not allowing bcc to fcc transformation	step reentrant with option, where option may be: 0="no argument", 1="NORMAL", 2="INITIAL_EQUILIBRIA" 3="EVALUATE_VARIABLES" 4="SEPARATE_PHASES" 7="MIXED-SCHEIL" "kode" must be 0 at start and returns a value less than 0 when finished, returns a "number of phases". Use the returned value of "kode" for each subsequent call. If using option 3 the variables to be evaluated must be specified in "variables".

3.6.2 TC_POST

Type	Name	Arguments	Description
void	tc_set_diagram_axis	integer: axis number address of string: variable name	Sets the diagram axis in POLY-3 postprocessor "axis number" indicates X=1, Y=2 and Z=3, the variable for this axis is set with "variable name". NOTE: "variable name" should have all necessary parameters for the command.
string	tc_get_diagram_axis	integer: axis number	Returns the diagram axis in POLY-3 postprocessor "axis number" indicates X=1, Y=2 and Z=3.
void	tc_set_axis_text	integer: axis number integer: status address of string axis text	Sets the axis text in POLY-3 postprocessor "axis number" indicates X=1, Y=2 and Z=3, status=1 indicates if automatic text should be used.
void	tc_set_diagram_title	address of string: title text	Sets the title text in POLY-3 postprocessor.
string	tc_get_diagram_title	none	Returns the title text in POLY-3 postprocessor.
string	tc_get_axis_text	integer: axis number address of integer: status	Returns the text set on an axis in POLY-3 postprocessor "axis number" indicates X=1, Y=2 and Z=3, "status" is set to 1 if automatic text should be used.
void	tc_set_scaling_status	integer: axis number integer: auto double: min double: max	Sets the scaling status for axis in POLY-3 postprocessor "axis number" indicates X=1, Y=2 and Z=3. Auto scaling is indicated by setting "auto" to 1, if "auto" is 0 the manual scaling is assumed with min max values in "min" and "max".
void	tc_get_plot_box	address of short integer: real x min coordinate address of short integer: real y min coordinate address of double: value min x coordinate address of double: value min y coordinate address of double: value max x coordinate address of double: value max y coordinate address of short integer: real x max coordinate address of short integer: real y max coordinate	Retrieves the plot box coordinates and the corresponding values.
void	tc_plot_diagram	address of string: output file	Plots a diagram on "output file". The "output file" can contain a valid filename or "SCREEN" (NOTE: SCREEN will not generate an output on screen).
void	tc_make_experimental_file	address of string: output file	Generates an experimental file on "output file".
void	tc_set_plot_format	integer: output device number	Sets the output device number for plotting.
integer	tc_get_plotted_data	array of short integer: x coordinates array of short integer: y coordinates	Returns the number (x,y) coordinates and the x-coordinates in "x", the y-coordinate in "y" and the pen color in "color", a negative value indicates that a move

		array of short integer: color/draw/move	should be performed.
logical	tc_is_gs_installed	None	Returns true is GhostScript is installed
logical	tc_gs_print_file	address of string: filename	Will print a file "filename" using the GhostScript printing facilities. NOTE: GhostScript package must be installed and in the path.
logical	tc_gs_dump_image	integer: format integer: resolution address of string: temp filename address of string: output filename	Will create an image file using the GhostScript conversion facilities. Resolution 1: low, 2 or 3 medium, 4 high, format: 1 png, 2: bmp, 3: pdf, 4: jpeg, 5: tiff, 6: postscript and 7: experimental file.
void	tc_set_diagram_type	integer: type	Sets the plotted diagram to normal/cartesian if "type" is 1 and to triangular if "type" is 2.
integer	tc_get_diagram_type	none	Returns the type of diagram, normal/cartesian if 1 is returned and triangular if 2 is returned.
void	tc_set_axis_type	integer: axis number integer: axis type	Sets the axis type in POLY-3 postprocessor. "axis numer" indicates X=1, Y=2 and Z=3 to LINear if "axis type=1" LOGarithmic if "axis type=2" and INVerse if "axis type=3".
integer	tc_get_axis_type	integer: axis number	Returns the axis type in POLY-3 postprocessor. "axis numer" indicates X=1, Y=2 and Z=3 to LINear if "axis type=1" LOGarithmic if "axis type=2" and INVerse if "axis type=3".
void	tc_set_tieline_status	integer: number of tielines	Set the number of tielines to be plotted, 0 disables the pot of tielines.
integer	tc_get_tieline_status	none	Return the number of tielines to be plotted.
void	tc_set_label_option	integer: option number	Sets the label option in POLY-3 postprocessor, where 0=none, 1=A, 2=B, 3=C, 4=D, 5=E and 6=F.
void	tc_add_label	double: xvalue double: yvalue	Adds an automatic label at coordinates (xvalue,yvalue).
void	tc_delete_label	double: xvalue double: yvalue	Deletes the label at cordinatees (xvalue,yvalue)
void	tc_amend_label	double: old_xvalue double: old_yvalue double: new_xvalue double: new_yvalue double: angle address of string: string address of string: fontname integer: fontsize	Amends the label which was previously at the coordinates (old_xvalue, old_yvalue) to contain the string "string" with rotation "angle" at coordinates (new_xvalue,new_yvalue).
integer	tc_nr_of_labels	none	Returns the number of labels
logical	tc_get_label	integer: the number of the label to retrieve integer: identifier of label integer: x-coordinate of label integer: y-coordinate of label integer: angle integer: fontsize address of string: fontname integer: length of string address of string: fontsize	Returns information about a specific label

integer	tc_label_info	integer: length of string address of string: info integer: length of string address of double array: x-coordinate address of double array :y-coordinate	Returns the number of labels, their coordinates and the label text
logical	tc_check_if_add_label	None	Returns true if add_label can be used.
void	tc_delete_all_labels	none	Deletes all labels.
void	tc_set_plot_of_conditions	integer: status	Sets the status of plot of conditions, 0 turns the function off and 1 turns it on.
void	tc_set_plot_of_raster	integer: status	Set the status of plot of raster, 0 turns the function off and 1 turns it on.
integer	tc_get_plot_of_conditions	none	Return the status of plot of conditions, 0 indicates that it is off and 1 that it is on.
Void	tc_list_data_table	Address of string: filename	Sorts data and writes it to an Excel fiel

3.6.3 TC_BINARY

Type	Name	Arguments	Description
void	tc_binary_diagram	address of string: first element address of string: second element double: temperature address of string: database integer: diagram type	Calculates a binary diagram with “first element” and “second element” from “database”. Phase diagram is calculated if “temperature” <0, G-curves if “temperature” >1 and a phase fraction diagram for the given fraction of “second element” if “temperature” <0 <1. “diagram type” also indicates the type of diagram to be calculated: 1=phase diagram, 2=G-curves, 3=A-curves, 4=phase fraction.
void	tc_binary_diagram2	address of string: first element address of string: second element double: temperature address of string: database integer: diagram type	Identical to binary_diagram though database information is assumed to be managed prior to call this routine.
integer	tc_binary_elements	address of string: element address of array of strings: elements integer: length of string	Returns the nnumber of elements and the elements in “elements” that may be used to calculate a binary phase diagram using element “element”.
void	tc_make_color_bin	address of string: first exp-file address of string: second exp-file	Generates a second exp-file from first exp-file with closed filled phase fields an appends the second exp-file with APPEND-EXPERIMENTS.
void	tc_binary_store_file	address of string: filename	Sets the filename where the binary module store calculation results.

3.6.4 TC_TERNARY

Type	Name	Arguments	Description
void	tc_ternary_diagram	address of string: first element address of strings: second element address of string: third element double: temperature double: minimum temperature double: maximum temperature double: temperature interval address of string: database integer: diagram type	First element Second element Third element Isothermal section temperature Min. temperature for isothermal lines Max. temperature for isothermal lines Temperature interval for isothermal lines Database name 1=isothermal section without stability check on
void	tc_ternary_diagram2	address of string: first element address of strings: second element address of string: third element double: temperature double: minimum temperature double: maximum temperature double: temperature interval address of string: database integer: diagram type	Identical to tc_ternary_diagram except that the retrieval of thermodynamic data is assumed to have been managed prior to the call of this routine
void	tc_ternary_elements	address of string: element address of string: element address of array of strings: elements integer: length of string	Empty or first selected element Empty or second selected element All, second, or third elements of assessed ternary system available in the database.
void	tc_ternary_store_file	address of string: filename	Sets the filename for storing calculation results.
void	tc_get_line_info	double: x double: y address of string: string integer: length of string	Returns info about a point in the diagram with coordinates x,y in string.

3.6.5 TC_SCHEIL

Type	Name	Arguments	Description
void	tc_scheil	integer: number of components address of string: dependent component address of strings: all components integer: string length of all components address of string: type of conditions array of double: conditions double: start temperature double: step in temperature address of string: POLY-3 filename	Sets up a scheil simulation. The temperature just above the liquidus is returned in "start temperature"
void	tc_scheil_get_variables	address of string: type of conditions address of string: variables integer: string length of variables double: start temperature double: step in temperature	Get the variables to use in the "step_reentrant" command. "start temperature" should be the temperature return from scheil.
void	tc_scheil_get_axis	integer number of variables address of array of strings: mnemonic	Get auxiliary variables in Scheil plot.

		integer: string length of mnemonic address of array of strings: explanation integer: string length of explanation	
void	tc_scheil_set_axis	address of string: axis variable integer: axis number	Sets the plot specific scheil plot axis for axis number 1 or 2, where 1 corresponds to x-axis and 2 y-axis. Axis variable must be a valid variable as returned from scheil_get_axis.

4 Programming examples

As already mentioned in section 2 executables for the included examples files can be generated by running the individual batch programs (mkexampleX.bat, where X=1, 2, 3, 4, 5, 6). The file mkallexamples.bat will generate executables for all examples. The complete source code for each example is presented below.

4.1 Example 1

```
/* *****  
/* Calculates an equilibrium in the Fe-Cr-C system and retrieves certain */  
/* quantities. */  
/* */  
/* */  
/* *****  
  
#include <stdio.h>  
#include <string.h>  
#include "tcapi.h"  
  
int main( int argc, char **argv)  
{  
    char *line;  
    double value;  
    char error[80];  
    int ierr;  
  
    /* Initialize the system */  
    tc_init_tc_root();  
    ierr=0;  
    if ( tc_error(&ierr,error,sizeof(error)) ) {  
        fprintf(stdout,"error: %d : %s\n",ierr,error);  
        tc_reset_error();  
    }  
  
    /* Open database */  
    line="SSOL2";  
    tc_open_database(line);  
    ierr=0;  
    if ( tc_error(&ierr,error,sizeof(error)) ) {  
        fprintf(stdout,"error: %d : %s\n",ierr,error);  
        tc_reset_error();  
    }  
  
    /* Select the elements */  
    line="FE";  
    tc_element_select(line);  
    ierr=0;  
    if ( tc_error(&ierr,error,sizeof(error)) ) {  
        fprintf(stdout,"error: %d : %s\n",ierr,error);  
        tc_reset_error();  
    }  
  
    line="CR";  
    tc_element_select(line);  
    ierr=0;
```

```

if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

line="C";
tc_element_select(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

/* Get the data from the database */
tc_get_data();
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

/* Set the necessary conditions */
line="W(CR); value=0.18;
tc_set_condition(line,value);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

line="W(C); value=0.05;
tc_set_condition(line,value);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

line="T; value=1273;
tc_set_condition(line,value);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

line="N; value=1;
tc_set_condition(line,value);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

line="P; value=101325;
tc_set_condition(line,value);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

```

```

/* Compute the equilibrium */
tc_compute_equilibrium();
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

tc_compute_equilibrium();
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}

/* Retrieve the desired value */

line="MU(CR)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);

line="X(CR)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);

line="NPM(BCC)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);

line="NPM(FCC)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);

line="NPM(CEM)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);

```

```
line="NPM(M7C3)";
value=tc_get_value(line);
ierr=0;
if ( tc_error(&ierr,error,sizeof(error)) ) {
    fprintf(stdout,"error: %d : %s\n",ierr,error);
    tc_reset_error();
}
fprintf(stdout," %s : %g\n",line,value);
}
```

4.2 Example 2

```
/* **** */
/* Calculates diffusion coefficients in the Fe-Cr-C system. */
/* */
/* */
/* **** */

#include <stdio.h>
#include <string.h>

#include "tcapi.h"

int main( int argc, char **argv)
{
    char *hlin1,*hlin2;
    int i;
    double value;

    tc_init_tc_root();

    hlin1="SSOL2";      tc_open_database(hlin1);
    hlin1="FE";   tc_element_select(hlin1);
    hlin1="CR";   tc_element_select(hlin1);
    hlin1="C";    tc_element_select(hlin1);
    get_data();

    hlin1="PFRIB";      tc_append_database(hlin1);
    hlin1="FE";   tc_element_select(hlin1);
    hlin1="CR";   tc_element_select(hlin1);
    hlin1="C";    tc_element_select(hlin1);
    get_data();

    hlin1="T";          value=1000.0; tc_set_condition(hlin1,value);
    hlin1="P";          value=101325.0;   tc_set_condition(hlin1,value);
    hlin1="N(FE)+N(CR)=1";  value=1.0;   tc_set_condition(hlin1,value);
    hlin1="N(C)";        value=0.01;   tc_set_condition(hlin1,value);
    hlin1="N(CR)";       value=0.1;    tc_set_condition(hlin1,value);

    hlin1="*";   hlin2="SUSPENDED";  value=0.0;
    tc_set_phase_status(hlin1,hlin2,value);

    hlin1="FCC"; hlin2="ENTERED";    value=1.0;
    tc_set_phase_status(hlin1,hlin2,value);

    tc_compute_equilibrium();

    hlin1="DC(FCC,C,C,FE)";          value=tc_get_value(hlin1);
    fprintf(stdout," %s = %9g\n",hlin1,value);

    hlin1="DC(FCC,C,CR,FE)";  value=tc_get_value(hlin1);
    fprintf(stdout," %s = %9g\n",hlin1,value);

    hlin1="DC(FCC,CR,C,FE)";  value=tc_get_value(hlin1);
    fprintf(stdout," %s = %9g\n",hlin1,value);

    hlin1="DC(FCC,CR,CR,FE)";  value=tc_get_value(hlin1);
    fprintf(stdout," %s = %9g\n",hlin1,value);

    return 0;}

```

4.3 Example 3

```
/* *****  
/* Calculates an isothermal section in the Fe-Cr-C system. */  
/* */  
/* */  
/* */  
/* *****  
  
#include <stdio.h>  
#include <string.h>  
  
#include "tcapi.h"  
  
int main( int argc, char **argv)  
{  
    char *line;  
    char *hlin1,*hlin2;  
    int i;  
    int linelength=80;  
    double value;  
    char line80[80];  
    char line80b[80];  
    int is;  
  
    tc_init_tc_root();  
    tc_error(&i,line80,linelength);  
    if (i != 0 ) {  
        fprintf(stdout,"error: %d, message:%s\n",i,line80);  
        tc_reset_error();  
    }  
  
    tc_version(line80b,linelength);  
    tc_error(&i,line80,linelength);  
    if (i != 0 ) {  
        fprintf(stdout,"error: %d, message:%s\n",i,line80);  
        tc_reset_error();  
    }  
    fprintf(stdout,"version: %s\n",line80b);  
  
    line="SSOL2";  
    tc_open_database(line);  
  
    tc_error(&i,line80,linelength);  
    if (i != 0 ) {  
        fprintf(stdout,"error: %d, message:%s\n",i,line80);  
        tc_reset_error();  
    }  
  
    hlin1="FE";  
    tc_element_select(hlin1);  
    tc_error(&i,line80,linelength);  
    if (i != 0 ) {  
        fprintf(stdout,"error: %d, message:%s\n",i,line80);  
        tc_reset_error();  
    }  
  
    hlin2="CR";
```

```

tc_element_select(hlin2);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}
hlin2="C";
tc_element_select(hlin2);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

tc_get_data();
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="W(CR)";
value=0.18;
tc_set_condition(hlin2,value);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="W(C)";
value=0.02;
tc_set_condition(hlin2,value);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="T";
value=1000.0;
tc_set_condition(hlin2,value);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="P";
value=101325.0;
tc_set_condition(hlin2,value);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="N";
value=1.0;
tc_set_condition(hlin2,value);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}
}

```

```

hlin2="W(CR)";
tc_set_axis_variable(1,hlin2,0,1,0.025,0);
hlin2="W(C)";
tc_set_axis_variable(2,hlin2,0,1,0.025,0);

line="example3.POLY3";
tc_save_poly3_file(line);

tc_compute_equilibrium();
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

map();
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
    tc_add_initial_equilibrium(0);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
    map();
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
}

////////////////////////////////////

hlin2="W-F CR";
tc_set_diagram_axis(1,hlin2);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

hlin2="W-F C";
tc_set_diagram_axis(2,hlin2);
tc_error(&i,line80,linelength);
if (i != 0 ) {
    fprintf(stdout,"error: %d, message:%s\n",i,line80);
    tc_reset_error();
}

i=5;
tc_set_plot_format(i);
line="example3.ps";
tc_plot_diagram(line);

return 0;
}

```

4.4 Example 4

```
/* ***** */
/* Calculates the Fe-C phase diagram. */
/* */
/* */
/* */
/* ***** */

#include <stdio.h>
#include <string.h>

#include "tcapi.h"

int main( int argc, char **argv)
{
    char *line;
    char *hlin1,*hlin2;
    int i,j;
    int linelength=80;
    double value;
    char line80[80];
    int is,ikod,nph;

    tc_init_tc_root();
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    line="SSOL2";
    tc_open_database(line);

    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin1="FE";
    tc_element_select(hlin1);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin2="C";
    tc_element_select(hlin2);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    tc_get_data();
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
    }
}
```

```

        tc_reset_error();
    }

    hlin2="W(C)";
    value=0.01;
    tc_set_condition(hlin2,value);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
    hlin2="T";
    value=1000.0;
    tc_set_condition(hlin2,value);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
    hlin2="P";
    value=101325.0;
    tc_set_condition(hlin2,value);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
    hlin2="N";
    value=1.0;
    tc_set_condition(hlin2,value);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
}

hlin2="W(C)";
tc_set_axis_variable(1,hlin2,0,1,0.025,0);
hlin2="T";
tc_set_axis_variable(2,hlin2,800,1800,25,0);

tc_add_initial_equilibrium(0);
tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
}

line="foo.POLY3";
tc_save_poly3_file(line);

ikod=12356;
while (ikod >= 0 ) {
    if (ikod==12356) {ikod=0;}
    tc_map_reentrant(&ikod,&nph,line80,linelength);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }
}
else {
    fprintf(stdout,"%d %d %s\n",ikod,nph,line80);
}
}

```

```

    }

////////////////////////////////////

    if ( tc_check_if_add_label() ) {
        fprintf(stdout," add_label can be used.\n"); }
    else {
        fprintf(stdout," add_label can be NOT used.\n"); }

    hlin2="M-F C";
    tc_set_diagram_axis(1,hlin2);
    tc_error(&i,line80,linelength);
    if ( i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin2="T-C";
    tc_set_diagram_axis(2,hlin2);
    tc_error(&i,line80,linelength);
    if ( i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    if ( tc_check_if_add_label() ) {
        fprintf(stdout," add_label can be used.\n"); }
    else {
        fprintf(stdout," add_label can be NOT used.\n"); }

    add_label(0.5,1200);
    tc_error(&i,line80,linelength);
    if ( i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    i=5;
    tc_set_plot_format(i);
    line="example4a.ps";
    tc_plot_diagram(line);

    tc_add_label(0.5,500);
    tc_error(&i,line80,linelength);
    if ( i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    i=5;
    tc_set_plot_format(i);
    line="example4b.ps";
    tc_plot_diagram(line);

    return 0;
}

```

4.5 Example 5

```
/* ***** */
/* Shows how to display certain values in arrays. */
/* */
/* */
/* */
/* ***** */

#include <stdio.h>
#include <string.h>

#include "tcapi.h"

typedef char str8[8];
typedef str8 strvect[100];
typedef int ivect[50];
typedef double rvect[50];

int main( int argc, char **argv)
{
    strvect alle;
    ivect subl;
    rvect nsite;

    char *line;
    char *hlin1,*hlin2;
    int i,j,k,n;
    int linelength=80;
    char line80[80];

    tc_init_tc_root();
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    line="PION";
    tc_open_database(line);

    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin2="FE";
    tc_element_reject(hlin2);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin2="CA";
    tc_element_select(hlin2);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
    }
}
```

```

        tc_reset_error();
    }

    hlin2="SI";
    tc_element_select(hlin2);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin2="O";
    tc_element_select(hlin2);
    tc_error(&i,line80,linelength);
    if (i != 0 ) {
        fprintf(stdout,"error: %d, message:%s\n",i,line80);
        tc_reset_error();
    }

    hlin1="ION";
    fprintf(stdout,"phase_constituents\n");
    n=tc_phase_constituents(hlin1,(int *)subl,(const char *)alle,8,(double
*)nsite);
    i=0;
    fprintf(stdout,"%d\n",n);
    for (j=0; j < n ; j++) {
        fprintf(stdout,"%d :",j+1);
        fprintf(stdout,"%f : ",nsite[j]);
        fprintf(stdout,"%d : ",subl[j]);
        for (k=0; k < subl[j]; k++) {
            fprintf(stdout,"%s ",alle[i]);
            i=i+1;
        }
        fprintf(stdout,"\n");
    }

    fprintf(stdout,"phase_all_constituents\n");
    n=phase_all_constituents(hlin1,(int *)subl,(const char *)alle,8,(double
*)nsite);
    i=0;
    fprintf(stdout,"%d\n",n);
    for (j=0; j < n ; j++) {
        fprintf(stdout,"%d :",j+1);
        fprintf(stdout,"%f : ",nsite[j]);
        fprintf(stdout,"%d : ",subl[j]);
        for (k=0; k < subl[j]; k++) {
            fprintf(stdout,"%s ",alle[i]);
            i=i+1;
        }
        fprintf(stdout,"\n");
    }

    fprintf(stdout,"\nrejecting VA#2 \n");

    hlin2="VA";
    tc_reject_constituent(hlin1,2,hlin2);
    hlin2="ION";

    fprintf(stdout,"phase_constituents\n");
    n=tc_phase_constituents(hlin2,(int *)subl,(const char *)alle,8,(double
*)nsite);
    i=0;
    fprintf(stdout,"%d\n",n);

```

```

for (j=0; j < n ; j++) {
    fprintf(stdout,"%d :",j+1);
    fprintf(stdout,"%f : ",nsite[j]);
    fprintf(stdout,"%d : ",subl[j]);
    for (k=0; k < subl[j]; k++) {
        fprintf(stdout,"%s ",alle[i]);
        i=i+1;
    }
    fprintf(stdout,"\n");
}

fprintf(stdout,"phase_all_constituents\n");
n=tc_phase_all_constituents(hlin2,(int *)subl,(const char
*)alle,8,(double *)nsite);
i=0;
fprintf(stdout,"%d\n",n);
for (j=0; j < n ; j++) {
    fprintf(stdout,"%d :",j+1);
    fprintf(stdout,"%f : ",nsite[j]);
    fprintf(stdout,"%d : ",subl[j]);
    for (k=0; k < subl[j]; k++) {
        fprintf(stdout,"%s ",alle[i]);
        i=i+1;
    }
    fprintf(stdout,"\n");
}

fprintf(stdout,"\nrestoring VA#2 \n");
hlin2="VA";
tc_restore_constituent(hlin1,2,hlin2);
hlin2="ION";

fprintf(stdout,"phase_constituents\n");
n=tc_phase_constituents(hlin2,(int *)subl,(const char *)alle,8,(double
*)nsite);
i=0;
fprintf(stdout,"%d\n",n);
for (j=0; j < n ; j++) {
    fprintf(stdout,"%d :",j+1);
    fprintf(stdout,"%f : ",nsite[j]);
    fprintf(stdout,"%d : ",subl[j]);
    for (k=0; k < subl[j]; k++) {
        fprintf(stdout,"%s ",alle[i]);
        i=i+1;
    }
    fprintf(stdout,"\n");
}
fprintf(stdout,"phase_all_constituents\n");
n=tc_phase_all_constituents(hlin2,(int *)subl,(const char
*)alle,8,(double *)nsite);
i=0;
fprintf(stdout,"%d\n",n);
for (j=0; j < n ; j++) {
    fprintf(stdout,"%d :",j+1);
    fprintf(stdout,"%f : ",nsite[j]);
    fprintf(stdout,"%d : ",subl[j]);
    for (k=0; k < subl[j]; k++) {
        fprintf(stdout,"%s ",alle[i]);
        i=i+1;
    }
    fprintf(stdout,"\n");
}
}
return 0;}

```

4.6 Example 6

```
/* *****  
/* Simple example showing how to create a window based application.      */  
/* The program calculates the solidus and liquidus temperatures for an    */  
/* Fe-Cr-Ni-C alloy.                                                    */  
/* *****  
#include <stdio.h>  
#include <string.h>  
#include <windows.h>  
#include "tcapi.h"  
  
#define APPLY 100  
  
#define EDIT_C 200  
#define EDIT_CR 201  
#define EDIT_NI 201  
  
#define STATIC_C 300  
#define STATIC_CR 301  
#define STATIC_NI 301  
  
#define LIQUIDUS 401  
#define SOLIDUS 402  
  
  
#define MAXOUT 25  
  
#define P3FILE "fecrnic.POLY3"  
  
LRESULT CALLBACK WndProc(HWND hWnd,UINT iMsg, WPARAM wParam, LPARAM lParam);  
  
HWND SetupWindow(char* cClass, char* cTitle, int nWidth, int nHeight, HINSTANCE  
hInstance);  
  
HWND hWndMe,  
    hAPPLY,  
    hEDIT_C, hEDIT_CR, hEDIT_NI,  
    hSTATIC_C, hSTATIC_CR, hSTATIC_NI,  
    hLIQUIDUS, hSOLIDUS;  
  
char cBuf[MAXOUT];  
  
short is_not_initialized=TRUE;  
  
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR  
lpCmdline, int iCmdShow)  
{  
    char* cClass = "SOL/LIQ";  
    char* cTitle = "SOL/LIQ";  
    const int nWidth = 300;  
    const int nHeight = 300;  
  
    MSG msg;  
  
    // Call routine to set up main window  
    hWndMe = SetupWindow(cClass, cTitle, nWidth, nHeight, hInstance);
```

```

// Set up buttons, input and output fields
hAPPLY=CreateWindow("button", "APPLY",
    WS_CHILD | WS_VISIBLE | ES_DEFPUSHBUTTON,
    220, 240, 70, 30,
    hWndMe, (HMENU)APPLY, hInstance, NULL);

hEDIT_C = CreateWindowEx(WS_EX_CLIENTEDGE, "edit", "",
    WS_CHILD | WS_VISIBLE | ES_RIGHT ,
    20, 20, 50, 25,
    hWndMe, (HMENU)EDIT_C, hInstance, NULL);

hSTATIC_C=CreateWindow("static", "w/p C",
    WS_CHILD | WS_VISIBLE,
    75, 20, 50, 25,
    hWndMe, (HMENU)STATIC_C, hInstance, NULL);

hEDIT_CR = CreateWindowEx(WS_EX_CLIENTEDGE, "edit", "",
    WS_CHILD | WS_VISIBLE | ES_RIGHT ,
    20, 50, 50, 25,
    hWndMe, (HMENU)EDIT_CR, hInstance, NULL);

hSTATIC_CR=CreateWindow("static", "w/p CR",
    WS_CHILD | WS_VISIBLE,
    75, 50, 50, 25,
    hWndMe, (HMENU)STATIC_CR, hInstance, NULL);

hEDIT_NI = CreateWindowEx(WS_EX_CLIENTEDGE, "edit", "",
    WS_CHILD | WS_VISIBLE | ES_RIGHT ,
    20, 80, 50, 25,
    hWndMe, (HMENU)EDIT_NI, hInstance, NULL);

hSTATIC_NI=CreateWindow("static", "w/p NI",
    WS_CHILD | WS_VISIBLE,
    75, 80, 50, 25,
    hWndMe, (HMENU)STATIC_NI, hInstance, NULL);

hLIQUIDUS=CreateWindow("static", "Liquidus:",
    WS_CHILD | WS_VISIBLE,
    20, 120, 200, 25,
    hWndMe, (HMENU)LIQUIDUS, hInstance, NULL);

hSOLIDUS=CreateWindow("static", "Solidus:",
    WS_CHILD | WS_VISIBLE,
    20, 160, 200, 25,
    hWndMe, (HMENU)SOLIDUS, hInstance, NULL);

// Display the window and enter main event loop
ShowWindow(hWndMe, iCmdShow);
UpdateWindow(hWndMe);

while(GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

```

```

        return msg.wParam;
    }

// Procedure to handle the calculation
void applyfunction()
{
    int istat,ierr;
    char errmsg[80];
    double cval,crval,nival,liqtemp,soltemp;
    short error;

// Retrieve information from input fields
GetWindowText(hEDIT_C, cBuf, MAXOUT);
cval = atof(cBuf);
GetWindowText(hEDIT_CR, cBuf, MAXOUT);
crval = atof(cBuf);
GetWindowText(hEDIT_NI, cBuf, MAXOUT);
nival = atof(cBuf);

// Initialize the Thermo-Calc system and retrieve thermodynamic
// information if necessary
if (is_not_initialized) {
    istat=tc_init_root();
    error=tc_error(&ierr,errmsg,80);
    if (error) {
        MessageBox(NULL, errmsg, "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    tc_read_poly3_file((const char *)P3FILE);
    error=tc_error(&ierr,errmsg,80);
    if (error) {
        MessageBox(NULL, errmsg, "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    is_not_initialized=FALSE;
}

// Calculate an initial equilibrium
tc_set_phase_status("LIQUID", "ENTERED",1.0);
tc_set_condition("T",2000.);
tc_set_condition("P",101325.);
tc_set_condition("N",1.);
tc_set_condition("W(C)",0.01*cval);
tc_set_condition("W(CR)",0.01*crval);
tc_set_condition("W(NI)",0.01*nival);
tc_compute_equilibrium();
error=tc_error(&ierr,errmsg,80);
if (error) {
    MessageBox(NULL, errmsg, "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return;
}

// Calculate the liquidus by setting the liquid to be fixed with
// a certain amount

```

```

tc_delete_condition("T");
tc_set_phase_status("LIQUID", "FIXED", 1.0);
tc_compute_equilibrium();
error=tc_error(&ierr,errmsg,80);
if (error) {
    MessageBox(NULL, errmsg, "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return;
}
liqtemp=tc_get_value("T");

// Calculate another initial equilibrium at a lower temperature
tc_set_phase_status("LIQUID", "ENTERED", 1.0);
tc_set_condition("T", liqtemp-100.);
tc_compute_equilibrium();
error=tc_error(&ierr,errmsg,80);
if (error) {
    MessageBox(NULL, errmsg, "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return;
}

// Calculate the solidus by setting the liquid to be fixed with
// a zero amount
tc_delete_condition("T");
tc_set_phase_status("LIQUID", "FIXED", 0.0);
tc_compute_equilibrium();
error=tc_error(&ierr,errmsg,80);
if (error) {
    MessageBox(NULL, errmsg, "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return;
}
soltemp=tc_get_value("T");

// Display the result
sprintf(cBuf, "Liquidus: %g C", liqtemp-273.15);
SetWindowText (hLIQUIDUS, cBuf);
sprintf(cBuf, "Solidus: %g C", soltemp-273.15);
SetWindowText (hSOLIDUS, cBuf);
}

// procedure to handle events
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static bool dot;
    static bool backsp;

    switch (iMsg)
    {
        case WM_DESTROY:
        {
            PostQuitMessage(0);
            return 0;
            break;
        }
        case WM_COMMAND:
        {
            switch(LOWORD(wParam))
            {
                case APPLY:
                {

```

```

        applyfunction();
        break;
    }
}
break;
}
}
return DefWindowProc(hWnd, iMsg, wParam, lParam);
}

// procedure to set up the initial window
HWND SetupWindow(char* cClass, char* cTitle, int nWidth, int nHeight, HINSTANCE
hInstance)
{
    WNDCLASS wClass;
    HWND hWnd;

    wClass.style = CS_HREDRAW | CS_VREDRAW;
    wClass.lpfnWndProc = WndProc;
    wClass.cbClsExtra = 0;
    wClass.cbWndExtra = 0;
    wClass.hInstance = hInstance;
    wClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wClass.hbrBackground = (HBRUSH)GetStockObject(LTGRAY_BRUSH);
    wClass.lpszMenuName = NULL;
    wClass.lpszClassName = cClass;

    RegisterClass(&wClass);

    hWnd = CreateWindow(cClass, cTitle, WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX, CW_USEDEFAULT, CW_USEDEFAULT, nWidth, nHeight, NULL, NULL,
hInstance, NULL);

    return hWnd;
}

```