

**The Geochemist's Workbench®**

Release 16

**GWB  
Reference  
Manual**



**The Geochemist's Workbench®**  
**Release 16**

---

---

**GWB**  
**Reference**  
**Manual**

---

---

**Craig M. Bethke**  
**Brian Farrell**

**Aqueous Solutions, LLC**  
**Champaign, Illinois**

**Printed March 25, 2022**

This document © Copyright 2022 by Aqueous Solutions LLC. All rights reserved. Earlier editions copyright 2000–2021. This document may be reproduced freely to support any licensed use of the GWB software package.

Software copyright notice: Programs GSS, Rxn, Act2, Tact, SpecE8, Gtplot, TEdit, React, Phase2, P2plot, X1t, X2t, Xtplot, and ChemPlugin © Copyright 1983–2022 by Aqueous Solutions LLC. An unpublished work distributed via trade secrecy license. All rights reserved under the copyright laws.

The Geochemist's Workbench®, ChemPlugin™, We put bugs in our software™, and The Geochemist's Spreadsheet™ are a registered trademark and trademarks of Aqueous Solutions LLC; Microsoft®, MS®, Windows 11®, and Windows 10® are registered trademarks of Microsoft Corporation; PostScript® is a registered trademark of Adobe Systems, Inc. Other products mentioned in this document are identified by the trademarks of their respective companies; the authors disclaim responsibility for specifying which marks are owned by which companies. The software uses zlib © 1995-2005 Jean-Loup Gailly and Mark Adler, and Expat © 1998-2006 Thai Open Source Center Ltd. and Clark Cooper.

The GWB software was originally developed by the students, staff, and faculty of the Hydrogeology Program in the Department of Geology at the University of Illinois Urbana-Champaign. The package is currently developed and maintained by Aqueous Solutions LLC at the University of Illinois Research Park.

Address inquiries to

Aqueous Solutions LLC  
301 North Neil Street, Suite 400  
Champaign, IL 61820 USA

**Warranty:** The Aqueous Solutions LLC warrants only that it has the right to convey license to the GWB software. Aqueous Solutions makes no other warranties, express or implied, with respect to the licensed software and/or associated written documentation. Aqueous Solutions disclaims any express or implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Aqueous Solutions does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the Licensed Software or documentation in terms of correctness, accuracy, reliability, currentness, or otherwise. Aqueous Solutions shall not be liable for any direct, indirect, consequential, or incidental damages (including damages for loss of profits, business interruption, loss of business information, and the like) arising out of any claim by Licensee or a third party regarding the use of or inability to use Licensed Software. The entire risk as to the results and performance of Licensed Software is assumed by the Licensee. See License Agreement for complete details.

**License Agreement:** Use of the GWB is subject to the terms of the accompanying License Agreement. Please refer to that Agreement for details.

Cover photo: Salinas de Janubio by Jorg Hackemann.

# Contents

---

---

## Chapter List

<b>1 Introduction</b> . . . . .	1
<b>2 Command Line Interface</b> . . . . .	3
<b>3 Thermo Datasets</b> . . . . .	9
<b>4 Surface Datasets</b> . . . . .	27
<b>5 Report Command</b> . . . . .	37
<b>6 Control Scripts</b> . . . . .	59
<b>7 Plug-in Feature</b> . . . . .	63
<b>8 Units Recognized</b> . . . . .	97
<b>9 Graphics Output</b> . . . . .	105
<b>10 Scatter Data</b> . . . . .	109
<b>11 Multiple Analyses</b> . . . . .	117
<b>12 Remote Control</b> . . . . .	121
<b>Index</b> . . . . .	127



# Contents

---

<b>1 Introduction</b>	<b>1</b>	<b>3.5 Virial coefficients</b>	<b>21</b>
1.1 Chapters in the manual	1	3.5.1 Temperature expansions	21
1.2 Legacy features	2	3.5.1.1 Legacy temperature expansion	22
<b>2 Command Line Interface</b>	<b>3</b>	3.5.1.2 New temperature expansion	22
2.1 Spelling completion	3	3.5.1.3 Temperature range of validity	22
2.2 History substitution	3	3.5.2 Data blocks for species pairs and triplets	22
2.3 Special characters	5	3.5.2.1 Legacy temperature expansion for Pitzer coefficients	22
2.4 Calculator	5	3.5.2.2 New temperature expansion for Pitzer coefficients	23
2.5 Startup files	6	3.5.2.3 New temperature expansion for SIT coefficients	23
2.6 Online documentation	6	3.6 Legacy dataset formats	24
2.7 System commands	6	3.6.1 Principal temperatures	24
2.8 Text size in the windows	7	3.6.2 Temperature expansions	24
2.9 Keyboard shortcuts	7	3.6.3 Initial lines	24
<b>3 Thermo Datasets</b>	<b>9</b>	3.6.4 Header variables	24
3.1 Dataset format	9	3.6.5 Arbitrary reaction definition	25
3.2 Temperature expansions	10	3.6.6 Redox couples	25
3.2.1 $T$ -table expansions	10	3.6.7 Free electron	25
3.2.2 Polynomial expansions	11	3.6.8 Formulae for aqueous species	25
3.2.3 Choice of expansion	12	3.6.9 Solid solutions	25
3.3 Header data	12	3.6.10 Fugacity coefficients	25
3.3.1 Initial lines	12	<b>4 Surface Datasets</b>	<b>27</b>
3.3.2 Principal temperatures	13	4.1 Sections in a surface dataset	27
3.3.3 Header variables	13	4.2 Temperature expansions	28
3.4 Species and reactions	14	4.3 Header data	28
3.4.1 Elements	15	4.3.1 Initial lines	28
3.4.2 Basis species	15	4.4 Species and reactions	29
3.4.3 Redox couples	16	4.4.1 Basis species	30
3.4.4 Aqueous species	17		
3.4.5 Free electron	18		
3.4.6 Minerals	19		
3.4.7 Solid solutions	19		
3.4.8 Gases	20		
3.4.9 Oxide components	20		

## Contents

---

4.4.2	Sorbing minerals . . . . .	30	7.4.2	Initializing the GWB application	91
4.4.3	Surface species . . . . .	31	7.4.3	Configuringandexecutingcal-	
4.5	Legacy dataset formats . . . . .	34	culations . . . . .	92	
4.5.1	Thermo data line . . . . .	34	7.4.4	Retrieving the results . . . . .	93
4.5.2	Three-layer models . . . . .	34	<b>8</b>	<b>Units Recognized</b>	<b>97</b>
4.5.3	Polydentate sorption . . . . .	34	<b>9</b>	<b>Graphics Output</b>	<b>105</b>
4.5.4	End line . . . . .	34	9.1	Clipboard . . . . .	105
4.5.5	Charged uncomplexed sites . . . . .	34	9.2	Saving images . . . . .	106
4.5.6	Site density units . . . . .	35	9.3	Font for data markers . . . . .	108
4.5.7	Arbitrary reaction definition . . . . .	35	<b>10</b>	<b>Scatter Data</b>	<b>109</b>
4.5.8	Temperature expansions . . . . .	35	10.1	Act2 and Tact . . . . .	110
<b>5</b>	<b>Report Command</b>	<b>37</b>	10.2	Gtplot . . . . .	111
<b>6</b>	<b>Control Scripts</b>	<b>59</b>	10.3	P2plot . . . . .	115
6.1	Control statements . . . . .	60	10.4	Xtplot . . . . .	115
6.2	Interacting with the application . . . . .	60	<b>11</b>	<b>Multiple Analyses</b>	<b>117</b>
6.3	Example control script . . . . .	61	11.1	Calculation procedure . . . . .	117
6.4	Tcl license agreement . . . . .	62	11.2	Example calculation . . . . .	118
<b>7</b>	<b>Plug-in Feature</b>	<b>63</b>	<b>12</b>	<b>Remote Control</b>	<b>121</b>
7.1	C++ . . . . .	64	12.1	C++ program, unnamed pipes . . . . .	122
7.1.1	Initializing the GWB application	64	12.2	C++ program, named pipes . . . . .	124
7.1.2	Configuringandexecutingcal-		12.3	Tcl script, unnamed pipes . . . . .	125
culations . . . . .	65		12.4	Perl script, unnamed pipes . . . . .	126
7.1.3	Retrieving the results . . . . .	66	<b>Index</b>		<b>127</b>
7.1.4	C++ programs . . . . .	67			
7.1.5	Compiling and linking . . . . .	70			
7.2	Fortran . . . . .	73			
7.2.1	Initializing the GWB application	74			
7.2.2	Configuringandexecutingcal-				
culations . . . . .	75				
7.2.3	Retrieving the results . . . . .	76			
7.2.4	Fortran programs . . . . .	77			
7.2.5	Compiling . . . . .	81			
7.3	Python . . . . .	84			
7.3.1	Initializing the GWB application	84			
7.3.2	Configuringandexecutingcal-				
culations . . . . .	86				
7.3.3	Retrieving the results . . . . .	86			
7.3.4	Python programs . . . . .	87			
7.3.5	Python command line . . . . .	89			
7.4	Other . . . . .	90			
7.4.1	GWBplugin.dll function pro-				
totypes . . . . .	90				



# Introduction

---

This **GWB Reference Manual** contains information about the command line interface on the **Command** pane, the format of thermodynamic datasets, the report command, control scripts, the plug-in feature for running the GWB applications from within your own applications, unit conversion within the programs, and manipulating graphics output.

The manual also describes several legacy features: text-format scatter data, using scripts to process multiple analyses, and the remote control feature. Each of these has been superseded by the **GSS** application or the plug-in feature.

This manual is intended as a supplement to the GWB manuals: **GWB Essentials Guide**, **GWB Reaction Modeling Guide**, **GWB Reactive Transport Modeling Guide**, and the **GWB Command Reference**.

Please consult the latter manual for specifics about the commands used to configure the GWB programs.

## 1.1 Chapters in the manual

This manual contains chapters that provide details about specific features of the GWB software package:

- **Command Line Interface** — The features of the user interface for the **Command** pane, including spelling completion, history substitution, and the built-in calculator.
- **Thermo Datasets** — Information about formatting and content of the thermodynamic databases that the GWB programs can read. This information is useful if you need to modify the database, or create your own.
- **Surface Datasets** — Information about formatting and content of surface datasets.
- **Report Command** — Format and use of the “report” command, which returns the results of calculations. This command provides a means of transmitting results to control scripts and to programs running a GWB application as a plug-in or by remote control.
- **Control Scripts** — How to set up within GWB input file scripts containing loops, branches, if checks, and so on.

- **Plug-in Feature** — Details how to use the capabilities of the GWB applications through the functions of a DLL.
- **Units Recognized** — A complete table of the unit names to be used in the commands.
- **Graphics Output** — How you can manipulate the graphical output from **Act2**, **Tact**, **Gtplot**, **P2plot**, and **Xtplot**.

## 1.2 Legacy features

The manual also describes several legacy features of the software:

- **Scatter Data** — The legacy method of adding scatter data to a diagram by importing a specially formatted table from a text file. The preferred method is to use a **GSS** spreadsheet as described in the **GWB Essentials Guide**.
- **Multiple Analyses** — Examples of how to process a number of chemical analyses from a spreadsheet and save the results to the spreadsheet.
- **Remote Control** — Details the deprecated legacy method of how you can run the GWB applications as slave processes from other programs and software environments. This method has been replaced by the **Plug-in Feature**.

# Command Line Interface

---

The command line interface for **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** includes a number of special features that you will find increasingly helpful as you gain experience.

## 2.1 Spelling completion

**Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** can complete the spelling of chemical and mineral names. The feature also completes the names of program commands and dataset names. To complete a spelling, begin typing the name and touch the tab (or, on most computers, escape) key. For example, if you type

```
1 free mole Musc[tab]
```

the program will complete the line

```
1 free mole Muscovite
```

When the program cannot identify a unique name from the letters provided, it will cycle through the possible completions with subsequent tab key presses. To list the possible completions, you can type **Ctrl+D**. For example, if you type

```
swap Al2^D
```

the program will respond with

```
Al2(OH)2++++ Al2(SO4)3 Al2(SO4)3:6H2O  
swap Al2|
```

leaving the cursor (“|”) in position to continue the command.

## 2.2 History substitution

**Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** maintain history lists. Previously executed commands are stored in the user’s profile directory (e.g., “c:\Documents

and Settings\ jones\Application Data\GWB”) in datasets such as “rxn\_history.dat”. If you type the command

```
history 10
```

the program lists the previous ten commands executed.

Each of the history substitution functions of the C-shell are available within the programs. For example, typing

```
!!  
!10  
!swap  
!?HCO3
```

causes the program to re-execute, respectively, the previous command, command number 10, the last command that began with “swap”, and the last command that contained the string “HCO3”.

Entries of the form

```
^string1^string2
```

replace the occurrence of “string1” with “string2” in the previous command, so that you can avoid retyping lengthy commands after simple errors. Finally, typing

```
!10-15
```

causes the program to re-execute commands 10 through 15. This latter feature is an extension to the C-shell protocol.

## 2.3 Special characters

The following special characters are used in the **Command** pane in **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t**:

Ctrl+D	Show choices for spelling completion of chemical names, commands, or file names.
Ctrl+F	Clear screen.
Ctrl+H	Backspace over a character.
Ctrl+N	Scroll forward through your command history to give the next command in list.
Ctrl+P	Scroll backward through your command history to give the previous command in list.
Ctrl+U	Backspace over entire line of input.
Ctrl+W	Backspace over previous word of input.
Tab or Esc	Cycle through choices for spelling completion of chemical names, commands, or file names.
\	Continue a command from one line to the next.

## 2.4 Calculator

Programs **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** provide an online calculator with these functions:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
( )	Grouping of terms
ln	Natural logarithm
log	Common logarithm
abs	Absolute value
sqrt	Square root
exp	Exponentiation of $e$
sin, cos, tan, cot, sec, csc	Trigonometric functions (arguments in radians)
e	Value of $e$
pi	Value of $\pi$

The online calculator is especially useful to a geochemist for converting numbers to their logarithms and vice versa, but can be used to evaluate any numerical expression. To use the calculator, type an expression at the prompt. Examples:

```
log .0003  
10^-4.5  
(200 - 32) * 5/9
```

In each case, the program evaluates the expression and returns its numerical value.

## 2.5 Startup files

Upon startup, **Rxn** looks for a file such as “rxn\_startup.rxn” in a user’s profile directory (found by typing %appdata% in the Windows Explorer Address bar, e.g., “c:\Documents and Settings\jones\Application Data\GWB”) and executes any commands in it; the other modeling programs similarly look for files named “act2\_startup.ac2”, “spece8\_startup.sp8”, and so on. These datasets provide a means for you to customize the working environment of each program. An “act2\_startup.ac2” file including the commands

```
background grey  
font Times
```

for example, will cause **Act2** to produce plots with grey backgrounds and Times lettering, unless told otherwise.

## 2.6 Online documentation

You can obtain online help for any of the programs on the **Docs** pane of the GWB dashboard, or using the “Help” pulldown menu on the menubar of any GWB program. The entire manual set, including this User’s Guide, can be accessed from the “Help” pulldown.

## 2.7 System commands

You can execute (“fork”) DOS commands from the command lines of **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t**. To do so, type a “\$” followed by the desired DOS command. Example:

```
$print React_output.txt
```

When a system command is executed, a “Command Prompt” window will appear briefly on your screen. Due to limitations of the Windows operating system, you cannot fork a command that requires user input, and you will not be able to see any output (including error messages) that might be generated by the command.

## 2.8 Text size in the GWB windows

You can control the font and size of the text within the program shells for **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** by choosing the desired font and point size from **View** → **Appearance...** "Reset" will change the point size back to the default value.

## 2.9 Keyboard shortcuts

Ctrl+Break	Break program (used during program execution, it stops the calculation and returns you to the command prompt)
Ctrl+A	Select all
Ctrl+C	Copy
Ctrl+Shift+C	Load conductivity data (SpecE8, React, Phase2, X1t, X2t)
Ctrl+F	Clear screen
Ctrl+G	Go, run calculation of the model
Ctrl+Shift+G	Load scatter data (Act2, Tact)
Ctrl+Shift+G	Go single, run on a single processor (Phase2, X1t, X2t)
Ctrl+I	Go initial, calculate the initial state of the medium (React, Phase2, X1t, X2t)
Ctrl+Shift+L	Launch Gtplot, P2plot, or Xtplot to show results (SpecE8, React, Phase2, X1t, X2t)
Ctrl+M	Save image... (Act2, Tact)
Ctrl+N	Add an entry to the basis
Ctrl+O	Read script (open data file)
Ctrl+Q	Quit the program
Ctrl+R	Reset configuration
Ctrl+Shift+R	Resume, restore the configuration from when the program was last exited
Ctrl+S	Save As...
Ctrl+Shift+S	Load sorbing surfaces (Rxn, SpecE8, React, Phase2, X1t, X2t)
Alt+S	Reset the size of the window to the default
Ctrl+T	Load thermo dataset
Ctrl+Shift+T	Load reaction trace (Act2, Tact)
Ctrl+U	Update trace (Act2, Tact)
Ctrl+V	Paste
Ctrl+Shift+W	Change working directory
Ctrl+X	Cut
Ctrl+Shift+X	Go X, run a single scanning path (Phase2)
Ctrl+Y	Redo (GSS)
Ctrl+Shift+Y	Go Y, calculate the staging path (Phase2)
Ctrl+Z	Undo (GSS)
F1	Open the User's Guide most relevant to the current app





# Thermo Datasets

---

The databases of thermodynamic data used by the programs are ascii (or character) datasets with a “.tdat” file extension. You can edit a “.tdat” file with **TEdit**, the thermo editor supplied as part of the GWB software. You can alternatively change a thermo file using a text editor, such as “Notepad” under MS Windows.

You are free to alter existing databases such as “thermo.tdat” by changing data or adding species, minerals, and so on. When changing a database, it is a good idea to copy the original database to a file with a new name, and then alter that file. You can also create your own databases using **TEdit**, or by following the dataset format in a text editor.

You access a thermo dataset from a GWB app in various ways: by dragging a “.tdat” file into the app’s window, opening the **File** → **Open** → **Thermo Data...** dialog, or using the “read” command from the **Command** pane or an input script. You can specify that specific dataset be read by default when a GWB app starts. To do so, set the dataset as the default thermodynamic file in **File** → **Preferences...** (see **Thermodynamic datasets** under **Configuring the Programs** in the **GWB Essentials Guide**).

The information in this chapter applies to the “mar21” format, which is used beginning with the GWB16 release. A description of legacy datasets appears at the end of the chapter.

## 3.1 Sections in a thermodynamic dataset

Each thermo dataset is composed of the following sections:

1. Initial lines, which identify the dataset
2. Header variables
3. Elements included in the dataset
4. Basis species
5. Redox couples
6. Aqueous species
7. Free electron

- 8. Minerals
- 9. Solid solutions
- 10. Gases
- 11. Oxide components
- 12. Virial coefficients, for datasets invoking a virial (“Pitzer” or SIT) activity model

Sections 3–10 begin with a header line such as

```
46 elements
```

which identifies the number of elements, species, and so on in each section. The count is ignored in the current software, but very old GWB releases require it to be accurate. A line

```
-end-
```

marks the end of each section.

You can include comment lines, identified by a “\*” as the first character, freely within the dataset. The programs read the data word-by-word, so it is not necessary to count spaces or align columns when adding or modifying data.

## 3.2 Temperature expansions

A primary purpose of a thermodynamic dataset is to define how certain values—the “header variables” and the log  $K$ s for the various reactions considered—vary as functions of temperature. In a dataset invoking the “Pitzer equations” or the SIT activity model, the virial coefficients can also vary with temperature; the formalism in this case is described in section **Virial coefficients**, later in this chapter.

For header variables and reaction log  $K$ s, a temperature expansion may be given as either a  $T$ -table or a set of polynomial coefficients. The two options are described below.

### 3.2.1 $T$ -table expansions

In the case of  $T$ -tables, the legacy method, an expansion is given as a set of values, one for each of the dataset’s principal temperatures:

```
* log Ks
-1.6590   -1.9641   -2.4627   -3.0613
-3.8086   500.0000   500.0000   500.0000
```

The corresponding principal temperatures are given in the header of the dataset, as described in the next section; commonly they are 0°C, 25°C, 60°C, 100°C, 150°C, 200°C, 250°C, and 300°C, though any number of principal temperatures can be chosen

and any values can be used. The first line above is a comment added here for clarity; it does not affect program execution.

A value of “500” in a  $T$ -table, as shown above, denotes a lack of data at the corresponding principal temperature (except pressure, for which “500” is interpreted as a real value). For log  $K$ s, the temperature range of validity for a  $T$ -table expansion is figured as the range of principal temperatures that map to non-“500” values; header variables are taken to span the principal temperatures, whether “500” entries are present or not.

In isothermal runs in which temperature is set to one of the principal temperatures, the GWB applications take the corresponding value from each data table. In a run at 25°C, for example, given the principal temperatures above, a GWB app takes the second entry in each  $T$ -table in the dataset.

Where temperature differs from a principal temperature, the application fits non-“500” values in each  $T$ -table to a polynomial

$$v = a_0 + a_1T + a_2T^2 + a_3T^3 + a_4T^4$$

with respect to temperature  $T$ , in °C. You can use **Rxn** to quickly check the polynomial fit of the log  $K$  for any chemical species:

```
react Anhydrite
long
go
```

for example. Finally, the command

```
span = on
```

forces a GWB app to use the polynomial expansion, even when it is working at one of the principal temperatures.

### 3.2.2 Polynomial expansions

Beginning with dataset format “jan19”, temperature expansions for header variables and log  $K$ s may alternatively be given as coefficients  $a$  through  $f$  in the polynomial

$$f(T_K) = a + b(T_K - T_r) + c(T_K^2 - T_r^2) + d\left(\frac{1}{T_K} - \frac{1}{T_r}\right) + e\left(\frac{1}{T_K^2} - \frac{1}{T_r^2}\right) + f \ln\left(\frac{T_K}{T_r}\right)$$

where  $T_K$  is absolute temperature, in Kelvin, and  $T_r$  is the reference temperature, 298.15 K.

The GWB apps distinguish data blocks of polynomial coefficients from  $T$ -tables by the presence of the tag “a=”. For example, a block defining a polynomial expansion might appear

```
* log10 K(298 K) = -1.9641
a= -1.964075      b= 0.24778      c= -0.00011177
d= -13261.67     e= 0              f= -102.477
TminK= 273.15    TmaxK= 373.15
```

where the first line is an optional comment. The coefficient assignments must span two lines and be set out in order, although the sequence may be truncated. Omitting coefficients  $e$  and  $f$  in the block above, for example, would cause them to be set to zero.

For log  $K$ s, an optional last line beginning with “TminK”, sets the expansion’s temperature range of validity. Unless the “extrapolate = on” option is set, a GWB app loads only species whose log  $K$  expansions are valid across the temperature range of the calculation. In the example above, the species in question would be loaded for a calculation performed at 298 K, or 25°C, but not one at 393 K, or 120°C. In the absence of this line, the range of validity is taken as the span of the dataset’s principal temperatures.

### 3.2.3 Choice of expansion

$T$ -table expansions, the legacy method, are broadly employed in GWB databases today. For this reason,  $T$ -tables are used to illustrate examples in the remainder of this chapter. To visualize a data block using polynomial expansions instead, simply replace the  $T$ -table values with two lines containing coefficients  $a$  through  $f$ , optionally along with a third line setting out a temperature range.

In developing new databases, we suggest taking advantage of the polynomial expansion feature, since the expansions can be imported directly from various repositories, do not require the GWB apps to re-fit polynomials internally, and because they allow temperature ranges of validity to be prescribed precisely.

## 3.3 Header data

### 3.3.1 Initial lines

A group of initial lines appears at the top of the dataset:

```
dataset of thermodynamic data for gwb programs
dataset format: mar21
activity model: b-dot
fugacity model: tsonopoulos
```

These lines identify the dataset, its format, and the activity and fugacity models to be invoked. The current format is “mar21”; earlier formats are described in the last section of this chapter.

A database may invoke a variant of the Debye-Hückel equation as its activity model, or be based on one formulated in terms of virial equations. Tags “b-dot” (equivalent to “debye-huckel”), “phreeqc”, “wateq4f”, “minteq”, and “davies” identify Debye-Hückel methods, whereas “h-m-w”, “phrpitz” (currently equivalent to “h-m-w”), and “sit” refer to virial formulations. The “pitzer” tag is outmoded and no longer supported.

The fugacity model is the default method for calculating gas partial pressure; it may be “tsonopoulos”, “peng-robinson”, or “spycher-reed”.

### 3.3.2 Principal temperatures

Following the initial lines, each “.tdat” file contains a table of principal temperatures

```
* temperatures
    0.00  25.00  60.00  100.00
   150.00 200.00 250.00 300.00
```

whether the dataset makes use of  $T$ -table temperature expansions, or not. By loose convention, the principal temperatures 0°C, 25°C, 60°C, 100°C, 150°C, 200°C, 250°C, and 300°C are commonly used, but the table can include as few as one entry

```
* temperatures
    25.00
```

or as many as are desired to span the dataset’s applicable range

```
* temperatures
   100.00 150.00 200.00 250.00
   300.00 350.00 400.00 450.00
   500.00 550.00 600.00
```

Values, in °C, should be set four per line until the maximum temperature is reached.

### 3.3.3 Header variables

Following the temperature table are temperature expansions giving pressure, coefficients for calculating parameters in activity coefficient correlations, and so on, either as  $T$ -tables or polynomials. The expansion for the first set of principal temperatures above looks like:

```
* pressures
    1.0134  1.0134  1.0134  1.0134
    4.7600 15.5490 39.7760 85.9270
* debye huckel a (adh)
    .4913  .5092  .5450  .5998
    .6898  .8099  .9785  1.2555
...

```

The expansion corresponding to the second looks like:

```
* pressures
    1.0134
* debye huckel a (adh)
    .5092
...
```

and the third:

```
* pressures
    500.0000  500.0000  500.0000  500.0000
    500.0000  500.0000  500.0000  500.0000
    500.0000  500.0000  500.0000
* debye huckel a (adh)
    .5646     .6346     .7190     .8215
    .9504     1.1182    1.3486    1.6882
    2.1947    2.8908    3.6724
...
```

The header variables are

- Pressure, in bar,
- The Debye-Hückel parameters  $A$  and  $B$ .

The various Debye-Hückel methods make use of both  $A$  and  $B$ , except for “davies”, which does not use  $B$ . The “sit” method, similarly, uses  $A$  alone. The “h-m-w” method uses  $A$  to determine the value of  $A^\phi$  as  $2.303A/3$  only when its temperature expansion is defined in polynomial form; the apps calculate  $A^\phi$  internally when  $A$  is set as a  $T$ -table using a method valid above the freezing point of water.

Datasets employing the “b-dot” activity model additionally contain temperature expansions for

- The Debye-Hückel extension  $\hat{B}$ ,
- Coefficients for calculating the activity coefficients for  $\text{CO}_2$  and some other electrically neutral species, and
- Coefficients for calculating the activity of water.

Temperature expansions for the header variables may be any combination of  $T$ -tables and polynomials.

### 3.4 Species and reactions

Next, a dataset holds entries defining a set of elements, several categories of aqueous species, minerals, solid solutions, gases, and oxides. Apart from the elements, basis

species, and solid solutions, entries include the reactions the species can undergo and the log  $K$ s for those reactions (note, the oxides contain reactions but no log  $K$ s). Reactions can be written in terms of any species in the dataset (basis species, redox couples, aqueous species, the free electron, minerals, and gases, but not elements, solid solutions, or fictive oxide components). Reactions are set with 3 species per line, until the reaction is complete.

Temperature expansions for the log  $K$ s must be either all  $T$ -tables or all polynomials. Whereas the two types of expansions may be mixed freely for header variables within a given dataset, the expansions for log  $K$ s must be of one type or the other.

Names can be no more than 31 characters, all of which should be drawn from the standard ASCII set. Names can not include a double colon, "::", as this will interfere with handling of solid solution end members. Names should not be reused (e.g., CO2(aq) and CO2(g) should be used rather than two entries named CO2).

### 3.4.1 Elements

The section begins with a list of elements from which species and so on in the database are composed. Each element has a name, chemical symbol, and mole weight

Oxygen	(O )	mole wt.=	15.9994 g
Silver	(Ag)	mole wt.=	107.8680 g
Aluminum	(Al)	mole wt.=	26.9815 g
Americium	(Am)	mole wt.=	241.0600 g
...			

### 3.4.2 Basis species

The following section sets out the basis species for the dataset, beginning with water (H<sub>2</sub>O). The entry for each species contains its charge, ion size parameter in ångstroms (for calculating its activity coefficient), mole weight (g/mol), and elemental composition

H2O			
charge=	0	ion size=	0.0 A mole wt.= 18.0152 g
2 elements in species			
1.000 O		2.000 H	
Ag+			
charge=	1	ion size=	2.5 A mole wt.= 107.8680 g
1 elements in species			
1.000 Ag			
Al+++			
charge=	3	ion size=	9.0 A mole wt.= 26.9815 g
1 elements in species			
1.000 Al			
Am+++			
charge=	3	ion size=	9.0 A mole wt.= 241.0600 g

```

1 elements in species
1.000 Am

```

The ion size parameter  $a_o$  in the dataset has special meaning for H<sub>2</sub>O in the various Debye-Hückel methods, where it controls the water activity calculation. When  $a_o = -3$ , water activity is set to one. When  $a_o = -2$ , the method of Wolery (1992b) derived for the B-dot model is used. When  $a_o = -1$ , the method of Garrels and Christ (1965) is used. Otherwise, the method of Helgeson (1969) is used in “b-dot” datasets and the Garrels and Christ method for “phreeqc”, “wateq4f”, “minteq”, and “davies” datasets.

For neutral species, similarly, the ion size parameter controls the activity coefficient calculation. When  $a_o \leq -1$ ,  $\hat{B}$  is treated as a Setschenow coefficient. The logarithm of the activity coefficient is set to the product  $\hat{B} \times I$ , where  $\hat{B}$  is given by the data tables above, and  $I$  is true ionic strength. When  $a_o = -0.5$ , the activity coefficient is calculated from the “CO2” coefficients in the data table section, according to equation 8.6 in the “Geochemical and Biogeochemical Reaction Modeling” text. Note, the methods yield a value of one for the activity coefficient in non-“b-dot” datasets, where the  $\hat{B}$  and “CO2” tables are not available. When  $a_o = -0.6$ , an alternate calculation for “CO2” from Drummond (1981) is used. Otherwise, the activity coefficient is set to one in “b-dot” datasets and calculated from the Setschenow equation (with a coefficient of 0.1), in “phreeqc”, “wateq4f”, “minteq”, and “davies” datasets.

Whenever a database is to include protonation and deprotonation reactions, the list of basis entries needs to include the hydrogen ion, H<sup>+</sup>, labeled as “H+”. Databases treating redox require as part of the list either dissolved dioxygen, O<sub>2</sub>(aq), or dissolved dihydrogen H<sub>2</sub>(aq); the species must be labeled, respectively, “O2(aq)” and “H2(aq)”. The species you choose, O<sub>2</sub>(aq) or H<sub>2</sub>(aq), is the database’s “redox pivot”.

### 3.4.3 Redox couples

Redox coupling reactions for the dataset are found in the following section. The Fe<sup>+++</sup>/Fe<sup>++</sup> couple, for example, is represented

```

Fe+++
charge= 3      ion size= 9.0 A   mole wt.= 55.8470 g
4 species in reaction
-0.500 H2O      1.000 Fe++      1.000 H+
0.250 O2(aq)
-10.0553 -8.4878 -6.6954 -5.0568
-3.4154 -2.0747 -0.8908 0.2679

```

The first two lines identify the redox species and give its charge, ion size parameter, and mole weight. The subsequent lines show the reaction by which the redox species dissociates. The final lines give log  $K$  values for this reaction at each of the principal temperatures.



The manner in which you represent electron acceptance and donation in the coupling reactions is flexible. For example, you might balance redox reactions in terms of aqueous or gaseous dioxygen, "O2(aq)" or "O2(g)":

```

Fe+++
charge= 3      ion size= 9.0 A   mole wt.= 55.8470 g
4 species in reaction
-0.500 H2O          1.000 Fe++          1.000 H+
0.250 O2(g)
-9.3901 -7.7630 -5.9309 -4.2756
-2.6496 -1.3462 -0.2258 0.8704

```

Or, you may set a half-cell reaction in terms of the free electron, "e-"

```

Fe+++
charge= 3      ion size= 9.0 A   mole wt.= 55.8470 g
2 species in reaction
1.000 Fe++          -1.000 e-
13.3713 13.0127 12.5821 12.1903
11.8236 11.5751 11.4559 11.5415

```

Similarly, you might use "H2(aq)" or "H2(g)".

```

Fe+++
charge= 3      ion size= 9.0 A   mole wt.= 55.8470 g
3 species in reaction
1.000 Fe++          1.000 H+          -0.500 H2(g)
13.6804 13.0127 12.2194 11.4873
10.7750 10.2446 9.8895 9.7741

```

In datasets that do not include a redox pivot within the basis, you would normally not include redox coupling reactions. In that case, the section would look like

```

0 redox couples
-end-

```

### 3.4.4 Aqueous species

The next section contains the aqueous species to be considered in addition to the basis and redox species. The entry for CaCl<sup>+</sup>, for example, is

```

CaCl+
charge= 1      ion size= 4.0 A   mole wt.= 75.5330 g
2 species in reaction
1.000 Ca++          1.000 Cl-

```

-0.9687	-0.7000	-0.5157	-0.4688
-0.5789	-0.8602	-1.3560	-2.2451

The entry contains a dissociation reaction for the species, and the log *K* values for this reaction.

To maintain the software's ability to couple and decouple redox reactions, you should balance reactions in this and following sections by avoiding the use, wherever possible, of O<sub>2</sub>(aq), O<sub>2</sub>(g), H<sub>2</sub>(aq), H<sub>2</sub>(g), and e<sup>-</sup>. To do so, you balance the reactions in terms of species of the same oxidation state as the species in question. The entry for H<sub>2</sub>S(aq), for example,

H2S(aq)			
charge=	0	ion size=	4.0 A
mole wt.=	34.0758 g		
2 species in reaction			
1.000 H+		1.000 HS-	
-7.6500	-6.9500	-6.6800	-6.6100
-6.7900	-7.1700	-7.7200	-8.4300

is properly balanced in terms of the redox species HS<sup>-</sup>, or the secondary aqueous species S<sup>2-</sup> perhaps, rather than the basis species SO<sub>4</sub><sup>2-</sup>.

Note that for any basis species, redox couple, or aqueous species, you may specify a stoichiometric formula

Acetic acid		formula= HCH3COO	
charge=	0	ion size=	4.0 A
mole wt.=	60.0524 g		
2 species in reaction			
1.000 H+		1.000 CH3COO-	
-4.7743	-4.7563	-4.8079	-4.9640
-5.3017	-5.8241	500.0000	500.0000

by appending a "formula=" field to the species' name.

### 3.4.5 Free electron

The half-cell reaction representing take-up of the free electron is typically in terms of either "O2(aq)" or "O2(g)":

e-			
charge=	-1	ion size=	0.0 A
mole wt.=	0.0000 g		
3 species in reaction			
0.500 H2O		-0.250 O2(g)	-1.000 H+
22.7614	20.7757	18.5130	16.4658
14.4732	12.9213	11.6817	10.6711

or in terms of "H2(aq)" or "H2(g)":

```
e-
charge= -1      ion size= 0.0 A      mole wt.= 0.0000 g
2 species in reaction
0.500 H2(g)      -1.000 H+
-0.3091  0.0000  0.3627  0.7030
1.0486  1.3305  1.5663  1.7673
```

### 3.4.6 Minerals

Minerals to be included in the database are found in the next section. The entry for the sodium feldspar albite looks like

```
Albite                      type= feldspar
formula= NaAlSi3O8
mole vol.= 100.250 cc      mole wt.= 262.2230 g
5 species in reaction
2.000 H2O                  1.000 Na+                1.000 Al+++
3.000 SiO2(aq)            -4.000 H+
3.9160  3.0973  1.9915  0.9454
-0.0499 -0.8183 -1.5319 -2.5197
```

The initial lines in the entry give the mineral's name, type, formula, and molar volume ( $\text{cm}^3/\text{mol}$ ) and weight ( $\text{g}/\text{mol}$ ). The remaining lines give the reaction by which the mineral decomposes and the corresponding  $\log K$  values at the principal temperatures. As with the aqueous species, the reaction should be written without change in oxidation state, if possible, but can otherwise be written in terms of any species in the dataset.

### 3.4.7 Solid solutions

The next section contains zero or more binary solid solutions. An example is,

```
High-Mg Calcite            type= guggenheim
Calcite                    tag= Cc          contains
Magnesite                  tag= Magn       from 0.0 to 0.4
a0 = -3.2                  a1 = 0.0
```

The first line in the block sets the solution's name and assigns an activity model, either "ideal" or "guggenheim". The second and third lines specify the end-member minerals, taken from the previous section.

Each end member is identified in the calculations by a short label or "tag". The name of a species, mineral, or gas in the thermo database cannot be used to name any solid solution, nor to tag an end member. The keyword "contains" optionally appears after the first mineral's tag, for clarity only.

You can on the next line specify a range for the second mineral's mole fraction in the solid solution; omitting the range falls through to the default, from 0 to 1. A nonideal solution requires a fourth line supplying the dimensionless Guggenheim parameters

$a_0$  and  $a_1$ . The sign of the second coefficient reflects the order in which end members appear in the block, as well as the GWB's convention for setting out the Guggenheim equations. For details, see **Solid solutions** under **Configuring the Programs** in the **GWB Essentials Guide**.

### 3.4.8 Gases

The next section contains the gases considered. The entry for CO<sub>2</sub>(g), for example, is

```
CO2(g)
mole wt.= 44.0098 g
chi= -1430.87 3.598 -.00227376 3.47644 -.0104247 8.46271e-6
Pcrit= 73.8 bar Tcrit= 304.1 K omega= .239
3 species in reaction
-1.000 H2O 1.000 H+ 1.000 HCO3-
-7.6827 -7.8184 -8.0628 -8.3849
-8.8297 -9.3208 -9.8841 -10.6132
```

The first lines give the name and mole weight (g/mol) of the gas.

The next two lines, which are optional and can appear in either order, hold data for calculating fugacity coefficients. A line starting with "chi=" gives the factors  $a$  through  $f$  used in the Spycher-Reed method to calculate the fugacity coefficient  $\varphi$

$$\ln \varphi = \left( \frac{a}{T_K^2} + \frac{b}{T_K} + c \right) P + \left( \frac{d}{T_K^2} + \frac{e}{T_K} + f \right) \frac{P^2}{2} \quad (3.1)$$

as a function of absolute temperature  $T_K$ , and pressure  $P$ , in bars.

A line beginning "Pcrit=" gives the data needed to evaluate the Tsonopoulos and Peng-Robinson pressure models: the critical point pressure  $P_{cr}$  and temperature  $T_{cr}$ , and the acentric factor  $\omega$ . For polar or hydrogen bonding gases, like H<sub>2</sub>O(g), the line may be extended to include factors  $a$  and  $b$

```
Pcrit= 221.2 bar Tcrit= 647.3 K omega= .344 a= -.0109 b= 0.0
```

used for such gases by the Tsonopoulos model.

As before, the remaining lines give the gas' dissociation reaction and corresponding log  $K$  values.

### 3.4.9 Oxide components

The oxide components are used as reactants in simulations that, for example, model the dissolution of a glass phase. An example is

```
Al2O3
mole wt.= 101.9616 g
3 species in reaction
-6.000 H+ 2.000 Al+++ 3.000 H2O
```

Since components are fictive entities used to describe bulk composition, oxides have no thermodynamic stability and hence there is no expansion for  $\log K$ .

### 3.5 Virial coefficients

The “Pitzer” and SIT activity models commonly used in geochemistry to describe chemical potentials in concentrated solutions rely on virial coefficients. Virial coefficients account for energetic interactions among aqueous species taken two and three at a time—i.e., species pairs and triplets.

For this reason, datasets calling on the “h-m-w”, “phrqpitz”, and “sit” activity models contain a final section defining virial coefficients for the species carried in the dataset; this last section is omitted from datasets employing the other activity models.

For the “h-m-w” (and “phrqpitz”) activity model, the section is made up of four segments of virial coefficients:

- Values of  $\beta^{(0)}$ ,  $\beta^{(1)}$ ,  $\beta^{(2)}$ ,  $c^\phi$ ,  $\alpha^{(1)}$ , and  $\alpha^{(2)}$  for cation-anion pairs,
- Values of  $\theta$  for anion-anion pairs,
- Values of  $\lambda$  for ion-neutral species pairs, and
- Values of  $\psi$  for species triplets.

In datasets calling the “sit” model, the section is made up of two segments:

- Values of  $\varepsilon$  for cation-anion pairs, and
- Values of  $\varepsilon$  for ion-neutral species pairs.

$\varepsilon$  is normally invariant with respect to ionic strength, but you can account for its variation by specifying two values  $\varepsilon_1$  and  $\varepsilon_2$ , in which case  $\varepsilon = \varepsilon_1 + \varepsilon_2 \times \log I$ .

The data format for a pair and triplet depends on the type of temperature expansion chosen for the entry.

#### 3.5.1 Temperature expansions

Virial coefficients  $\beta^{(0)}$ ,  $\beta^{(1)}$ ,  $\beta^{(2)}$ ,  $c^\phi$ ,  $\theta$ ,  $\lambda$ , and  $\psi$ , as well as  $\varepsilon$ ,  $\varepsilon_1$ , and  $\varepsilon_2$ , vary with temperature and hence carry temperature expansions in a dataset. Coefficients  $\alpha^{(1)}$  and  $\alpha^{(2)}$  are invariant with temperature.

The GWB apps recognize temperature expansions for virial coefficients in two similar forms: a legacy polynomial, and the polynomial used to expand header variables and  $\log K$ s, as described already. The two types of temperature expansions may be mixed freely within a given dataset, but only a single type may be used within the data block for a given pair or triplet of species.

### 3.5.1.1 Legacy temperature expansion

The legacy temperature expansion casts a virial coefficient's variation according to the polynomial

$$\beta = \beta_{25} + c_1 (T_K - T_r) + c_2 \left( \frac{1}{T_K} - \frac{1}{T_r} \right) + c_3 \ln \left( \frac{T_K}{T_r} \right) + c_4 (T_K^2 - T_r^2) + c_5 \left( \frac{1}{T_K^2} - \frac{1}{T_r^2} \right)$$

Here,  $\beta$  represents the virial coefficient in question,  $\beta_{25}$  is its value at 25°C,  $T_K$  is absolute temperature, and the reference temperature  $T_r$  is 298.15 K.

Each coefficient is defined on a line containing at a minimum the value at 25°C and optionally one or more of the polynomial coefficients  $c_1 - c_5$ . Any omitted entries for the polynomial are treated as zero values. Hence, a line containing  $\beta_{25}$  and  $c_1$  defines the virial coefficient in terms of its 25°C value and its first temperature derivative.

### 3.5.1.2 New temperature expansion

The temperature expansion used for header variables and log  $K$ 's beginning with dataset format "jan19" may also be used to describe the behavior of virial coefficients. In this case, an arbitrary coefficient  $\beta$  is given by coefficients  $a$  through  $f$  according to

$$\beta = a + b (T_K - T_r) + c (T_K^2 - T_r^2) + d \left( \frac{1}{T_K} - \frac{1}{T_r} \right) + e \left( \frac{1}{T_K^2} - \frac{1}{T_r^2} \right) + f \ln \left( \frac{T_K}{T_r} \right)$$

where  $T_K$  is Kelvin temperature, and the reference temperature  $T_r$  is 298.15 K. As you can see, the new and legacy expansions differ only in the order in which the terms are presented and the labels assigned to the coefficients.

The coefficients  $a$  through  $f$  are presented in order. The series may be truncated, in which case any remaining coefficients are carried as zero. Coefficient  $a$  is required, and  $a$  and  $b$  must be given at a minimum to set variation with temperature.

### 3.5.1.3 Temperature range of validity

A temperature range of validity may optionally be set for each species pair and triplet, regardless of the temperature expansion chosen. If no range is set, the span of the principal temperatures is assumed.

A GWB app loads only virial coefficients for pair or triplets whose range of validity spans the calculation's temperature range, unless the "extrapolate = on" command is given. If a pair's or triplet's coefficients are not loaded, the app gives a warning message and continues.

## 3.5.2 Data blocks for species pairs and triplets

A GWB app distinguishes virial coefficients expanded in terms of the new, rather than legacy polynomial by checking for the tag "a=" on a data line.

### 3.5.2.1 Legacy temperature expansion for Pitzer coefficients

An example block for the cation-anion pair  $\text{Na}^+\text{-Cl}^-$  in which the virial coefficients are expanded in terms of the legacy polynomial is

Na+	Cl-					
beta0	= .07558	-.06352	9931	37.47	2e-5	-508663
beta1	= .2765	-.1714	27034	102.3	6e-5	-1335514
beta2	= 0.0					
cphi	= .0015267	.03114	-4635	-18.1	-1e-5	221646.
alpha1	= 2.0					
alpha2	= 0.0					
TminK=	273.15	TmaxK=	480.15			

In this case, the value of  $\beta^{(0)}$  at 25°C is 0.07558, and the value's temperature variation is defined by polynomial coefficients  $c_1$ – $c_5$  of  $-.06352$ ,  $9931$ ,  $37.47$ ,  $2 \times 10^{-5}$ , and  $-508663$ .

A block for the  $\text{Na}^+$ - $\text{K}^+$ - $\text{Cl}^-$  triplet in which  $\psi$  is expanded in terms of the legacy polynomial is

Na+	K+	Cl-	
psi =	-.001800	2.047e-5	
TminK=	273.15	TmaxK=	393.15

### 3.5.2.2 New temperature expansion for Pitzer coefficients

An example of the block for the  $\text{Na}^+$ - $\text{Cl}^-$  pair, this time expanded in terms of the new polynomial form is

Na+	Cl-					
beta0	a= .07558	b= -.06352	c= 2e-5	d= 9931	e= -508663	f= 37.47
beta1	a= .2765	b= -.1714	c= 6e-5	d= 27034	e= -1335514	f= 102.3
beta2	a= 0.0					
cphi	a= .0015267	b= .03114	c= -1.5	d= -4635	e= 221646	f= -18.1
alpha1=	2.0					
alpha2=	0.0					
TminK=	273.15	TmaxK=	480.15			

The coefficients  $a$ – $f$  defining  $\beta^{(0)}$  in the example are  $.07558$ ,  $-.0635$ ,  $2 \times 10^{-5}$ ,  $9931$ ,  $-508663$ , and  $37.47$ .

Similarly, a block for the  $\text{Na}^+$ - $\text{K}^+$ - $\text{Cl}^-$  triplet is

Na+	K+	Cl-	
psi	a= -.001800	b= 2.047e-5	
TminK=	273.15	TmaxK=	393.15

### 3.5.2.3 New temperature expansion for SIT coefficients

An example of the block for the  $\text{Na}^+$ - $\text{Cl}^-$  pair in the SIT model, expanded in terms of the new polynomial form, is

Na+	Cl-						
epsilon	a= .03	b= -.052	c= 1e-5	d= 8734	e= -453573	f= 22.13	
TminK=	273.15	TmaxK=	353.15				

To account for variation of  $\varepsilon$  with ionic strength, include a second coefficient  $\varepsilon_2$ :

Na+	NO3-		
epsilon1	a= -.049		
epsilon2	a= .044		
TminK=	273.15	TmaxK=	323.15

### 3.6 Legacy dataset formats

Six legacy formats for the thermo databases may be used with the current release of the software. The legacy formats are labeled “apr20”, “jan19”, “jul17”, “oct13”, “oct94” and “feb94”.

The first five formats are preferred for “h-m-w” and “phrqpitz” activity models, and the latter accepts the outmoded and deprecated “pitzer” feature, which should not be used. GWB Release 12 works only with “jul17” and earlier formats. The differences between the legacy and current formats are summarized below.

#### 3.6.1 Principal temperatures

Dataset formats “apr20” and earlier must contain exactly eight principal temperatures.

#### 3.6.2 Temperature expansions

Dataset formats “jul17” and earlier support only  $T$ -table temperature expansions for header variables and log  $K$ s, and the legacy polynomial for virial coefficients. Datasets in these formats additionally do not recognize temperature ranges of validity for virial coefficients.

#### 3.6.3 Initial lines

Datasets invoking the “B-dot” form of the extended Debye-Hückel equation in “apr20” and earlier formats use the activity model label “debye-huckel” instead of “b-dot”. Datasets in formats “jan19” and earlier cannot invoke the “sit” activity model. Datasets in formats “oct13” and earlier do not support partial pressure calculations and therefore do not contain an initial line specifying the fugacity model.

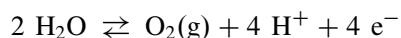
#### 3.6.4 Header variables

Thermo datasets in “jul17” and earlier formats include sets of header variables for calculating activities of CO<sub>2</sub> and some other neutral species, and for H<sub>2</sub>O, regardless of the dataset’s activity model.

Datasets in the “oct94” and “feb94” formats include data blocks for four header variables



- The log  $K$  values for the half-cell reaction



and

- The log  $K$  values for the solubilities of the gases  $\text{O}_2(\text{g})$ ,  $\text{H}_2(\text{g})$ , and  $\text{N}_2(\text{g})$  although only the values for  $\text{O}_2(\text{g})$  were ever used by the software

These four header variables are not carried forward to later formats

### 3.6.5 Arbitrary reaction definition

Datasets in “jan19” and earlier formats required reactions to be written in terms of only basis species, redox couples, the aqueous or gaseous form of the redox pivot, or the free electron.

### 3.6.6 Redox couples

Redox coupling reactions in “oct94” and “feb94” format datasets can be balanced only in terms of “O2(aq)” as the electron acceptor.

### 3.6.7 Free electron

Databases in the “oct94” and “feb94” format do not include a reaction representing take-up of the free electron. Instead, the software constructs the reaction using the header variable for the half-cell reaction mentioned above.

### 3.6.8 Formulae for aqueous species

Databases in “oct13” and earlier formats do not include stoichiometric formulae for the aqueous species.

### 3.6.9 Solid solutions

Dataset formats “apr20” and earlier do not include a section for defining solid solutions.

### 3.6.10 Fugacity coefficients

Beginning with the “jul17” format, gas species blocks can contain an optional line giving the factors needed to calculate fugacity coefficients by the Spycher-Reed equation, as well as a line holding values used by the Tsonopoulos and Peng-Robinson models.



# Surface Datasets

---

Databases of surface reactions, which have a “.sdat” file extension, are similar to the thermodynamic datasets described in the previous chapter. You access a surface dataset from a GWB app by dragging a “.sdat” file into the app’s window, opening the **File** → **Open** → **Sorbing Surfaces...** dialog, or using the “surface\_data” command from the **Command** pane or an input script. You can load as many unique surface datasets into an app as you’d like.

The information in this chapter applies to the “may20” format, which is used beginning with the GWB15 release. A description of legacy datasets appears at the end of the chapter.

## 4.1 Sections in a surface dataset

Each surface dataset is composed of some or all of the following sections:

1. Initial lines, which identify the dataset
2. Basis species
3. Sorbing minerals
4. Surface species

Sections 2–4 begin with a header line such as

```
2 basis species
```

which identifies the number of surface basis species, sorbing minerals, and surface species in each section. Depending on the surface model, one or more such sections are omitted, as described below. A line

```
-end-
```

marks the end of each section.

As with thermo datasets, you can include comment lines freely within the dataset.

## 4.2 Temperature expansions

Beginning with dataset format “may20”, you can define how log  $K$ s for the reactions considered in two-layer, triple-layer, CD-MUSIC, and Langmuir datasets vary as functions of temperature using the polynomial expansion previously described for log  $K$ s in the thermo dataset

$$f(T_K) = a + b(T_K - T_r) + c(T_K^2 - T_r^2) + d\left(\frac{1}{T_K} - \frac{1}{T_r}\right) + e\left(\frac{1}{T_K^2} - \frac{1}{T_r^2}\right) + f \ln\left(\frac{T_K}{T_r}\right)$$

where  $T_K$  is absolute temperature, in Kelvin, and  $T_r$  is the reference temperature, 298.15 K.

For example, a block defining a polynomial expansion might appear

```
log K a= -1.964075 b= 0.24778 c= -0.00011177 d= -13261.67 e= 0 f= -102.477
TminK= 273.15 TmaxK= 323.15
```

The coefficients must be set in order, although the sequence may be truncated after the second coefficient. Omitting coefficients  $e$  and  $f$  in the block above, for example, would cause them to be set to zero. Unlike in thermo datasets, the coefficient assignments must span a single line. In practice, data is seldom available to constrain more than the first coefficient. There is no provision for describing temperature variation of coefficients in the ion-exchange,  $K_d$ , and Freundlich models.

## 4.3 Header data

### 4.3.1 Initial lines

A group of initial lines appears at the top of the dataset:

```
Dataset of surface reactions for gwb programs
Dataset format: may20
Surface type: HFO
Model type: two-layer
Surface potential: variable (specify a value in mV to set a constant potential model)
Surface capacitance: variable (specify a value in F/m2 to set a constant capacitance model)
Thermo dataset: thermo.tdat
```

The first three lines identify the dataset, its format, and its label, to be used for input and output. A unique label is required for each surface in a calculation.

The fourth line specifies the surface model to invoke. The model type may be “two-layer”, “three-layer”, “cd-music”, “Langmuir”, “ion-exchange”, “Kd”, or “Freundlich”.

For the first four model types listed, you may optionally append a convention for polydentate complexes: “stoichiometric”, “davis-leckie”, “hiemstra-vanriemsdijk” (the default), or “appelo-postma” (see **Polydentate sorption** in the **GWB Essentials Guide**)

Model type: cd-music hiemstra-vanriemsdijk

For ion-exchange datasets, similarly, specify the activity convention for the sorbed sites: “Gaines-Thomas” (the default), “Vanselow”, or “Gapon”. You must balance reactions accordingly (see **Ion exchange** in the **GWB Essentials Guide**)

Model type: ion-exchange Gapon

In the two-layer model, the Surface potential and Surface capacitance lines are set to “variable” to make the programs use the full diffuse-layer model. You can alternatively set the former to a numerical value (in mV) to implement the constant potential model (see **Constant potential model** in the **GWB Essentials Guide**)

Surface potential: 0  
Surface capacitance: variable

Surface capacitance can similarly be set (in  $F/m^2$ ) to implement the constant capacitance model (see **Constant capacitance model** in the **GWB Essentials Guide**)

Surface potential: variable  
Surface capacitance: 0.9

In the triple-layer and CD-MUSIC models, set two capacitances  $C_1$  and  $C_2$

Surface potential: n/a  
Surface capacitance: 1.1 .2

For all other surface models, use “n/a”

Surface potential: n/a  
Surface capacitance: n/a

The last header line identifies in **TEdit** the thermo dataset from which to draw species to include in the surface dataset. When a surface dataset is loaded in a calculation, either the specified thermo dataset or one that is compatible (i.e., one that includes all the necessary species, uses the same spellings, and so on) should also be loaded.

## 4.4 Species and reactions

Next, a dataset holds entries defining a set of surface species, the reactions they undergo, log  $K$ s or other coefficients describing their stabilities, and perhaps information about the minerals containing the sorbing surfaces.

#### 4.4.1 Basis species

The entry for a basis species, where appropriate, includes its charge, mole weight (g/mol), and elemental composition. Descriptions for the various surface models follow.

In the surface complexation models (two-layer, triple-layer, and CD-MUSIC models), you specify one or more basis species representing the sorbing sites. The species represents the site in its uncomplexed state and may be uncharged, as is common in two-layer and triple-layer models, or charged, as is common in the CD-MUSIC model.

```
>(s)FeOH
  charge= 0      mole wt.= 72.8543 g
  3 elements in species
    1.000 Fe      1.000 H      1.000 O

>(w)FeOH
  charge= 0      mole wt.= 72.8543 g
  3 elements in species
    1.000 Fe      1.000 H      1.000 O
```

In the Langmuir isotherm, specify a single sorbing site per dataset. Most commonly, the site is uncharged, carries no mass, and is not composed of any elements.

```
>L:
  charge= 0      mole wt.= 0.0000 g
  0 elements in species
```

In the ion-exchange model, set a single basis species, which is the surface complex for one of the exchanging ions. It is customary to choose a monovalent ion for the basis species. The site should be uncharged, and the “sorbed species” must be a basis entry in the thermo dataset in use.

```
>X:Na
  charge= 0      mole wt.= 22.9898 g
  sorbed species= Na+
  1 elements in species
    1.000 Na
```

$K_d$  and Freundlich model datasets do not contain a Basis species because there is no mass balance on sorbing sites.

#### 4.4.2 Sorbing minerals

In the surface complexation datasets, identify one or more sorbing minerals. For each, specify its specific surface area (m<sup>2</sup>/g) and the density of each sorbing site. Site density can be specified in moles sites per mole mineral, as in FeOH.sdat

```
Fe(OH)3(ppd)
  surface area= 600.0000 m2/g
```

```

2 sorption sites
>(s)FeOH   site density=   .0050 mol/mol mineral
>(w)FeOH   site density=   .2000 mol/mol mineral

```

or sites per square nanometer, as in Goethite\_Se.sdat

```

Goethite
  surface area= 52.0000 m2/g
  1 sorption sites
  >FeOH   site density=   7.0000 sites/nm2

```

Datasets for the other surface models do not include a Sorbing minerals section. Note, the sorption capacity and exchange capacity of Langmuir and ion-exchange sites, respectively, are set in the apps.

#### 4.4.3 Surface species

The entry for a surface species might include a description of charge, or change in charge, on one or more planes, as well as the mole weight (g/mol) for the species. It additionally includes the reaction by which the species dissociates to basis, redox, or aqueous species in the specified thermo dataset, as well as the sorbing site, if applicable. Finally, the entry includes the stability for the specified reaction. Descriptions for the various surface models follow.

In the two-layer surface complexation model, set a single value for charge of the surface complex and set the log  $K$  temperature expansion described above:

```

>(w)FeO-
  charge=  -1      mole wt.=  71.8464 g
  2 species in reaction
    1.000 >(w)FeOH      -1.000 H+
  log K   a=  8.9300   b=  0.0

>(w)FeOH2+
  charge=   1      mole wt.=  73.8622 g
  2 species in reaction
    1.000 >(w)FeOH      1.000 H+
  log K   a= -7.2900   b=  0.0

>(s)FeOPb+
  charge=   1      mole wt.=  279.0464 g
  3 species in reaction
    1.000 >(s)FeOH      -1.000 H+          1.000 Pb++
  log K   a= -4.6500   b=  0.0

```

The triple-layer model is similar to the two-layer model, except the charge of a complex is attributed either to the  $\alpha$ -plane,  $\beta$ -plane, or both. For example

```

>FeOH2+
  charge0= 1   chargeb= 0   mole wt.= 73.8622 g
  2 species in reaction
    1.000 >FeOH      1.000 H+
  log K   a= -5.8000   b= 0.0

>FeO-:Na+
  charge0= -1  chargeb= 1   mole wt.= 94.8362 g
  3 species in reaction
    1.000 >FeOH      1.000 Na+      -1.000 H+
  log K   a= 8.8000   b= 0.0
  
```

The CD-MUSIC model is similar to the triple-layer model in that charge is attributed to multiple planes. In this case, specify the changes in charge  $\Delta z_0$ ,  $\Delta z_1$ , and  $\Delta z_2$  for each of the planes 0,  $\beta$ , and  $d$ , respectively:

```

>FeOH2+0.5
  charge= .5   mole wt.= 73.8622 g
  deltaz0= -1  deltaz1= 0   deltaz2= 0
  2 species in reaction
    1.000 >FeOH-0.5   1.000 H+
  log K   a= -9.2000   b= 0.0

>FeOHNa+0.5
  charge= .5   mole wt.= 95.8441 g
  deltaz0= 0   deltaz1= 0   deltaz2= -1
  2 species in reaction
    1.000 >FeOH-0.5   1.000 Na+
  log K   a= 1.0000   b= 0.0
  
```

Recall that since GWB datasets describe dissociation, a reaction's  $\log K$  as well as its  $\Delta z$  values are negated relative to the corresponding association reaction. Note, in modern usage,  $\Delta z_2$  is commonly taken to be 0.

In the Langmuir model, the surface species are complexes of the basis species with ions in solution. Enter equilibrium constants for dissociation reactions in the same manner as the two-layer, triple-layer, and CD-MUSIC surface complexes.

```

>L:Ca++
  charge= 2   mole wt.= 40.0800 g
  2 species in reaction
    1.000 >L:      1.000 Ca++
  log K   a= -2.1000   b= 0.0
  
```



In the ion-exchange model, the surface species are formed by ion exchange with the basis species. Enter selectivity coefficients as linear, not logarithmic values (they are converted to log values internally in the program).

The selectivity coefficients here refer to mass action expressions for the exchange reaction written in terms of the ion activities, rather than molalities. To be precise, therefore, data from the literature presented in the molal convention should be corrected according to the ions' activity coefficients.

Be sure your selectivity coefficients correspond to the reaction specified. Selectivity coefficients reported in the literature may be normalized to a unit charge equivalent of exchange sites.

The Gaines-Thomas convention holds that the activity of the sorbed ion is its equivalent fraction of the sorbing sites. You balance reactions on the exchanging ions:

```
>X2:Ca
charge= 0      mole wt.= 40.0800 g
3 species in reaction
  2.000 >X:Na      1.000 Ca++      -2.000 Na+
Selectivity coefficient= .0300
```

The Vanselow convention is the same, except that the sorbed ion activity is the molar fraction of the sorbing sites.

In the Gapon convention, you balance reactions on the sites rather than ions:

```
>X:Ca(0.5)
charge= 0      mole wt.= 20.0400 g
3 species in reaction
  1.000 >X:Na      .500 Ca++      -1.000 Na+
Selectivity coefficient= .0300
```

The sorbed ion activities can be expressed as either equivalent or molar fractions, since the corresponding reaction coefficients are the same on both sides of the reaction.

In  $K_d$  model datasets, surface species are formed by sorption of ions. Enter  $K_d$ 's as linear, not logarithmic, values:

```
>Ni++
charge= 2      mole wt.= 58.7100 g
1 species in reaction
  1.000 Ni++
Kd= .050
```

The  $K_d$  values are presented here in units of mol/g solid. A  $K_d$  value represents the ratio of an ion's sorbed concentration in mol/g solid to its free activity (not molality) in solution. In taking data from the literature, be careful to note the units used and make the appropriate conversion, accounting as necessary for the ion's activity coefficient and perhaps the extent of complexation in solution.

In Freundlich model datasets, surface species are formed by sorption of ions. Enter  $K_f$ 's as linear, not logarithmic values:

```
>Pb++  
  charge= 2      mole wt.= 207.2000 g  
  1 species in reaction  
    1.000 Pb++  
  Kf= .0025   nf= .800
```

The  $K_f$  values are presented here in units of mol/g solid. A  $K_f$  value represents the ratio of an ion's sorbed concentration in mol/g solid to its free activity (not molality) in solution, the latter raised to the Freundlich exponent,  $n_f$ . In taking data from the literature, be careful to note the units used and make the appropriate conversion, accounting as necessary for the ion's activity coefficient and perhaps the extent of complexation in solution.

## 4.5 Legacy dataset formats

Five legacy formats for the thermo databases may be used with current releases of the software. The legacy formats are labeled "apr18", "jan14", "dec99", "jun95", and "jan94".

GWB release 16 recognizes "may20" and earlier formats, whereas Release 12 works only with "jan14" and earlier formats. The differences between the legacy and current formats are summarized below.

### 4.5.1 Thermo data line

Dataset formats "dec99" and earlier do not include a line specifying the thermo dataset from which to draw species for the surface reactions, for use in the **TEdit** app.

### 4.5.2 Three-layer models

Dataset formats "jan14" and earlier do not support the triple-layer or CD-MUSIC surface models.

### 4.5.3 Polydentate sorption

Dataset formats "jan14" and earlier do not support specification of different conventions for polydentate complexes.

### 4.5.4 End line

Dataset formats "dec99" and earlier do not include an extra -end- line before the basis species section.

### 4.5.5 Charged uncomplexed sites

Dataset formats "jan14" and earlier do not support surface sites with non-zero charge.

#### 4.5.6 Site density units

Dataset formats “jan14” and earlier do not support specification of site density in sites/nm<sup>2</sup> units.

#### 4.5.7 Arbitrary reaction definition

Dataset formats “apr18” and earlier support reactions for two-layer, triple-layer, and CD-MUSIC complexes written in terms of the sorbing sites (surface basis species) as well as basis, redox, and aqueous species in the thermo dataset. Reactions cannot be written in terms of other surface species, such as protonated or deprotonated forms of the sorbing site.

#### 4.5.8 Temperature expansions

Dataset formats “apr18” and earlier support specifying a log  $K$  value and temperature derivative at 25°C for the two-layer, triple-layer, CD-MUSIC, and Langmuir models. They do not support the log  $K$  temperature polynomial nor the temperature range of validity allowed in the “may20” format.



## Report Command

---

The “report” command returns the results calculated by a GWB program. The command is available in programs **Rxn**, **SpecE8**, **React**, **X1t**, and **X2t**. After you have run the program (i.e., after you have selected **Run** → **Go**, or otherwise triggered the calculation), you can use the “report” command to retrieve the program results.

You can use the “report” command from the **Command** pane as a way to explore the calculation results interactively. More commonly, the command is used when writing control scripts (see [Control Scripts](#)), or when invoking the programs with the plug-in feature (see [Plug-in Feature](#)) or by remote control (see [Remote Control](#)), as a method of transmitting calculation results to the script or controlling program.

You might, for example, type

```
report pH
```

on **React**’s **Command** pane. The program would respond with the most recently calculated value of pH. The programs recognize a number of “report” command keywords, like “pH” in the example above; the keywords are listed in the table at the end of this chapter. Typing the command

```
report options
```

returns a list of available keywords.

Depending on the keyword, the “report” command may return a single value, several values, or a vector of values. You control the command’s response using arguments specific to a keyword; arguments are shown in boldface in the table at the end of the chapter. The “concentration” keyword is a good example of using arguments:

```

concentration <argument(s)> <name(s) | index> ...
  argument:      related arguments:
  original      <fluid | system | sorbed | stagnant>
  current       <fluid | system | sorbed>
  aqueous
  surf_species
  elements     <fluid | system | sorbed | stagnant>
  minerals
    
```

Selecting from the list of types, you can enter a command such as

```
report concentration aqueous
```

which will cause the program to return a vector of the concentrations of aqueous species in the system. The command

```
report aqueous
```

displays the names of those species.

As a second example, typing

```
report concentration original fluid
```

gives a vector of the concentrations of the original basis components in the fluid, the names of which are returned by the command

```
report basis original
```

Notably, the first example above returns concentrations of free species, whereas the second example returns the total or bulk concentrations of the components that make up the solution.

Continuing the first example, you can request the concentration of an individual species by name

```
report concentration aqueous Na+
```

or by vector position

```
report concentration aqueous 12
```

The GWB applications index vectors by offset, so the first entry is identified as "0", the second is "1", and so on.

You can stack arguments on a command line, so typing

```
report concentration aqueous H+ Na+ Cl-
```

prompts the program to return three values, one for each of the species listed. Typing

```
report concentration aqueous 0 1 2
```

also returns three values, for the first three entries in the vector of aqueous species.

In a client application, you may wish to work in terms of vector indices. The number of aqueous species, for example, is returned by the command

```
report naqueous
```

By writing a loop in which a counter  $i$  varies from zero to the number of aqueous species, less one, you can use the command

```
report aqueous i  
report concentration aqueous i
```

to retrieve the names and concentrations of the aqueous species, one at a time.

When using the “report” command, remember to enclose multi-word arguments, such as species names, in quotes, just as you would in any GWB command. For example, the command

```
report mass_reacted "Albite low"
```

gives the expected result.

The “report” command normally returns values in terms of a default unit set for each keyword, as shown in the table at the end of this chapter. To find the default unit for a given “report” keyword, type a command of the form

```
report get_default_units concentration
```

In this case, the application will respond that results for concentration are reported by default in molal units.

You may nonetheless request results in any of the units listed in the [Units Recognized](#) chapter in this guide. To do so, affix the unit name to the end of a “report” command. For example,

```
report concentration aqueous mmol/kg
```

returns the concentration of each aqueous species, in units of mmol/kg. If the unit conversion fails, the program will respond with “ANULL”, the flag for an undefined value.

You can use the “set\_units” keyword to set the application to return results invariably in terms of a specific unit. To override the default units in this way, enter a command such as

```
report set_units "mmol/kg"
```

Having issued this command, unit conversion for commands such as “report temperature” will fail until you have unset the option. To return to default behavior, enter

```
report set_units ?
```

The command

```
report get_units
```

shows the current setting for the overriding unit, if one has been set.

To change the number of significant digits in the numerical results returned by the “report” command, type, for example

```
report set_digits 8
```

By default, the applications return four significant digits.

In **X1t**, you can specify the node of interest by typing

```
report set_node 5
```

for example. The command above tells the program to return values associated with the node with index 5. Similarly, in **X2t**, you might type

```
report set_node 8 8
```

giving first an  $x$ -direction, then  $y$ -direction index. Node indices vary from 0 to the total number of nodes, less one. Indexing starts in the bottom left corner of the domain and increases from left to right in the bottom row, then the next highest row, and so on.

Finally, note that a subset of keywords gives access to information even if the run is not complete. The “database” keyword provides information about the thermo dataset in use, while “configuration” and “constraints” give information about how the user has configured the basis.



(This page left blank.)

Keyword	Arguments	Description
<b>activity</b>	<aqueous   surf_species   end_members> <name(s)   index> . . .	Species, end member activities
<b>alkalinity</b>		Carbonate alkalinity
<b>aqueous</b>	<index> . . .	Names of aqueous species
<b>basis</b>	<original   current> <index> . . .	Names of basis entries
<b>biomass</b>	<reactant(s)   index> . . .	Biomass concentration
<b>boltzman</b>	<surf_species   index> . . .	Boltzman factors for surface species
<b>bulk_volume</b>		Bulk volume of nodal block
<b>cat_area</b>	<reactant(s)   index> . . .	Areas of catalyzing surfaces
<b>charge</b>	<original   current   aqueous   surf_species><name(s)   index> . . .	Charge on components or species
<b>chlorinity</b>		Chlorinity
<b>colloids</b>	<index> . . .	Names of mobile colloids
<b>coef_dispersion</b>		Coefficient of dispersion
<b>concentration</b>	<b>original</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>current</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>elements</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>aqueous</b> <b>surf_species</b> <b>minerals</b> <equilibrium   kinetic   both   all> <b>end_members</b> <equilibrium   kinetic   both   all> <name(s)   index> . . .	Concentration of components, aqueous or surface species, minerals, end members, or elements
<b>configuration</b>	<basis   swap   type   unit   scale   as> <names(s)   index> . . .	Basis configuration, including entry, swap species, constraint type, unit, scale, and "as" unit
<b>constraints</b>	<name(s)   index> . . .	Values constraining each basis entry
<b>contact_area</b>	<reactant(s)   index> . . .	Contact areas for kinetic gases
<b>couples</b>	<index> . . .	Names of redox couples

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
	double		✓	✓	✓	✓
mg/kg sol'n as CaCO <sub>3</sub>	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	strings		✓	✓	✓	✓
mg/kg	double			✓	✓	✓
	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
cm <sup>2</sup>	double			✓	✓	✓
	double		✓	✓	✓	✓
molal	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
cm <sup>2</sup> /s	double				✓	
molal	double		✓	✓	✓	✓
	strings		✓	✓		
	double		✓	✓		
cm <sup>2</sup>	double			✓	✓	✓
	strings		✓	✓	✓	✓

Keyword	Arguments	Description
<b>database</b>	<elements   basis   redox   aqueous   electron   minerals   end_members   solid_solutions   gas   oxide>	Names of entries in the thermo dataset, whether included in the current simulation or not
<b>discharge</b>		Specific discharge
<b>Deltat</b>		Length of current time step
<b>EC</b>		Electrical conductivity
<b>efflux</b>	<name(s)   index> . . .	The net flux of original basis components out of the domain
<b>Eh</b>	<system   couples> <name(s)   index> . . .	The system Eh or Nernst Eh values for redox couples
<b>elements</b>	<index> . . .	Names of elements
<b>end_members</b>	<index> . . .	End member names
<b>equil_eqn</b>		Equilibrium equation as a text string
<b>equil_favors</b>		Whether reaction products or reactants are favored
<b>equil_temp</b>		Equilibrium temperature
<b>exchange_capacity</b>	<surface_type(s)   index> . . .	Capacity of ion exchange surface
<b>freeflowing</b>		Volume of free-flowing zone in nodal block
<b>FA</b>	<reactant(s)   index> . . .	Kinetic factor for electron acceptance by microbes
<b>FD</b>	<reactant(s)   index> . . .	Kinetic factor for electron donation by microbes
<b>fugacity</b>	<gas(es)   index> . . .	Gas fugacities
<b>gamma</b>	<aqueous   surf_species   end_members> <name(s)   index> . . .	Species, end member activity coefficients
<b>gas_pressure</b>	<gas(es)   index> . . .	Gas partial pressures
<b>gases</b>	<index> . . .	Names of gases
<b>get_default_units</b>	<keyword>	Default unit for command
<b>get_units</b>		Current unit, if set

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
	strings	✓	✓	✓	✓	✓
cm <sup>3</sup> /cm <sup>2</sup> s	double				✓	✓
s	double			✓	✓	✓
μS/cm or umho/cm	double		✓	✓	✓	✓
moles	double				✓	✓
volts	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	string	✓				
	string	✓				
	string	✓				
eq	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
	double			✓	✓	✓
	double			✓	✓	✓
	double		✓	✓	✓	✓
	double		✓	✓	✓	✓
bar	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	string	✓	✓	✓	✓	✓
	string	✓	✓	✓	✓	✓

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>hardness</b>		Hardness
<b>hardness_carb</b>		Carbonate
<b>hardness_ncarb</b>		Non-carbonate
<b>hyd_pot</b>		Hydraulic potential
<b>imbalance</b>		Charge imbalance
<b>imbalance_error</b>		Error percentage
<b>inert_volume</b>		Inert volume in system
<b>influx</b>	<name(s)   index> . . .	The net flux of original basis components into the domain
<b>IS or Tionst</b>		System ionic strength
<b>isotopes</b>	<symbols>	Names of or symbols for isotope systems
<b>iterations</b>		Number of iterations required for Newton-Raphson to converge
<b>Kd</b>	<name(s)   index> . . .	Net $K_d$ for sorption of original basis entries onto all surfaces
<b>logfO2</b>		Log fugacity of O <sub>2</sub>
<b>logk</b>		Log equilibrium constant
<b>logks</b>		List of log $K$ 's at principle temperatures
<b>logQoverK or SI</b>	<minerals   solid_solutions   reactants> <name(s)   index> . . .	Saturation index for minerals, solid solutions, or reactants
<b>mass</b>	<b>original</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>current</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>elements</b> <fluid   system   rock   sorbed   stagnant   colloid> <b>aqueous</b> <b>surf_species</b> <b>minerals</b> <equilibrium   kinetic   both   all> <b>end_members</b> <equilibrium   kinetic   both   all> <name(s)   index> . . .	Mass of components, aqueous or surface species, minerals, end members, or elements

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
mg/kg sol'n as CaCO <sub>3</sub>	double		✓	✓	✓	✓
mg/kg sol'n as CaCO <sub>3</sub>	double		✓	✓	✓	✓
mg/kg sol'n as CaCO <sub>3</sub>	double		✓	✓	✓	✓
bar	double				✓	✓
eq/kg	double		✓	✓	✓	✓
% error	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
moles	double				✓	✓
molal	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	int		✓	✓		
liter/kg	double		✓	✓	✓	✓
log fugacity	double		✓	✓	✓	✓
	double	✓				
	double	✓				
log $Q/K$	double		✓	✓	✓	✓
moles	double		✓	✓	✓	✓

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>mass_reacted</b>	<reactant(s)   index> . . .	Mass of a reactant that has reacted
<b>mass_remaining</b>	<reactant(s)   index> . . .	Mass of a reactant remaining to react
<b>minerals</b>	< <b>equilibrium</b>   <b>kinetic</b>   <b>both</b>   <b>all</b> > <index> . . .	Names of minerals
<b>mineral_mass</b>		Mass of minerals, system or block
<b>mineral_volume</b>		Volume of minerals, syst. or block
<b>mixing_fraction</b>		Mixing fraction in flash model
<b>mobility</b>	<surface_type(s)   index> . . .	Mobility of colloidal surfaces
<b>mv</b>	<mineral(s)   index> . . .	Mineral molar volume
<b>mw</b>	< <b>original</b>   <b>current</b>   <b>aqueous</b>   <b>surf_species</b>   <b>elements</b>   <b>minerals</b>   <b>end_members</b>   <b>gases</b> > <name(s)   index> . . .	Mole weight of components, species, or elements
<b>naqueous</b>		Number of aqueous species
<b>nbasis</b>		Number of basis entries
<b>ncolloids</b>		Number of mobile colloids
<b>ncouples</b>		Number of redox couples
<b>nelements</b>		Number of elements
<b>nend_members</b>		Number of end members
<b>ngases</b>		Number of gases
<b>nisotopes</b>		Number of isotope systems
<b>nlogks</b>		Number of log <i>K</i> values in list
<b>nminerals</b>	< <b>equilibrium</b>   <b>kinetic</b>   <b>both</b>   <b>all</b> >	Number of minerals
<b>Nnode</b>		Number of nodal blocks
<b>nreactants</b>	< <b>simple</b>   <b>fixed</b>   <b>sliding</b>   <b>kin_mineral</b>   <b>kin_endmember</b>   <b>kin_redox</b>   <b>microbial</b>   <b>kin_aqueous</b>   <b>kin_surface</b>   <b>kin_gas</b>   <b>all</b> >	Number of reactants, kinetic reactions
<b>nsolid_solutions</b>		Number of solid solutions



Default units	Return type	Rxn	SpecE8	React	X1t	X2t
moles	double			✓	✓	✓
moles	double			✓	✓	✓
	strings		✓	✓	✓	✓
kg	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
	double			✓		
	double		✓	✓	✓	✓
cm <sup>3</sup> /mol	double		✓	✓	✓	✓
g/mol	double		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int	✓				
	int		✓	✓	✓	✓
	int				✓	✓
	int			✓	✓	✓
	int		✓	✓	✓	✓

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>nsorbed</b>		Number of original basis species that sorb
<b>nsorbing_surfaces</b>		Number of sorbing surface types
<b>nstagnant</b>		One for dual porosity, else zero
<b>nsurf_species</b>		Number of surface species
<b>Nx</b>		Number of nodes along <i>x</i>
<b>Ny</b>		Number of nodes along <i>y</i>
<b>options</b>		List of keywords for the report command
<b>pe</b>	< <b>system</b>   <b>couples</b> > <name(s)   index> . . .	The system pe or theoretical pe for redox couples
<b>permeability</b>		Sediment permeability
<b>pH</b>		System pH
<b>polyfit</b>		Coefficients for polynomial fit of log <i>K</i> to temperature
<b>porosity</b>		Porosity
<b>pressure</b>		Pressure
<b>PV</b>		Pore volumes displaced
<b>QoverK</b>	< <b>minerals</b>   <b>solid_solutions</b>   <b>reactants</b> > <name(s)   index> . . .	<i>Q/K</i> for a mineral, solid solution, or reactant
<b>rate_con</b>	<reactant(s)   index> . . .	Rate constants for kinetic reactions
<b>ratecon_unit</b>	<reactant(s)   index> . . .	Units of kinetic rate constants
<b>reactant_area</b>	<reactant(s)   index> . . .	Surface areas of kinetic minerals
<b>reactant_type</b>	<reactant(s)   index> . . .	simple, fixed, sliding, kin_mineral, microbial, and so on
<b>reactants</b>	< <b>simple</b>   <b>fixed</b>   <b>sliding</b>   <b>kin_mineral</b>   <b>kin_endmember</b>   <b>kin_redox</b>   <b>microbial</b>   <b>kin_aqueous</b>   <b>kin_surface</b>   <b>kin_gas</b>   <b>all</b> > <index> . . .	Names of reactants and kinetic reactions
<b>reaction</b>		Chemical reaction as a text string

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int		✓	✓	✓	✓
	int				✓	✓
	int					✓
	strings	✓	✓	✓	✓	✓
	double		✓	✓	✓	✓
darcy $\approx \mu\text{m}^2$	double		✓	✓	✓	✓
	double		✓	✓	✓	✓
	doubles	✓				
volume fraction	double		✓	✓	✓	✓
bars	double		✓	✓	✓	✓
	double			✓	✓	✓
	double			✓	✓	✓
(varies)	double			✓	✓	✓
	strings			✓	✓	✓
cm <sup>2</sup>	double			✓	✓	✓
	strings			✓	✓	✓
	strings			✓	✓	✓
	string	✓				

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>rxn_rate</b>	<reactant(s)   index> . . .	Reaction rates
<b>set_digits</b>	<significant digits>	Set number of significant digits in results
<b>set_node</b>	<node index   x y>	Results are associated with the node index or the x and y node indices
<b>set_units</b>	<unit   ?>	Report results in a unit different than the default
<b>SIS or Sionst</b>		Stoichiometric ionic strength
<b>solid_solutions</b>	<index> . . .	Names of solid solutions
<b>soln_compressibility</b>		Solution compressibility
<b>soln_density</b>		Solution density
<b>soln_expansivity</b>		Solution thermal expansivity
<b>soln_mass</b>		Solution mass
<b>soln_viscosity</b>		Viscosity of fluid
<b>soln_volume</b>		Volume of fluid
<b>sorb_area</b>	<surface_type(s)   index> . . .	Areas of sorbing surfaces
<b>sorbed</b>	<index> . . .	Names of original basis species that sorb
<b>sorption_capacity</b>	<surface_type(s)   index> . . .	Capacity of a Langmuir surface
<b>stagnant</b>		Volume of stagnant zone in nodal block
<b>success</b>		Returns a value of one if the GWB application has successfully completed a calculation, zero if it has not or if the last calculation failed
<b>surf_charge</b> or <b>surf_charge0</b>	<surface_type(s)   index> . . .	Electrical charge at 0-plane of a sorbing surface
<b>surf_chargeb</b>	<surface_type(s)   index> . . .	Electrical charge at $\beta$ -plane
<b>surf_charged</b>	<surface_type(s)   index> . . .	Electrical charge at $d$ -plane
<b>surf_potential</b> or <b>surf_potential0</b>	<surface_type(s)   index> . . .	Electrical potential at 0-plane of a sorbing surface
<b>surf_potentialb</b>	<surface_type(s)   index> . . .	Electrical potential at $\beta$ -plane
<b>surf_potentiald</b>	<surface_type(s)   index> . . .	Electrical potential at $d$ -plane

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
mol/s	double			✓	✓	✓
	int	✓	✓	✓	✓	✓
	int				✓	✓
	string	✓	✓	✓	✓	✓
molal	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
bar <sup>-1</sup>	double		✓	✓	✓	✓
g/cm <sup>3</sup>	double		✓	✓	✓	✓
°C <sup>-1</sup>	double		✓	✓	✓	✓
kg	double		✓	✓	✓	✓
cp	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
cm <sup>2</sup>	double		✓	✓	✓	✓
	strings		✓	✓	✓	✓
mol	double		✓	✓	✓	✓
cm <sup>3</sup>	double		✓	✓	✓	✓
	int	✓	✓	✓	✓	✓
μC/cm <sup>2</sup>	double		✓	✓	✓	✓
μC/cm <sup>2</sup>	double		✓	✓	✓	✓
μC/cm <sup>2</sup>	double		✓	✓	✓	✓
mV	double		✓	✓	✓	✓
mV	double		✓	✓	✓	✓
mV	double		✓	✓	✓	✓

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>surf_species</b>	<index> . . .	Names of surface species
<b>surf_type</b>		Types of reacting surfaces
<b>surfaces</b>		Names of reacting surfaces
<b>TDS</b>		Total dissolved solids
<b>temperature</b> or <b>T</b>		Temperature
<b>temps</b>		List of principle temperatures
<b>Tend</b>		Final time of simulation
<b>Time</b>		Current point in time
<b>total_biomass</b>		Biomass in system or block
<b>total_reacted</b>		Mass reacted into system or block
<b>TPF</b>	<reactant(s)   index> . . .	Thermodynamic potential factor
<b>Tstart</b>		Beginning time of simulation
<b>velocity</b>		Fluid velocity
<b>Watact</b>		Activity of water
<b>watertype</b>		Ion type of water
<b>Wmass</b>		Water mass
<b>xcoef_dispersion</b>		x coefficient of dispersion
<b>xdischarge</b>		x specific discharge
<b>Xfree</b>		Free-flowing fraction
<b>Xi</b>		Reaction progress
<b>xpermeability</b>		x permeability
<b>xsorbed</b>	<name(s)   index> . . .	Sorbed fraction of an original basis entry
<b>xss</b>	<equilibrium   kinetic   both   all> <name(s)   index> . . .	End member mole fractions
<b>xvelocity</b>		Fluid velocity along x
<b>xycoef_dispersion</b>		xy coefficient of dispersion
<b>ycoef_dispersion</b>		y coefficient of dispersion
<b>ydischarge</b>		y specific discharge

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
	strings		✓	✓	✓	✓
	strings		✓	✓	✓	✓
	strings		✓	✓	✓	✓
mg/kg	double		✓	✓	✓	✓
°C	double		✓	✓	✓	✓
°C	doubles	✓				
s	double			✓	✓	✓
s	double			✓	✓	✓
mg/kg	double			✓	✓	✓
g	double			✓	✓	✓
	double			✓	✓	✓
s	double			✓	✓	✓
cm/s	double				✓	✓
	double		✓	✓	✓	✓
	string		✓	✓	✓	✓
kg	double		✓	✓	✓	✓
cm <sup>2</sup> /s	double					✓
cm <sup>3</sup> /cm <sup>2</sup> s	double					✓
	double		✓	✓	✓	✓
	double			✓	✓	✓
darcy $\approx \mu\text{m}^2$	double					✓
	double		✓	✓	✓	✓
	double		✓	✓	✓	✓
cm/s	double					✓
cm <sup>2</sup> /s	double					✓
cm <sup>2</sup> /s	double					✓
cm <sup>3</sup> /cm <sup>2</sup> s	double					✓

<b>Keyword</b>	<b>Arguments</b>	<b>Description</b>
<b>ypermeability</b>		y permeability
<b>yvelocity</b>		Fluid velocity along y
<isotope   <b>Hydrogen-2</b>   <b>Carbon-13</b>   <b>Oxygen-18</b>   <b>Sulfur-34</b>   symbol   <b>2-H</b>   <b>13-C</b>   <b>18-O</b>   <b>34-S</b> >	< <b>fluid</b>   <b>rock</b>   <b>sorbate</b>   <b>system</b> > < <b>solvent</b>   <b>aqueous</b>   <b>minerals</b>   <b>end_members</b>   <b>gases</b>   <b>surf_species</b>   <b>reactants</b> > <name(s)   index> . . .	Isotopic compositions of various aspects of system



---

Default units	Return type	Rxn	SpecE8	React	X1t	X2t
darby $\approx \mu\text{m}^2$	double					✓
cm/s	double					✓
$\delta$ (‰)	double		✓	✓	✓	✓

---



# Control Scripts

---

When you read an ordinary script in one of the GWB applications (**Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **Phase2**, **X1t**, or **X2t**), the application steps through the script line-by-line, executing the commands encountered, until it reaches the script's end. A control script differs from an ordinary script in that it can contain control statements, such as assignments, loops, and if-then-else constructs.

This chapter describes how to set up a control script and gives an example of such a script. The [Multiple Analyses](#) chapter in this guide shows a further example of how control scripts can be applied, in this case to add the results of GWB calculations to a spreadsheet containing the results of a number of chemical analyses.

A control script can occupy an entire script file, or just a portion of one. The control script is preceded by the statement

```
script
```

The control script terminates at the end of the file, or with the statement

```
script end
```

The lines within the control script may be either application commands (e.g., commands recognized by **React**) or control statements.

When writing file names within control scripts, following Tcl syntax, you use double rather than single backslashes (i.e., “\\” instead of “\”) as separators. In addition, you enclose filenames containing spaces or special characters with braces (“{...}”). For example, the application commands

```
read GWB\react.rea  
data "c:\Program Files\GWB\gtdata\thermo.data"
```

would appear as

```
read GWB\\react.rea  
data {"c:\\Program Files\\GWB\\gtdata\\thermo.data"}
```

within a control script.

## 6.1 Control statements

The applications recognize control statements in the form of Tcl commands. Tcl (pronounced “tickle”) is an open-source scripting language, designed to be easy to learn and use. You can find information about the Tcl syntax on various web sites such as [www.tcl.tk](http://www.tcl.tk) and [mini.net/tcl](http://mini.net/tcl), as well as a number of widely available textbooks (you can search on “Tcl” at [www.amazon.com](http://www.amazon.com)).

In Tcl, you define a variable with the “set” command, and use a “\$” in front of the variable name to reference its value. For example, the commands

```
set pH 4.5
set label "The pH is"
puts "$label $pH"
```

assign a value of 4.5 to a variable named “pH” and a literal string to variable “label”, and then write them to the screen using the “puts” command.

Other useful commands include

for	Define a loop
while	Define a loop
eval	Evaluate a command
expr	Evaluate an expression
if {...} elseif {...} else	Set an if-then-else block
proc	Define a procedure
open	Open a file
close	Close a file
gets	Get input from a file or prompt
puts	Output data to a file or prompt
info commands	List Tcl commands

If you write procedures (using the “proc” command), you should be careful not to name them using GWB keywords, or the names of species in the thermo dataset. For example, if you were to name a procedure “Fe++”, you would no longer be able to constrain the concentration of ferrous iron.

Before beginning to write a command script, you will want to consult a more complete Tcl reference to learn a few details of the language syntax.

## 6.2 Interacting with the application program

You can use the Tcl “eval” command to construct a GWB command and execute it within the GWB application. In a **React** control script, for example, the Tcl command

```
eval {"pH =" $value}
```

causes pH in the **React** run to be set to the contents of variable “value”.

You can interrogate the GWB application program about its calculation results using the “report” command. For example, once you have calculated a geochemical model using **React**, the command

```
report pH
```

returns the predicted pH. You can set the value of a variable “new\_pH” in a command script to this value with the Tcl command

```
set new_pH [eval report pH]
```

### 6.3 Example control script

The following control script uses program **React** to search for the rate constant that describes the results of a hydrothermal experiment. In the experiment, 1 kg of water with an initial silica concentration of 1 mg/kg reacts at 100°C with 5000 g of quartz.

After 5 days, the silica concentration is observed to be .55 *mmolal*. The script searches for a rate constant in the range  $10^{-16}$  to  $10^{-14}$  mol/cm<sup>2</sup> sec that explains this result. In the script, **React** commands are shown in bold face, for clarity, whereas Tcl commands are shown in normal typeface.

```
time begin = 0 days, end = 5 days
T = 100
SiO2(aq) = 1 mg/kg
react 5000 g Quartz
kinetic Quartz surface = 1000

script start
proc find_ratecon {low high species conc} {
  set gotit 0
  for {set i 0} {$i < 50} {incr i 1} {
    set test [expr {($low + $high) / 2}]
    eval {kinetic Quartz rate_con = $test}
    go
    set back [eval report molality $species]
    if {[expr {abs($back - $conc)}] < 1e-6} {
      set gotit 1
      break
    }
    if {$back < $conc} {
      set low $test
    } else {set high $test}
  }
  if {$gotit} {
```

```
    puts "The optimum rate constant is $test mol/cm2 s"
    puts "The control script converged in $i iterations"
  } else {
    puts "The control script did not converge"
  }
}
# Find the rate constant for Quartz dissolution that gives a
# SiO2(aq) concentration of .00055 molal after 5 days.
find_ratecon 1e-16 1e-14 "SiO2(aq)" .00055
script end
```

## 6.4 Tcl license agreement

GWB control scripts are evaluated according to the Tcl scripting language, using open source software distributed under the following license agreement:

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files. The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply. IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS. GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

# Plug-in Feature

---

The GWB plug-in feature is implemented as a Dynamic-Link Library (DLL). For ease of use, GWB provides wrapper classes for C++, Fortran, and Python that handle loading the DLL, binding to the needed functions, and conversion to C data types. You might create your own wrapper for the plug-in feature in other languages.

You can plug-in the capabilities of **Rxn**, **SpecE8**, **React**, **Phase2**, **X1t**, and **X2t** to a program that you write using the GWB plug-in feature.

In writing a program of your own, for example, you might need to determine the saturation state of calcite in a fluid of arbitrary composition. Instead of developing code to calculate the distribution of mass and mineral saturation states in a fluid, you could use **SpecE8** from within your program to do the work for you.

Similarly, you could use the plug-in feature to balance reactions with **Rxn**, or figure the results of irreversible reaction paths with **React**.

In each case, you would configure the GWB plug-in with text commands, trigger the calculation with a “go” command, and then retrieve the calculation results to use for your own purposes.

You transfer the results from the GWB plug-in with the “results” function, an interface for the “report” command (documented in the [Report Command](#) chapter in this **GWB Reference Manual**).

You may also simply read datasets, such as “SpecE8\_output.txt”, produced by the GWB applications, into your program.

The following sections describe and give examples of how to take advantage of the plug-in feature when developing your software with specific languages and compilers.

## 7.1 C++

GWB provides a GWBplugin wrapper class contained in the header file “GWBplugin.h”, and the GWBplugin.dll export library “GWBplugin.lib” to link to. GWBplugin.h is installed in the “src” subdirectory of the GWB installation directory while the GWBplugin.lib is installed to the main GWB installation directory. In order to locate the GWB DLLs the GWBplugin class uses, you must add the GWB installation directory to the PATH environment variable.

This is the C++ wrapper class provided in GWBplugin.h:

```
// GWBplugin.h

#define ANULL -999999.0    // marker for an undefined value

class GWBplugin {
public:
    GWBplugin();
    ~GWBplugin();
    int initialize(const char* app_name, const char* file_name = NULL,
                  const char* cmds = NULL);
    int exec_cmd(char* uline);
    int results(void* data, const char* value, const char* units = NULL,
               int ix = 0, int jy = 0);
};
```

### 7.1.1 Initializing the GWB application

Within your code, first create a GWBplugin object.

```
#include "GWBplugin.h"

GWBplugin myPlugin;
```

Next, use the “initialize” function to start the GWB application of interest by passing the application name, an optional output file name, and any command-line type arguments. The “initialize” function must be called before calling any of the other functions.

```
int initialize (
    const char* app_name,
    const char* file_name = NULL,
    const char* cmds = NULL
);
```

Parameters:



**app\_name**

String containing the GWB application name - "rxn", "spece8", "react", "x1t", or "x2t".

**file\_name** (optional)

String containing the name of the file you want the GWB output written to. This can be NULL or an empty string if you do not want the output to be written to a file.

**cmds** (optional)

String containing command-line options you could normally pass to the application when running it from the command-line. This can be NULL or an empty string.

Command-line options:

-cd	Change the working directory to the directory containing the input script specified with the <code>-i</code> option.
-nocd	Do not change the working directory.
-i <input_script>	Read initial input commands from the specified file.
-gtd <gtdata_dir>	Set directory to search for thermodynamic datasets.
-cond <cond_data>	Set the dataset for calculating electrical conductivity.
-d <thermo_data>	Set the thermodynamic dataset.
-s <surf_data>	Set a dataset of surface sorption reactions.
-iso <isotope_data>	Set a dataset of isotope fractionation factors.

**Return value**

Non-zero on success and zero on failure.

**Examples**

Some examples of how to start the GWB plug-in in various ways:

```
// plug-in SpecE8 with no output written and no command-line options
int success = myPlugin.initialize("spece8");

// plug-in React with output written to output.txt and no command-line options
int success = myPlugin.initialize("react", "output.txt");

// plug-in X1t with no output written, no working directory change,
// and input read from pb_contam.x1t
int success = myPlugin.initialize("x1t", NULL, "-nocd
-i \\\"c:/program files/gwb/script/pb_contam.x1t\\\"");
```

## 7.1.2 Configuring and executing calculations

Use the "exec\_cmd" function to transmit commands to the GWB plug-in. Each application has a chapter in the **GWB Command Reference** that is a comprehensive guide to the commands available. Use these commands to configure the application and then send a "go" command to trigger the calculations.

```
int exec_cmd (
    char* uline // command string to be sent to the GWB application
);
```

**Return value**

Non-zero on success and zero on failure.

**Examples**

```
myPlugin.exec_cmd("3 mmol H+");
myPlugin.exec_cmd("2 mmol Ca++");
myPlugin.exec_cmd("5 mmolar Cl-");
myPlugin.exec_cmd("go");           // trigger the calculation
```

**7.1.3 Retrieving the results**

Transfer calculation results from the GWB application to your program with the “results” function. The keywords, arguments, default units, and return types are the same as those listed in the table in the [Report Command](#) chapter of this reference manual.

To use the “results” function, you provide the address of a data block to fill, along with the report command and keywords, optional desired units, and the node location of choice (X1t and X2t only).

```
int results(
    void* data,
    const char* value,
    const char* units = NULL,
    int ix = 0,
    int jy = 0
);
```

Parameters:

**data**

Address of data block to fill. This can be NULL to determine data block size.

**value**

String containing the report command keyword and arguments.

**units** (optional)

String containing the units you want the results returned in. This can be NULL or an empty string if you want the results returned in the default units.

**ix** (optional)

X node position. This is only used when running X1t and X2t, otherwise it is ignored.

**jy** (optional)

Y node position. This is only used when running X2t, otherwise it is ignored.

**Return value**

The number of values written (or to be written) to the data block.

**Remarks**

To determine the size of data block you will need, first call this function with the data parameter as NULL and with the rest of the parameters filled. If you know that the report command you are using only returns a single value, you can simply pass a pointer to the correct data type. See the [Report Command](#) chapter for details on data types and available keywords.

## Examples

```
// get aqueous species names
int ndata = myPlugin.results(NULL, "species");
char** Species = new char*[ndata];
myPlugin.results(Species, "species");

// get aqueous species concentrations in mg/kg
double* Conc = new double[ndata];
myPlugin.results(Conc, "concentration aqueous", "mg/kg");

// get pH at node 3,5
double pH = ANULL;
myPlugin.results(&pH, "pH", NULL, 3, 5);
```

If the command fails for any reason, for example if the requested data doesn't exist or the specified unit conversion failed, the data will be filled with ANULL (-999999.0).

### 7.1.4 C++ code examples using the plug-in feature

Normally you would use the GWB plug-in within your program with no output being written to a file. The following is an example of this:

```
#include "GWBplugin.h"

int main(int argc, char* argv[ ])
{
    // create the plug-in object
    GWBplugin* myGWBrun = new GWBplugin();

    // start the GWB program
    if (myGWBrun->initialize("spece8", NULL, "-nocd")) { // started successfully

        printf("Beginning run.\n");
        // configure SpecE8 and trigger calculation
        const char* cmds[3] = {"pH = 8", "molality Cl- = .05", "go"};
        for(int i = 0; i < 3; i++)
            myGWBrun->exec_cmd(cmds[i]);
        printf("Finished run.\n\n");

        // retrieve results
        double pH;
        myGWBrun->results(&pH, "pH");

        double Cl;
        myGWBrun->results(&Cl, "concentration Cl-"); // in default units
        printf("concentration of Cl- in molal is %10.4g\n", Cl);
        myGWBrun->results(&Cl, "concentration Cl-", "mg/kg"); // in different units
```

```
printf("concentration of Cl- in mg/kg %10.4g\n", Cl);

// get size of data
int nspec = myGWBrun->results(NULL, "species");

// create data blocks
const char** Name = new const char*[nspec];
double* Spec = new double[nspec];

// send data pointer with keyword and arguments
myGWBrun->results(Name, "species");
myGWBrun->results(Spec, "concentration aqueous", "mg/kg");

printf("\n There are %d aqueous species\n\n", nspec);
for(int i = 0; i < nspec; i++)
    printf("%-4s = %10.4g mg/kg\n", Name[i], Spec[i]);

delete[ ] Name;
delete[ ] Spec;
}
else {
    // handle failure to start within your program
}
delete myGWBrun;
return 0;
}
```

To familiarize yourself with the plug-in feature, you might want the GWB program's normal output and results to be written to the console and to text files. The following code shows examples of this:

```
#include "GWBplugin.h"
#include <stdio.h>

int main(int argc, char* argv[ ])
{
    fprintf(stdout, "Starting program SpecE8\n");

    GWBplugin* myGWBrun = new GWBplugin();

    if (myGWBrun->initialize(
        "spece8",
        "test_output.txt",
        "-nocd -i \"C:/Program Files/Gwb/Script/Freshwater.sp8 \"
        -s \"C:/Program Files/Gwb/Gtdata/FeOH.sdat\"")
    {
        // started successfully

    }

    fprintf(stdout, "writing output to test_output.txt\n");
}
```

```

myGWBrun->exec_cmd("show surfaces");// write to output file

fprintf(stdout, "Executing test\n");

FILE *fp;
if ((fp=fopen("test_results.txt", "w")) == NULL) {
    fprintf(stderr, "can't open test_results.txt\n");
}
else {
    fprintf(stdout, "writing results to test_results.txt\n");

    char* cmds[3] = {"pH = 8", "molality Cl- = .05", "go"};
    for(int i = 0; i < 3; i++)
        myGWBrun->exec_cmd(cmds[i]);

    double pH;
    myGWBrun->results(&pH, "pH");
    fprintf(fp, "pH = %.1f\n", pH);

    double Cl;
    myGWBrun->results(&Cl, "concentration Cl-");
    fprintf(fp, "Cl = %12.5e molal\n", Cl);

    myGWBrun->results(&Cl, "concentration Cl-", "mg/kg");
    if(Cl != ANULL)
        fprintf(fp, "Cl = %12.5e mg/kg\n\n", Cl);
    else
        fprintf(fp, "unit conversion failed - Cl = ANULL\n\n");

    int nspec = myGWBrun->results(NULL, "species");
    const char** Name = new const char*[nspec];
    double* Spec = new double[nspec];

    if (myGWBrun->results(Name, "species")) {
        if (myGWBrun->results(Spec, "concentration aqueous")) {
            for (int i = 0; i < nspec; i++)
                fprintf(fp, "%-32s %12.5e molal\n", Name[i], Spec[i]);
        }
        if (myGWBrun->results(Spec, "concentration aqueous", "mg/kg")) {
            for (int i = 0; i < nspec; i++)
                fprintf(fp, "%-32s %12.5e mg/kg\n", Name[i], Spec[i]);
        }
    }
}

if (fp)
    fclose(fp);

```

```
        delete[ ] Name;
        delete[ ] Spec;
    }
}
else
    fprintf(stderr, "SpecE8 failed to start\n");

delete myGWBrun;

fprintf(stdout, "press return to exit> ");
getchar();

return 0;
}
```

### 7.1.5 C++ compiling and linking

The GWB plug-in has been tested on C++ compilers from Microsoft, Intel, and GCC. The version of the compiler you are using must be the same as the version of GWB installed (32-bit vs. 64-bit).

To compile the GWBplugin Example1 on the command line with the Microsoft or Intel compiler, follow these steps:

```
// open the Microsoft Visual Studio or Intel Command Prompt

// create a working folder and change to that folder
mkdir "%homepath%\GWBplugin"
cd "%homepath%\GWBplugin"

// copy the "src" folder from GWB installation (default install path shown)
copy /Y "C:\Program Files\GWB\src"

// copy the GWBplugin.lib file from GWB installation (default install path shown)
copy /Y "C:\Program Files\GWB\gwbplugin.lib"

// add the GWB installation folder to your path
set path=C:\Program Files\GWB;%path%

// compile the example file and tell the compiler to use the GWBplugin library
cl GWBplugin_Cpp_Example1.cpp GWBplugin.lib // Microsoft
// or
icl GWBplugin_Cpp_Example1.cpp GWBplugin.lib // Intel

// run the example program
GWBplugin_Cpp_Example1.exe
```

To compile the GWBplugin Example1 on the command line with MinGW, MSYS, and g++, follow these steps:

```
// launch the MinGW shell

// create a working folder and change to that folder
mkdir -p ~/GWBplugin
cd ~/GWBplugin

// copy the "src" folder from GWB installation (default install path shown)
cp /c/program\ files/gwb/src/* .

// copy the GWBplugin.lib file from GWB installation (default install path shown)
cp /c/program\ files/gwb/gwbplugin.lib .

// add the GWB installation folder to your path
PATH=/c/program\ files/gwb:$PATH

// compile the example file and tell the compiler to use the GWBplugin library
g++ GWBplugin_Cpp_Example1.cpp GWBplugin.lib -o
    GWBplugin_Cpp_Example1.exe           // all on one line

// run the example program
./GWBplugin_Cpp_Example1.exe
```

To compile the GWBplugin Example1 in Microsoft Visual Studio, follow these steps:

```
// open Visual Studio

// create a new project (Ctrl+Shift+N)

// select the "Visual C++" project type and use the "Empty Project" template

// select Project->Add Existing Item... (Shift+Alt+A)
// browse to the "src" subfolder of the GWB installation, select the
// "GWBplugin_Cpp_example1.cpp" file and click "Add"

// open Project->Properties->Configuration Properties

// next to Configuration: select "All configurations"

// next to Platform: select Win32 for 32-bit builds or x64 for 64-bit builds

// under Configuration Properties->
// C/C++->General->Additional Include Directories -
// add the GWB installation "src" folder
// "c:\program files\gwb\src"
```

## GWB Reference Manual

---

```
// Linker->Input->Additional Dependencies -  
// add the GWBplugin.lib library  
// "c:\program files\gwb\gwbplugin.lib"  
  
// Debugging->Environment -  
// add the GWB install folder to the path  
// path=%path%;c:\program files\gwb  
  
// build using Build->Build Solution  
  
// run using Debug->Start Without Debugging
```

Note: If you need to debug your program you must attach a debugger after the "initialize" call to GWBplugin. A good way to do this is to put in a `getchar()` call that will pause the program until you can attach the debugger.



## 7.2 Fortran

GWB provides a Fortran interface, “GWBplugin.f90”, and the GWBplugin.dll export library “GWBplugin.lib” to link to. GWBplugin.f90 is installed in the “src” subdirectory of the GWB installation directory while the GWBplugin.lib is installed in the main GWB installation directory. The GWB plug-in has been tested on Fortran compilers from Intel and GCC. The version of the compiler you are using must be the same as the version of GWB installed (32-bit vs. 64-bit).

This is the Fortran wrapper interface provided in GWBplugin.f90:

```
// GWBplugin.f90

MODULE GWBpluginModule

  INTEGER, PARAMETER :: ANULL = -999999 ! marker for undefined value
  INTEGER, PARAMETER :: GWB_MAX_RESPONSE = 32

  TYPE GWBplugin

  FUNCTION initialize(plugin, app_name, file_name, cmds) RESULT(retval)
    TYPE(GWBplugin), INTENT(out), TARGET :: plugin
    CHARACTER(LEN = *), INTENT(in) :: app_name
    CHARACTER(LEN = *), INTENT(in), OPTIONAL :: file_name, cmds
    INTEGER(C_INT) :: retval

  FUNCTION exec_cmd(plugin, uline) RESULT(retval)
    TYPE(GWBplugin), INTENT(in), TARGET :: plugin
    CHARACTER(LEN = *), INTENT(in) :: uline
    INTEGER(C_INT) :: retval

  FUNCTION results(plugin, f_data, f_value, f_units, ix, jy) RESULT(retval)
    TYPE(GWBplugin), INTENT(in), TARGET :: plugin
    ! f_data :
    CHARACTER(LEN = GWB_MAX_RESPONSE), INTENT(out),
      OPTIONAL :: f_data(:)
      or
    REAL(8), INTENT(out), OPTIONAL :: f_data(:)
      or
    INTEGER, INTENT(out), OPTIONAL :: f_data(:)
    CHARACTER(LEN = *), INTENT(in) :: f_value
    CHARACTER(LEN = *), INTENT(in), OPTIONAL :: f_units
    INTEGER, INTENT(in), OPTIONAL :: ix, jy
    INTEGER(C_INT) :: retval
```

```
SUBROUTINE destroy(plugin)
  TYPE(GWBplugin), INTENT(in), TARGET :: plugin
```

### 7.2.1 Initializing the GWB application

Within your code, first create a GWBplugin object.

```
INCLUDE "GWBplugin.f90"

USE GWBpluginModule
TYPE(GWBplugin) :: myPlugin
```

Next, use the “initialize” function to start the GWB application of interest by passing the application name, an optional output file name, and any command-line type arguments.

```
FUNCTION initialize(plugin, app_name, file_name, cmds) RESULT(retval)
  TYPE(GWBplugin), INTENT(out), TARGET :: plugin
  CHARACTER(LEN = *) , INTENT(in) :: app_name
  CHARACTER(LEN = *) , INTENT(in), OPTIONAL :: file_name, cmds
  INTEGER(C_INT) :: retval
```

Parameters:

**plugin**

An instance of type GWBplugin.

**app\_name**

String containing the GWB application name - “rxn”, “spece8”, “react”, “x1t”, or “x2t”.

**file\_name** (optional)

String containing the name of the file you want the GWB output written to. This can be an empty string if you do not want the output to be written to a file.

**cmds** (optional)

String containing command-line options you could normally pass to the application when running it from the command-line. This can be an empty string.

Command-line options:

-cd	Change the working directory to the directory containing the input script specified with the -i option.
-nocd	Do not change the working directory.
-i <input_script>	Read initial input commands from the specified file.
-gtd <gtdata_dir>	Set directory to search for thermodynamic datasets.
-cond <cond_data>	Set the dataset for calculating electrical conductivity.
-d <thermo_data>	Set the thermodynamic dataset.
-s <surf_data>	Set a dataset of surface sorption reactions.
-iso <isotope_data>	Set a dataset of isotope fractionation factors.

**Return value**

Non-zero on success and zero on failure.

### Examples

Some examples of how to start the GWB plug-in in various ways:

```

INTEGER :: success

! plug-in SpecE8 with no output written and no command-line options
success = initialize(myPlugin, "spece8")

! plug-in React with output written to output.txt and no command-line options
success = initialize(myPlugin, "react", "output.txt")

! plug-in X1t with no output written, no working directory change,
! and input read from pb_contam.x1t
success = initialize(myPlugin, "x1t", "", &
                    '-nocd -i \'c:/program files/gwb/script/pb_contam.x1t\'')
    
```

Function "destroy" is used at the end of the program to free up the underlying memory associated with the GWBplugin object.

```

CALL destroy(myPlugin)
    
```

## 7.2.2 Configuring and executing calculations

Use the "exec\_cmd" function to transmit commands to the GWB plug-in. Each application has a chapter in the **GWB Command Reference** that is a comprehensive guide to the commands available. Use these commands to configure the application and then send a "go" command to trigger the calculations.

```

FUNCTION exec_cmd(plugin, uline) RESULT(retval)
    TYPE(GWBplugin), INTENT(in), TARGET :: plugin
    CHARACTER(LEN = *), INTENT(in) :: uline ! command string to be sent
                                           ! to the GWB application
    INTEGER(C_INT) :: retval
    
```

### Return value

Non-zero on success and zero on failure.

### Examples

```

err = exec_cmd(myPlugin, "3 mmol H+")
err = exec_cmd(myPlugin, "2 mmol Ca++")
err = exec_cmd(myPlugin, "5 mmolar Cl-")
err = exec_cmd(myPlugin, "go")           ! trigger the calculation
    
```

### 7.2.3 Retrieving the results

Transfer calculation results from the GWB application to your program with the “results” function. The keywords, arguments, default units, and return types are the same as those listed in the table in the [Report Command](#) chapter of this reference manual.

To use the “results” function, you provide the address of a data block to fill, along with the report command and keywords, optional desired units, and the node location of choice (X1t and X2t only).

```

FUNCTION results(plugin, f_data, f_value, f_units, ix, jy) RESULT(retval)
  TYPE(GWBplugin), INTENT(in), TARGET :: plugin
  ! f_data :
    CHARACTER(LEN = GWB_MAX_RESPONSE), INTENT(out),
      OPTIONAL :: f_data(:)
    !or
    REAL(8), INTENT(out), OPTIONAL :: f_data(:)
    !or
    INTEGER, INTENT(out), OPTIONAL :: f_data(:)
  CHARACTER(LEN = *), INTENT(in) :: f_value
  CHARACTER(LEN = *), INTENT(in), OPTIONAL :: f_units
  INTEGER, INTENT(in), OPTIONAL :: ix, jy
  INTEGER(C_INT) :: retval

```

Parameters:

**plugin**

An instance of type GWBplugin.

**f\_data**

Address of data block to fill, omit it to find the size of data block needed.

**f\_value**

String containing the report command keyword and arguments.

**f\_units** (optional)

String containing the units you want the results returned in. This can be an empty string if you want the results returned in the default units.

**ix** (optional)

X node position. This is only used when running X1t and X2t, otherwise it is ignored.

**jy** (optional)

Y node position. This is only used when running X2t, otherwise it is ignored.

**Return value**

The number of values written (or to be written) to the data block.

**Remarks**

To determine the size of data block you will need, first call this function with the data parameter omitted and with the rest of the parameters filled. If you know that the report command you are using only returns a single value, you can simply pass an array of size 1 of the correct data type. See the [Report Command](#) chapter for details on data types and available keywords.

**Examples**

```

! get aqueous species names
INTEGER :: nspec
nspec = results(myPlugin, "species");
CHARACTER(LEN=GWB_MAX_RESPONSE), ALLOCATABLE :: Species(:)
ALLOCATE(Species(nspec))
results(myPlugin, Species, "species")

! get aqueous species concentrations in mg/kg
REAL(8), ALLOCATABLE :: Conc(:)
ALLOCATE(Conc(nspec))
results(myPlugin, Conc, "concentration aqueous", "mg/kg")

! get pH at node 3,5
REAL(8) :: pH(1)
results(myPlugin, pH, "pH", "", 3, 5)
    
```

A named constant with the maximum size of strings output by the “results” command is declared in the module:

```

INTEGER, PARAMETER :: GWB_MAX_RESPONSE = 32
    
```

If the command fails for any reason, for example if the requested data doesn't exist or the specified unit conversion failed, the data block will be filled with ANULL (-999999).

### 7.2.4 Fortran code examples using the plug-in feature

The following are Fortran code examples that do the same things as the C++ code examples above. This is a code example using the GWB plug-in within your program with no output being written to a file:

```

INCLUDE "GWBplugin.f90"

PROGRAM FortranPlugin_Example1
  USE GWBpluginModule
  IMPLICIT NONE

  INTEGER :: nspec, err, i
  TYPE(GWBplugin) :: myGWBrun
  CHARACTER (LEN=GWB_MAX_RESPONSE), ALLOCATABLE :: Name(:)
  REAL(8), ALLOCATABLE :: Spec(:)
  REAL(8) :: pH(1), Cl(1)
  CHARACTER(LEN=255), dimension(3), PARAMETER :: &
    cmds = [character(len=255) :: &
      "pH = 8", "molality Cl- = .05", "go"]

  ! create and start the GWB program
    
```

```

IF (initialize(myGwbrun, "spece8", "", "-nocd") /= 0) THEN

    ! started successfully

    WRITE(*,*) "Beginning run."

    ! configure SpecE8 and trigger calculation
    DO i = 1, 3
        err = exec_cmd(myGwbrun, cmds(i))
    END DO

    WRITE(*,*) "Finished run."
    WRITE(*,*)

    ! retrieve results
    err = results(myGwbrun, pH, "pH")
    err = results(myGwbrun, Cl, "concentration Cl-")
    WRITE(*,!("concentration of Cl- in molal is ", G10.3)) Cl
    err = results(myGwbrun, Cl, "concentration Cl-", "mg/kg")
    WRITE(*,!("concentration of Cl- in mg/kg is ", G10.3)) Cl
    WRITE(*,*)

    ! get size of data
    nspec = results(myGwbrun, "species");
    WRITE(*,!(There are", I2, " aqueous species")) nspec
    WRITE(*,*)

    ! create data blocks
    ALLOCATE(Name(nspec))
    ALLOCATE(Spec(nspec))

    ! send data pointer with keyword and arguments
    err = results(myGwbrun, Name, "species")
    err = results(myGwbrun, Spec, "concentration aqueous", "mg/kg")

    DO i = 1, nspec
        WRITE(*,!(A4, " = ", G10.3, " mg/kg")) Name(i), Spec(i)
    END DO
    WRITE(*,*)

    ! use the data in your program

    DEALLOCATE(Name)
    DEALLOCATE(Spec)
ELSE
    ! handle failure to start within your program
END IF

```

```
CALL destroy(myGwbrun)

END PROGRAM FortranPlugin_Example1
```

Code example with the GWB application's normal output and results written to the console or to text files:

```
INCLUDE "GWBplugin.f90"

PROGRAM FortranPlugin_Example2
  USE GWBpluginModule
  IMPLICIT NONE

  INTEGER :: nspec, err, i
  TYPE (GWBplugin) :: myGWBrun
  CHARACTER (LEN=GWB_MAX_RESPONSE), ALLOCATABLE :: Name(:)
  REAL(8), ALLOCATABLE :: Spec(:)
  REAL(8) :: pH(1), Cl(1)
  CHARACTER(LEN=255), dimension(3), PARAMETER :: &
    cmds = [character(len=255) :: &
      "pH = 8", "molality Cl- = .05", "go"]
  CHARACTER :: a

  WRITE(*,*) "Starting program SpecE8"

  IF (initialize(myGwbrun, "spece8" , "test_output.txt",&
    '-nocd -i "C:/Program Files/Gwb/Script/Freshwater.sp8"&
    &s "C:/Program Files/Gwb/Gtdata/FeOH.sdat") /= 0) THEN

    err = exec_cmd(myGwbrun, "show surfaces")

    WRITE(*,*) "Executing Test"

    OPEN(10, file = 'test_results.txt')
    WRITE(*,*) "Writing results to test_results.txt"

    DO i = 1, 3
      err = exec_cmd(myGwbrun, cmds(i))
    END DO

    err = results(myGwbrun, pH, "pH")
    WRITE(10, '( "pH = ", F4.1)') pH

    err = results(myGwbrun, Cl, "concentration Cl-")
    WRITE(10, '( "Cl = ", ES12.5, " molal")') Cl

    err = results(myGwbrun, Cl, "concentration Cl-", "mg/kg")
```

```
IF (err /= ANULL) THEN
  WRITE(10, '( "Cl = ", ES12.5, " mg/kg")') Cl
ELSE
  WRITE(10,*) "unit conversion failed - CL = ANULL"
END IF

WRITE(10,*)

nspec = results(myGwbrun, "species");

ALLOCATE(Name(nspec))
ALLOCATE(Spec(nspec))

IF (results(myGwbrun, Name, "species") /= 0) THEN
  IF (results(myGwbrun, Spec, "concentration aqueous") /= 0) THEN
    DO i = 1, nspec
      WRITE(10, '(A32, ES12.5, " molal")') Name(i), Spec(i)
    END DO
  END IF

  WRITE(10,*)
  WRITE(10,*)

  IF (results(myGwbrun, Spec, "concentration aqueous", "mg/kg") /= 0) THEN
    DO i = 1, nspec
      WRITE(10, '(A32, ES12.5, " mg/kg")') Name(i), Spec(i)
    END DO
  END IF
END IF

CLOSE(10)

DEALLOCATE(Name)
DEALLOCATE(Spec)

END IF

CALL destroy(myGwbrun)

WRITE(*,*) "enter any letter to exit> "
READ(*,*) a

END PROGRAM FortranPlugin_Example2
```



## 7.2.5 Fortran compiling

The GWB plug-in has been tested on Fortran compilers from Intel and GCC. The version of the compiler you are using must be the same as the version of GWB installed (32-bit vs. 64-bit).

To compile the GWBplugin Example1 on the command line with Intel's Fortran compiler, follow these steps:

```
! open the Intel Command Prompt

! create a working folder and change to that folder
mkdir "%homepath%\GWBplugin"
cd "%homepath%\GWBplugin"

! copy the "src" folder from GWB installation (default install path shown)
copy /Y "C:\Program Files\GWB\src"

! copy the "GWBplugin.lib" file from GWB installation (default install path shown)
copy /Y "C:\Program Files\GWB\gwbplugin.lib"

! add the GWB installation folder to your path
set path=C:\Program Files\GWB;%path%

! compile the example file and tell the compiler to use the GWBplugin library
ifort GWBplugin_Fortran_Example1.f90 GWBplugin.lib

! run the example
GWBplugin_Fortran_Example1.exe
```

To compile the GWBplugin Example1 on the command line with MinGW, MSYS, and gfortran, follow these steps:

```
! launch the MinGW Shell

! create a working folder and change to that folder
mkdir -p ~/GWBplugin
cd ~/GWBplugin

! copy the "src" folder from GWB installation (default install path shown)
cp /c/program\ files/gwb/src/* .

! copy the "GWBplugin.lib" file from GWB installation (default install path shown)
cp /c/program\ files/gwb/gwbplugin.lib .

! add the GWB installation folder to your path
PATH=/c/program\ files/gwb:$PATH
```

```
! compile the example file and tell the compiler to use the GWBplugin library
gfortran GWBplugin_Fortran_Example1.f90 GWBplugin.lib -o
      GWBplugin_Fortran_Example1.exe           ! all on one line

! run the example
./GWBplugin_Fortran_Example1.exe
```

To compile the GWBplugin Example1 in Microsoft Visual Studio with Intel's Fortran compiler, follow these steps:

```
! open Visual Studio

! create a new project (Ctrl+Shift+N)

! select the "Intel(R) Visual Fortran" project type,
! select "Console Application", and use the "Empty Project" template

! select Project->Add Existing Item... (Shift+Alt+A)
! browse to the "src" subfolder of the GWB installation,
! select the "GWBplugin_Fortran_example1.f90" file and click "Add"

! open Project->Properties->Configuration Properties

! next to Configuration: select "All configurations"

! next to Platform: select Win32 for 32-bit builds or x64 for 64-bit builds

! under Configuration Properties->

! Fortran->General->Additional Include Directories -
! add the GWB installation "src" folder
! "c:\program files\gwb\src"

! Linker->Input->Additional Dependencies -
! add the GWBplugin.lib library
! "c:\program files\gwb\gwbplugin.lib"

! Debugging->Environment -
! add the GWB install folder to the path
! path=%path%;c:\program files\gwb

! build using Build->Build Solution

! run using Debug->Start Without Debugging
```

Note: If you need to debug your program you must attach a debugger after the "initialize" call to GWBplugin, otherwise your program will encounter a run-time error.

## 7.3 Python

GWB provides a GWBplugin wrapper class contained in the Python script file “GWBplugin.py” which handles dealing with the C data type conversion and calling the DLL.

Since Python is a dynamically typed language, there are some minor differences with its “results” functions compared to statically typed languages.

To include GWBplugin.py in your Python script you just need to import the class.

```
# import GWBplugin class
from GWBplugin import *
```

If you didn’t select “Set user PATH and PYTHONPATH environment variables” when installing GWB you will first need to append the “src” folder of the GWB installation to sys.path in Python, and then import the class.

```
import os,sys

## append full path to GWBplugin.py ...
sys.path.append("c:/program files/gwb/src")

## or relative path ...
sys.path.append(os.path.abspath('.'))

# import GWBplugin class
from GWBplugin import *
```

This is the Python wrapper class provided in GWBplugin.py in the “src” directory of the GWB installation folder:

```
# GWBplugin.py

ANULL = -999999

class GWBplugin:
    Name = "GWBplugin"
    def __init__(self):
    def initialize (self, app_name, file_name = None, cmds = None):
    def exec_cmd (self, uline):
    def results (self, value, units = None, ix = 0, jy = 0):
    def destroy (self):
```

### 7.3.1 Initializing the GWB application

Within your code, first create a GWBplugin object.

```
myPlugin = GWBplugin()
```

Next, use the “initialize” function to start the GWB application of interest by passing the application name, an optional output file name, and any command-line type arguments. The “initialize” function must be called before calling any of the other functions.

```
def initialize (
    self,
    app_name,
    file_name = None,
    cmds = None):
```

Parameters:

**app\_name**

String containing the GWB application name - “rxn”, “spece8”, “react”, “x1t”, or “x2t”.

**file\_name** (optional)

String containing the name of the file you want the GWB output written to. This can be None or an empty string if you do not want the output to be written to a file.

**cmds** (optional)

String containing command-line options you could normally pass to the application when running it from the command-line. This can be None or an empty string for defaults.

Command-line options:

-cd	Change the working directory to the directory containing the input script specified with the <code>-i</code> option.
-nocd	Do not change the working directory.
-i <input_script>	Read initial input commands from the specified file.
-gtd <gtdata_dir>	Set directory to search for thermodynamic datasets.
-cond <cond_data>	Set the dataset for calculating electrical conductivity.
-d <thermo_data>	Set the thermodynamic dataset.
-s <surf_data>	Set a dataset of surface sorption reactions.
-iso <isotope_data>	Set a dataset of isotope fractionation factors.

**Return value**

Non-zero on success and zero on failure.

**Examples**

```
# plug-in SpecE8 with no output written and no command-line options
success = myPlugin.initialize("spece8")

# plug-in React with output written to output.txt and no command-line options
success = myPlugin.initialize("react", "output.txt")
```

```
# plug-in X1t with no output written, no working directory change,  
# and input read from pb_contam.x1t  
success = myPlugin.initialize("x1t", "", "-nocd  
-i \"c:/program files/gwb/script/pb_contam.x1t\\\"")
```

Function “destroy” can be used at the end of the program to free up the underlying memory associated with the GWBplugin object.

```
myPlugin.destroy()
```

### 7.3.2 Configuring and executing calculations

Use the “exec\_cmd” function to transmit commands to the GWB plug-in. Each application has a chapter in the **GWB Command Reference** that is a comprehensive guide to the commands available. Use these commands to configure the application and then send a “go” command to trigger the calculations.

```
def exec_cmd (  
    self,  
    uline # command string to be sent to the GWB application  
):
```

#### Return value

Non-zero on success and zero on failure.

#### Examples

```
myPlugin.exec_cmd("3 mmol H+")  
myPlugin.exec_cmd("2 mmol Ca++")  
myPlugin.exec_cmd("5 mmolar Cl-")  
myPlugin.exec_cmd("go") # trigger the calculation
```

### 7.3.3 Retrieving the results

Transfer calculation results from the GWB application to your program with the “results” functions. The keywords, arguments, default units, and return types are the same as those listed in the table in the **Report Command** chapter of this reference manual. Use the “results” functions by providing the report command and keywords, optional desired units, and the node location of choice (X1t and X2t only).

```
# results function  
def results (self, value, units = None, ix = 0, jy = 0):
```

Parameters:

#### value

String containing the report command keyword and arguments.

**units** (optional)

String containing the units you want the results returned in. This can be None or an empty string if you want the results returned in the default units.

**ix** (optional)

X node position. This is only used when running X1t and X2t, otherwise it is ignored.

**iy** (optional)

Y node position. This is only used when running X2t, otherwise it is ignored.

**Return value**

Array containing the requested results.

**Remarks**

The data is returned as an array, even when requesting a single value.

If the command fails for any reason, for example if the requested data doesn't exist or the specified unit conversion failed, the data will be filled with ANULL (-999999).

**Examples**

```
# get aqueous species names
Species = myPlugin.results("species")

# get aqueous species concentrations in mg/kg
Conc = myPlugin.results("concentration aqueous", "mg/kg")

# get pH at node 3,5
pH = myPlugin.results("pH", "", 3,5)[0]
```

**7.3.4 Python code examples using the plug-in feature**

Normally you would use the GWB plug-in within your program with no output being written to a file. The following is an example of this:

```
# import GWBplugin class
from GWBplugin import *

# create the plug-in object
myGWBrun = GWBplugin()

# start the GWB program
if myGWBrun.initialize("spece8", "", "-nocd"):

    print "Beginning run."

    cmds = ["pH = 8", "molality Cl- = .05", "go"]
    for cmd in cmds:
        myGWBrun.exec_cmd(cmd)

    print "Finished run.\n"

#retrieve results
pH = myGWBrun.results("pH")[0]
```

```

Cl = myGWBrun.results("concentration Cl-")[0]
print "concentration of Cl- in molal is %10.4g" % Cl
Cl = myGWBrun.results("concentration Cl-", "mg/kg")[0]
print "concentration of Cl- in mg/kg is %10.4g" % Cl

species = myGWBrun.results("species")
conc = myGWBrun.results("concentration aqueous", "mg/kg")

print "\nThere are" , len(species) , "aqueous species.\n"
for i in range(len(species)):
    print "%-4s = %10.4g mg/kg" % (species[i], conc[i])

myGWBrun.destroy()

```

To familiarize yourself with the plug-in feature, you might want the GWB program's normal output and results to be written to the console and to text files. The following code shows examples of this:

```

# import GWBplugin class
from GWBplugin import *

print "Starting program SpecE8"

myGWBrun = GWBplugin()

if myGWBrun.initialize("spece8",
    "test_output.txt",
    "-nocd \
-i \"c:/program files/gwb/script/freshwater.sp8\" \
-s \"c:/program files/gwb/gtdata/feoh.sdat\""):

    print "writing output to test_output.txt"
    myGWBrun.exec_cmd("show surfaces")
    print "Executing test"
    fp = open("test_results.txt", "w")
    if fp.closed:
        stderr.write("can't open test_results.txt")
    else:
        print "writing results to test_results.txt"
        cmds = ["pH = 8", "molality Cl- = .05", "go"]
        for cmd in cmds:
            myGWBrun.exec_cmd(cmd)

    pH = myGWBrun.results("pH")[0]
    fp.write("pH = %4.1f\n" % pH)

    Cl = myGWBrun.results("concentration Cl-")[0]

```



```
fp.write("Cl = %12.5e molal\n" % Cl)

Cl = myGWBrun.results("concentration Cl-", "mg/kg")[0]
if Cl != ANULL:
    fp.write("Cl = %12.5e mg/kg\n\n" % Cl)
else:
    fp.write("unit conversion failed - Cl = ANULL\n\n")

Name = myGWBrun.results("species")
Spec = myGWBrun.results("concentration aqueous")
for i in range(len(Name)):
    fp.write("%-32s %12.5e molal\n" % (Name[i], Spec[i]))
Spec = myGWBrun.results("concentration aqueous", "mg/kg")
fp.write("\n\n")
for i in range(len(Name)):
    fp.write("%-32s %12.5e mg/kg\n" % (Name[i], Spec[i]))
fp.close()

myGWBrun.destroy()
input("press return to exit> ")
```

### 7.3.5 Python command line

The GWB plug-in has been tested with Python for Windows version 3.10.2. The version of Python you are using must be the same as the version of GWB installed (32-bit vs. 64-bit).

To run the GWBplugin Example1 on the command line with Python, follow these steps:

```
# open the command prompt
cmd.exe

# run the example with Python
python "/Program Files/GWB/src/GWBplugin_Python_Example1.py"
```

## 7.4 Other languages

Any language that can load DLLs, call C functions from them, and handle some basic C data types should be able to use the GWB plug-in feature. You must have your GWB installation folder added to the PATH environment variable so that all of the required DLLs can be found. The C data types that need to be handled are void\*, char\*, int\*, double\*, and int. If the language you want to use is similar to one that a wrapper is provided for, a good place to start is to look at how that wrapper is implemented.

To create a wrapper class, interface, or whatever makes sense for your target language, follow these steps:

- **Load the GWBplugin DLL.** Generally this will be done during run-time with a call to LoadLibrary or whatever the equivalent is in the language. Some languages, mostly compiled and linked ones, can instead link to the export library GWBplugin.lib.
- **Tell your program about the functions you will call from the DLL.** This is usually done by giving prototypes in some way or by directly including GWBplugin.h. The DLL functions and their prototypes are listed in the next section.
- **Encapsulate.** Create functions in your wrapper that call the corresponding DLL function and handle data type conversions. The wrapper, if possible, should also have a void\* member variable that can be passed by address to the DLL functions. This void\* member variable keeps track of a particular GWBplugin instance.

### 7.4.1 GWBplugin.dll function prototypes

Following is the list of the definitions and functions exported from GWBplugin.dll that your wrapper will need to use. Note that function parameters labeled as (optional) are in fact required when you call the C function. It is suggested, however, that you make these arguments optional for your own wrapper if possible and use the provided suggested defaults.

```
// GWBplugin.h

#define ANULL -999999.0          // marker for an undefined value

extern "C" __declspec(dllexport)
    int c_initialize(void* plugin, const char* app_name,
                    const char* file_name, const char* cmds);

extern "C" __declspec(dllexport)
    int c_exec_cmd(void* plugin, char* uline);

extern "C" __declspec(dllexport)
    int c_results(void* plugin, void* data, const char* value,
                 const char* units, int ix, int jy);
```

```
extern "C" __declspec(dllexport)
int c_results_c(void* plugin, void* data, const char* value,
               const char* units, int ix, int jy, int* slen);

extern "C" __declspec(dllexport)
int c_destroy(void* plugin);
```

## 7.4.2 Initializing the GWB application

Within your code, first create a void\* equivalent variable or something that can hold a pointer data type... i.e. that is 32-bits long (for a 32-bit application) or 64-bits long (for a 64-bit application). This will be a member variable of your class if possible.

Next, use the “c\_initialize” function to start the GWB application of interest by passing the address of the void\* variable, the application name, an optional output file name, and any command-line type arguments. The “c\_initialize” function must be called before calling any of the other functions.

```
int c_initialize (
    void* plugin,
    const char* app_name,
    const char* file_name,
    const char* cmds
);
```

Parameters:

### **plugin**

A dereferenceable pointer that points to a pointer which can be assigned a value. It keeps track of a particular plugged-in GWB application.

### **app\_name**

String containing the GWB application name - “rxn”, “spece8”, “react”, “x1t”, or “x2t”.

### **file\_name** (optional) (default: NULL or empty string)

String containing the name of the file you want the GWB output written to. This can be NULL or an empty string if you do not want the output to be written to a file.

### **cmds** (optional) (default: NULL or empty string)

String containing command-line options you could normally pass to the application when running it from the command-line. This can be NULL or an empty string.

Command-line options:

<code>-cd</code>	Change the working directory to the directory containing the input script specified with the <code>-i</code> option.
<code>-nocd</code>	Do not change the working directory.
<code>-i &lt;input_script&gt;</code>	Read initial input commands from the specified file.
<code>-gtd &lt;gtdata_dir&gt;</code>	Set directory to search for thermodynamic datasets.
<code>-cond &lt;cond_data&gt;</code>	Set the dataset for calculating electrical conductivity.
<code>-d &lt;thermo_data&gt;</code>	Set the thermodynamic dataset.
<code>-s &lt;surf_data&gt;</code>	Set a dataset of surface sorption reactions.
<code>-iso &lt;isotope_data&gt;</code>	Set a dataset of isotope fractionation factors.

### Return value

Non-zero on success and zero on failure.

### Examples

Some examples of how to start the GWB plug-in in various ways:

```
void* myPlugin = NULL;

// plug-in SpecE8 with no output written and no command-line options
int success = c_initialize(&myPlugin, "spece8");

// plug-in React with output written to output.txt and no command-line options
int success = c_initialize(&myPlugin, "react", "output.txt");

// plug-in X1t with no output written, no working directory change,
// and input read from pb_contam.x1t
int success = c_initialize(&myPlugin, "x1t", NULL, "-nocd
-i \"c:/program files/gwb/script/pb_contam.x1t\"");
```

Function "c\_destroy" can be used at the end of the program to free up the underlying memory associated with the plugged-in GWB application.

```
c_destroy(&myPlugin);
```

### 7.4.3 Configuring and executing calculations

Use the "c\_exec\_cmd" function to transmit commands to the GWB plug-in. Each application has a chapter in the **GWB Command Reference** that is a comprehensive guide to the commands available. Use these commands to configure the application and then send a "go" command to trigger the calculations.

```
int c_exec_cmd(
    void* plugin,
    char* uline
);
```

Parameters:

**plugin**

A dereferenceable pointer that has already been used with `c_initialize`. It keeps track of a particular plugged-in GWB application.

**uline**

String containing the command to be sent to the GWB application.

**Return value**

Non-zero on success and zero on failure.

**Examples**

```
c_exec_cmd(&myPlugin, "3 mmol H+");
c_exec_cmd(&myPlugin, "2 mmol Ca+");
c_exec_cmd(&myPlugin, "5 mmolar Cl-");
c_exec_cmd(&myPlugin, "go");
```

### 7.4.4 Retrieving the results

Transfer calculation results from the GWB application to your program with the “`c_results`” function. The keywords, arguments, default units, and return types are the same as those listed in the table in the [Report Command](#) chapter of this reference manual. Use the “`c_results`” function by providing the plugin parameter, the address of a data block to fill, the report command and keywords, optional desired units, and the node location of choice (X1t and X2t only).

```
int c_results(
    void* plugin,
    void* data,
    const char* value,
    const char* units,
    int ix,
    int jy
);
```

Parameters:

**plugin**

A dereferenceable pointer that has already been used with `c_initialize`. It keeps track of a particular plugged-in GWB application.

**data**

Address of data block to fill. This can be NULL to determine data block size.

**value**

String containing the report command keyword and arguments.

**units** (optional) (default: NULL or empty string)

String containing the units you want the results returned in. This can be NULL or an empty string if you want the results returned in the default units.

**ix** (optional) (default: 0)

X node position. This is only used when running X1t and X2t, otherwise it is ignored.

**iy** (optional) (default: 0)

Y node position. This is only used when running X2t, otherwise it is ignored.

#### Return value

The number of values written (or to be written) to the data block.

#### Remarks

To determine the size of data block you will need, first call this function with the data parameter as NULL and with the rest of the parameters filled. If you know that the report command you are using only returns a single value, you can simply pass a pointer to the correct data type. See the [Report Command](#) chapter for details on data types and available keywords.

If the command fails for any reason, for example if the requested data doesn't exist or the specified unit conversion failed, the data will be filled with ANULL (-999999.0). For this reason, you should "#define ANULL -999999.0" (or language equivalent) in your wrapper.

For languages that are dynamically typed (e.g. Python and Perl), you will either need to create multiple wrapper "results" functions (one for each possible data type: int, double, char\*) or pass the expected type as an extra parameter. It is often best to omit the data parameter in the wrapper function. You then can call "c\_results" with a NULL value for data to get the size, allocate C compatible memory, call "c\_results" with the data parameter, convert data, and then return an array of the results. See [GWBplugin.pm](#) or [GWBplugin.py](#) for examples of this.

#### Examples

```
// get aqueous species names
int ndata = c_results(&myPlugin, NULL, "species");
char** Species = (char**) malloc(sizeof(char*) * ndata);
c_results(&myPlugin, Species, "species");

// get aqueous species concentrations in mg/kg
double* Conc = (double*) malloc(sizeof(double) * ndata);
c_results(&myPlugin, Conc, "concentration aqueous", "mg/kg");

// get pH at node 3,5
double pH = ANULL;
c_results(&myPlugin, &pH, "pH", NULL, 3, 5);
```

If you are retrieving string values and you need to know the string lengths for conversion purposes, you will need to use the "c\_results\_c" function. It is equivalent to the "c\_results" function, but it also takes an extra parameter which will store the length of the strings.

```
int c_results_c(
    void* plugin,
    void* data,
    const char* value,
    const char* units,
```

```

int ix,
int jy,
int* slen // address of data block to fill with retrieved string lengths
);

```

### Examples

```

// get aqueous species names
int ndata = c_results(&myPlugin, NULL, "species");
char** Species = (char**) malloc(sizeof(char*) * ndata);
int* Lengths = (int*) malloc(sizeof(int) * ndata);
c_results_c(&myPlugin, Species, "species", NULL, 0, 0, Lengths);

```





## Units Recognized

---

The following is a complete table of the unit names recognized by the GWB. The qualifier “free” specifies that the constraint applies to the free rather than to the bulk entry. Use the “log” qualifier to set the variable on a logarithmic scale. Examples:

Cl- 4.1 mg/kg  
 Cl- 4.1 free mg/kg  
 Cl- 0.612784 log free mg/kg

Dimension	Units			
<b>Mass and Concentration</b>	mol	mmol	umol	nmol
	molal	mmolal	umolal	nmolal
	mol/kg	mmol/kg	umol/kg	nmol/kg
	mol/l	mmol/l	umol/l	nmol/l
	kg	g	mg	ug
	ng			
	g/kg	mg/kg	ug/kg	ng/kg
	wt fraction	wt%		
	g/l	mg/l	ug/l	ng/l
	eq	meq	ueq	neq
	eq/kg	meq/kg	ueq/kg	neq/kg
	eq/l	meq/l	ueq/l	neq/l
	cm <sup>3</sup>	m <sup>3</sup>	km <sup>3</sup>	l
	mol/cm <sup>3</sup>	mmol/cm <sup>3</sup>	umol/cm <sup>3</sup>	nmol/cm <sup>3</sup>
	kg/cm <sup>3</sup>	g/cm <sup>3</sup>	mg/cm <sup>3</sup>	ug/cm <sup>3</sup>
	ng/cm <sup>3</sup>			
	mol/m <sup>3</sup>	mmol/m <sup>3</sup>	umol/m <sup>3</sup>	nmol/m <sup>3</sup>
	kg/m <sup>3</sup>	g/m <sup>3</sup>	mg/m <sup>3</sup>	ug/m <sup>3</sup>
	ng/m <sup>3</sup>			
	vol. fract.	volume%		

## GWB Reference Manual

<b>Dimension</b>	<b>Units</b>			
<b>Activity</b>	activity	ratio		
<b>Fugacity</b>	fugacity			
<b>Electrical Potential (Eh)</b>	V	mV	pe	
<b>pH</b>	pH			
<b>Percentage</b>	%			
<b>Time</b>	s mon	min yr	hr m.y.	day
<b>Distance</b>	mm in	cm ft	m mi	km
<b>Reaction Rate</b>	mol/s kg/s ng/s cm <sup>3</sup> /s ft <sup>3</sup> /s mol/min kg/min ng/min cm <sup>3</sup> /min ft <sup>3</sup> /min mol/hr kg/hr ng/hr cm <sup>3</sup> /hr ft <sup>3</sup> /hr mol/day kg/day ng/day cm <sup>3</sup> /day ft <sup>3</sup> /day mol/yr kg/yr ng/yr cm <sup>3</sup> /yr ft <sup>3</sup> /yr	mmol/s g/s  m <sup>3</sup> /s  mmol/min g/min  m <sup>3</sup> /min  mmol/hr g/hr  m <sup>3</sup> /hr  mmol/day g/day  m <sup>3</sup> /day  mmol/yr g/yr  m <sup>3</sup> /yr	umol/s mg/s  l/s  umol/min mg/min  l/min  umol/hr mg/hr  l/hr  umol/day mg/day  l/day  umol/yr mg/yr  l/yr	nmol/s ug/s  gal/s  nmol/min ug/min  gal/min  nmol/hr ug/hr  gal/hr  nmol/day ug/day  gal/day  nmol/yr ug/yr  gal/yr

**Units Recognized**

<b>Dimension</b>	<b>Units</b>			
<b>Reaction Rate</b>	mol/m.y.	mmol/m.y.	umol/m.y.	nmol/m.y.
	kg/m.y.	g/m.y.	mg/m.y.	ug/m.y.
	ng/m.y.			
	cm <sup>3</sup> /m.y.	m <sup>3</sup> /m.y.	l/m.y.	gal/m.y.
	ft <sup>3</sup> /m.y.			
	mol/cm <sup>3</sup> /s	mmol/cm <sup>3</sup> /s	umol/cm <sup>3</sup> /s	nmol/cm <sup>3</sup> /s
	kg/cm <sup>3</sup> /s	g/cm <sup>3</sup> /s	mg/cm <sup>3</sup> /s	ug/cm <sup>3</sup> /s
	ng/cm <sup>3</sup> /s			
	cm <sup>3</sup> /cm <sup>3</sup> /s	volume%/s		
	mol/cm <sup>3</sup> /min	mmol/cm <sup>3</sup> /min	umol/cm <sup>3</sup> /min	nmol/cm <sup>3</sup> /min
	kg/cm <sup>3</sup> /min	g/cm <sup>3</sup> /min	mg/cm <sup>3</sup> /min	ug/cm <sup>3</sup> /min
	ng/cm <sup>3</sup> /min			
	cm <sup>3</sup> /cm <sup>3</sup> /min	volume%/min		
	mol/cm <sup>3</sup> /hr	mmol/cm <sup>3</sup> /hr	umol/cm <sup>3</sup> /hr	nmol/cm <sup>3</sup> /hr
	kg/cm <sup>3</sup> /hr	g/cm <sup>3</sup> /hr	mg/cm <sup>3</sup> /hr	ug/cm <sup>3</sup> /hr
	ng/cm <sup>3</sup> /hr			
	cm <sup>3</sup> /cm <sup>3</sup> /hr	volume%/hr		
	mol/cm <sup>3</sup> /day	mmol/cm <sup>3</sup> /day	umol/cm <sup>3</sup> /day	nmol/cm <sup>3</sup> /day
	kg/cm <sup>3</sup> /day	g/cm <sup>3</sup> /day	mg/cm <sup>3</sup> /day	ug/cm <sup>3</sup> /day
	ng/cm <sup>3</sup> /day			
	cm <sup>3</sup> /cm <sup>3</sup> /day	volume%/day		
	mol/cm <sup>3</sup> /yr	mmol/cm <sup>3</sup> /yr	umol/cm <sup>3</sup> /yr	nmol/cm <sup>3</sup> /yr
	kg/cm <sup>3</sup> /yr	g/cm <sup>3</sup> /yr	mg/cm <sup>3</sup> /yr	ug/cm <sup>3</sup> /yr
	ng/cm <sup>3</sup> /yr			
	cm <sup>3</sup> /cm <sup>3</sup> /yr	volume%/yr		
	mol/cm <sup>3</sup> /m.y.	mmol/cm <sup>3</sup> /m.y.	umol/cm <sup>3</sup> /m.y.	nmol/cm <sup>3</sup> /m.y.
	kg/cm <sup>3</sup> /m.y.	g/cm <sup>3</sup> /m.y.	mg/cm <sup>3</sup> /m.y.	ug/cm <sup>3</sup> /m.y.
	ng/cm <sup>3</sup> /m.y.			
	cm <sup>3</sup> /cm <sup>3</sup> /m.y.	volume%/m.y.		
	mol/m <sup>3</sup> /s	mmol/m <sup>3</sup> /s	umol/m <sup>3</sup> /s	nmol/m <sup>3</sup> /s
	kg/m <sup>3</sup> /s	g/m <sup>3</sup> /s	mg/m <sup>3</sup> /s	ug/m <sup>3</sup> /s
	ng/m <sup>3</sup> /s			
m <sup>3</sup> /m <sup>3</sup> /s				
mol/m <sup>3</sup> /min	mmol/m <sup>3</sup> /min	umol/m <sup>3</sup> /min	nmol/m <sup>3</sup> /min	
kg/m <sup>3</sup> /min	g/m <sup>3</sup> /min	mg/m <sup>3</sup> /min	ug/m <sup>3</sup> /min	
ng/m <sup>3</sup> /min				
m <sup>3</sup> /m <sup>3</sup> /min				

## GWB Reference Manual

Dimension	Units			
<b>Reaction Rate</b>	mol/m <sup>3</sup> /hr	mmol/m <sup>3</sup> /hr	umol/m <sup>3</sup> /hr	nmol/m <sup>3</sup> /hr
	kg/m <sup>3</sup> /hr	g/m <sup>3</sup> /hr	mg/m <sup>3</sup> /hr	ug/m <sup>3</sup> /hr
	ng/m <sup>3</sup> /hr			
	m <sup>3</sup> /m <sup>3</sup> /hr			
	mol/m <sup>3</sup> /day	mmol/m <sup>3</sup> /day	umol/m <sup>3</sup> /day	nmol/m <sup>3</sup> /day
	kg/m <sup>3</sup> /day	g/m <sup>3</sup> /day	mg/m <sup>3</sup> /day	ug/m <sup>3</sup> /day
	ng/m <sup>3</sup> /day			
	m <sup>3</sup> /m <sup>3</sup> /day			
	mol/m <sup>3</sup> /yr	mmol/m <sup>3</sup> /yr	umol/m <sup>3</sup> /yr	nmol/m <sup>3</sup> /yr
	kg/m <sup>3</sup> /yr	g/m <sup>3</sup> /yr	mg/m <sup>3</sup> /yr	ug/m <sup>3</sup> /yr
	ng/m <sup>3</sup> /yr			
	m <sup>3</sup> /m <sup>3</sup> /yr			
	mol/m <sup>3</sup> /m.y.	mmol/m <sup>3</sup> /m.y.	umol/m <sup>3</sup> /m.y.	nmol/m <sup>3</sup> /m.y.
	kg/m <sup>3</sup> /m.y.	g/m <sup>3</sup> /m.y.	mg/m <sup>3</sup> /m.y.	ug/m <sup>3</sup> /m.y.
	ng/m <sup>3</sup> /m.y.			
	m <sup>3</sup> /m <sup>3</sup> /m.y.			
	molal/s	mmolal/s	umolal/s	nmolal/s
	mol/kg/s	mmol/kg/s	umol/kg/s	nmol/kg/s
	g/kg/s	mg/kg/s	ug/kg/s	ng/kg/s
	cm <sup>3</sup> /kg/s			
	molal/min	mmolal/min	umolal/min	nmolal/min
	mol/kg/min	mmol/kg/min	umol/kg/min	nmol/kg/min
	g/kg/min	mg/kg/min	ug/kg/min	ng/kg/min
	cm <sup>3</sup> /kg/min			
	molal/hr	mmolal/hr	umolal/hr	nmolal/hr
	mol/kg/hr	mmol/kg/hr	umol/kg/hr	nmol/kg/hr
	g/kg/hr	mg/kg/hr	ug/kg/hr	ng/kg/hr
	cm <sup>3</sup> /kg/hr			
	molal/day	mmolal/day	umolal/day	nmolal/day
	mol/kg/day	mmol/kg/day	umol/kg/day	nmol/kg/day
g/kg/day	mg/kg/day	ug/kg/day	ng/kg/day	
cm <sup>3</sup> /kg/day				
molal/yr	mmolal/yr	umolal/yr	nmolal/yr	
mol/kg/yr	mmol/kg/yr	umol/kg/yr	nmol/kg/yr	
g/kg/yr	mg/kg/yr	ug/kg/yr	ng/kg/yr	
cm <sup>3</sup> /kg/yr				
molal/m.y.	mmolal/m.y.	umolal/m.y.	nmolal/m.y.	
mol/kg/m.y.	mmol/kg/m.y.	umol/kg/m.y.	nmol/kg/m.y.	
g/kg/m.y.	mg/kg/m.y.	ug/kg/m.y.	ng/kg/m.y.	
cm <sup>3</sup> /kg/m.y.				
<b>Flow Rate</b>	cm <sup>3</sup> /s	m <sup>3</sup> /s	l/s	gal/s
	ft <sup>3</sup> /s			
	cm <sup>3</sup> /min	m <sup>3</sup> /min	l/min	gal/min
	ft <sup>3</sup> /min			

**Units Recognized**

<b>Dimension</b>	<b>Units</b>			
<b>Flow Rate</b>	cm3/hr	m3/hr	l/hr	gal/hr
	ft3/hr			
	cm3/day	m3/day	l/day	gal/day
	ft3/day			
	cm3/yr	m3/yr	l/yr	gal/yr
	ft3/yr			
	cm3/m.y.	m3/m.y.	l/m.y.	gal/m.y.
ft3/m.y.				
<b>Velocity</b>	mm/s	cm/s	m/s	km/s
	mm/hr	cm/hr	m/hr	km/hr
	mm/day	cm/day	m/day	km/day
	mm/mon	cm/mon	m/mon	km/mon
	mm/yr	cm/yr	m/yr	km/yr
	mm/m.y.	cm/m.y.	m/m.y.	km/m.y.
	in/s	ft/s	mi/s	
	in/hr	ft/hr	mi/hr	
	in/day	ft/day	mi/day	
	in/mon	ft/mon	mi/mon	
	in/yr	ft/yr	mi/yr	
	in/m.y.	ft/m.y.	mi/m.y.	
	<b>Specific Discharge</b>	cm3/cm2/s	m3/m2/s	ft3/ft2/s
cm3/cm2/hr		m3/m2/hr	ft3/ft2/hr	
cm3/cm2/day		m3/m2/day	ft3/ft2/day	
cm3/cm2/mon		m3/m2/mon	ft3/ft2/mon	
cm3/cm2/yr		m3/m2/yr	ft3/ft2/yr	
cm3/cm2/m.y.		m3/m2/m.y.	ft3/ft2/m.y.	
<b>Density</b>	kg/cm3	g/cm3	mg/cm3	ug/cm3
	ng/cm3			
	kg/m3	g/m3	mg/m3	ug/m3
	ng/m3			
<b>Titration Alkalinity</b>	eq_acid	meq_acid	ueq_acid	neq_acid
	eq_acid/kg	meq_acid/kg	ueq_acid/kg	neq_acid/kg
	eq_acid/l	meq_acid/l	ueq_acid/l	neq_acid/l
	g/kg_as_CaCO3	mg/kg_as_CaCO3	ug/kg_as_CaCO3	ng/kg_as_CaCO3
	wt%_as_CaCO3			
	g/l_as_CaCO3	mg/l_as_CaCO3	ug/l_as_CaCO3	ng/l_as_CaCO3
	mol/kg_as_CaCO3	mmol/kg_as_CaCO3	umol/kg_as_CaCO3	nmol/kg_as_CaCO3
	mol/l_as_CaCO3	mmol/l_as_CaCO3	umol/l_as_CaCO3	nmol/l_as_CaCO3

GWB Reference Manual

<b>Dimension</b>	<b>Units</b>			
<b>Titration Acidity</b>	eq_base eq_base/kg eq_base/l	meq_base meq_base/kg meq_base/l	ueq_base ueq_base/kg ueq_base/l	neq_base neq_base/kg neq_base/l
<b>Sorption Capacity</b>	mol/grock	mmol/grock	umol/grock	nmol/grock
<b>Exchange Capacity</b>	eq/grock	meq/grock	ueq/grock	neq/grock
<b>Surface Charge</b>	uC/cm2			
<b>Pore Volumes</b>	pore_volumes			
<b>Dynamic Viscosity</b>	cp	poise		
<b>Compressibility</b>	/Pa /psi	/MPa	/atm	/bar
<b>Thermal expansivity</b>	/C	/F	/K	/R
<b>Pressure</b>	Pa psi	MPa	atm	bar
<b>Permeability</b>	m2 darcy	cm2 mdarcy	um2 udarcy	
<b>Diffusion/Dispersion Coefficients</b>	cm2/s	m2/s		
<b>Distribution Coefficients (KDs)</b>	l/kg	ml/g	ml/mg	
<b>Activity Coefficients</b>	act. coef.			
<b>Electrical Conductivity</b>	uS/cm	umho/cm		
<b>Energy</b>	J	kJ	cal	kcal
<b>Energy Content</b>	J/mol	kJ/mol	cal/mol	kcal/mol
<b>Heat Capacity</b>	J/g/C	J/kg/K	cal/g/C	
<b>Thermal Conductivity</b>	W/cm/C	W/m/K	cal/cm/s/C	cal/m/s/C
<b>Internal Heat Source</b>	J/cm3/s cal/cm3/s W/cm3	J/cm3/yr cal/cm3/yr W/m3	J/m3/s cal/m3/s	J/m3/yr cal/m3/yr
<b>Thermal Transmissivity</b>	W/C cal/s/C	W/K cal/s/K	J/s/C	J/s/K
<b>Percent Removal</b>	% removal			
<b>Saturation</b>	Q/K			

**Units Recognized**

<b>Dimension</b>	<b>Units</b>			
<b>Deuterium (<sup>2</sup>H)</b>	SMOW-2H			
<b>Tritium (<sup>3</sup>H)</b>	TU			
<b>Foaming Agents</b>	g/l	mg/l	ug/l	ng/l
<b>Carbon 13</b>	PDB			
<b>Percent Modern Carbon</b>	PMC			
<b>Oxygen Demand</b>	g/l	mg/l	ug/l	ng/l
<b>Oxygen 18</b>	SMOW-18O			
<b>Sulfur 34</b>	CDT			
<b>Odor</b>	TON			
<b>Turbidity</b>	NTU			
<b>Corrosivity</b>	Cor			
<b>Colonies per Volume</b>	colonies/ml			
<b>Radioactive Emission per Volume</b>	pCi/l			
<b>Radioactive Exposure over Time</b>	mrem/yr			
<b>Temperature</b>	C	F	K	R
<b>Angle</b>	radians	degrees		
<b>Color</b>	CU			
<b>Number</b>	number			
<b>Text</b>	text			





# Graphics Output

---

Programs **Act2**, **Tact**, **Gtplot**, **P2plot**, and **Xtplot** can render plots on a variety of devices, including your computer screen and black-and-white and color printers. The programs can also save your plots in a variety of graphics formats; you can later import these images to documents, web pages, or presentations that you prepare with other software. Finally, you can copy your plots to the MS Windows clipboard and paste them directly into other applications, in a format meaningful to the application.

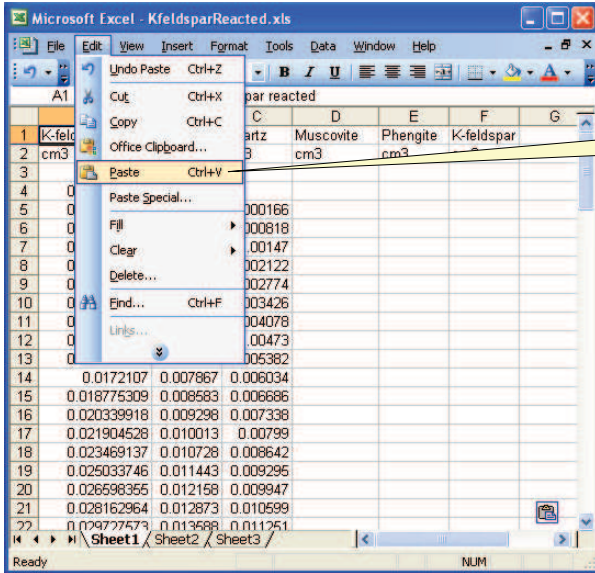
## 9.1 Clipboard

To copy the current plot to the clipboard, select **Edit** → **Copy** from the menubar on the graphics window, or touch **Ctrl+C**. You can then paste the plot directly into a variety of word processing and presentation graphics programs.

If you paste the plot into MS PowerPoint, it will appear as an EMF (an MS Enhanced Metafile) graphic object. Pasting into Adobe Illustrator places a native AI graphic.

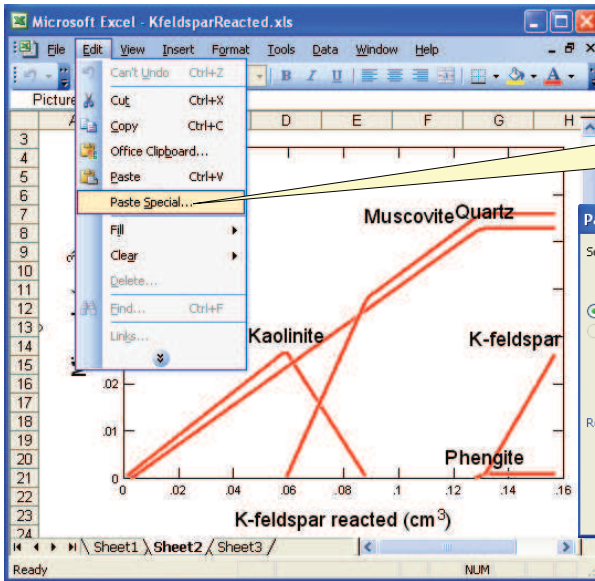
If you paste a plot from **Gtplot**, **P2plot**, or **Xtplot** into MS Excel or a text editor such as Notepad or MS Word, the numerical values of the data points that make up the lines on the plot will appear in spreadsheet format.

You can control the format in which the plot is copied to the clipboard by selecting **Edit** → **Copy As**. You can choose to copy the plot as an AI object, an EMF object, a bitmap, or the data points in the plot, as tab delimited or space delimited text. Use the tab delimited option to paste the data into a spreadsheet program like MS Excel. For examining the data in a text file created with an editor like Notepad or MS Word, the space delimited option writes a nicely aligned table.

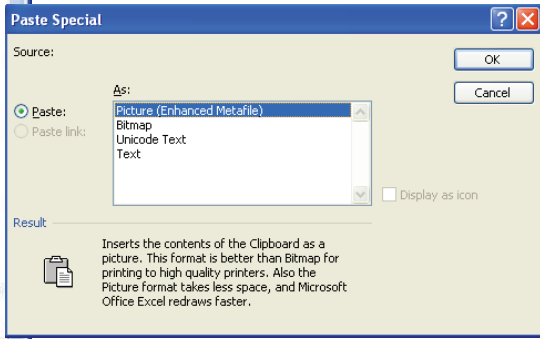


“Paste” in Excel or Word inserts numerical values

In MS Word or MS Excel, use **Paste Special...** to paste the plot as a picture instead.



“Paste Special...” in Excel or Word inserts a picture



## 9.2 Saving images

In many cases you will want to adjust label positions or change the annotation or coloring on your plot. Such changes can be made quickly using an illustration program such as PowerPoint. You can, furthermore, save images and import them into your reports or documents prepared with a word processor such as MS Word.

To save an image, select **File** → **Save Image...** from the menubar, then choose one of the file formats supported:

- PNG (.png)
- JPEG (.jpg)
- TIFF (.tif)
- Bitmap (.bmp)
- Enhanced Metafile (.emf)
- Adobe Illustrator (.ai)
- PDF File (.pdf)
- Scalable Vector Graphics (.svg)
- Compressed SVG (.svgz)
- Encapsulated PostScript (.eps)
- Color PostScript (.ps)
- Black-and-White PostScript (.ps)

**Gtplot**, **P2plot**, and **Xtplot** also support:

- Spreadsheet File (Tab delimited) (.txt)
- Text File (Space delimited) (.txt)

The programs save the plot images to files with names such as “Image\_1.eps”, “Image\_2.eps”, and so on; the suffix represents the file format (Encapsulated PostScript, in this case), as shown above.

Note that since each of these graphics formats has its own limitations, the plot once imported to another program may appear somewhat different than on your computer screen. Using your illustration program or word processor, however, you can quickly alter the diagram’s appearance to suit your needs.

When saving a PNG, JPEG, TIFF, or bitmap file, you may specify the quality of the saved image by choosing its resolution: High, Medium, Low, or Custom. Use **Custom...** to set the pixel width and height of the image, and to choose whether to preserve the aspect ratio of the plot.

Use the **Spreadsheet File (Tab delimited)** or **Text File (Space delimited)** option to save into a table the numerical coordinates of the data points on the plot. The spreadsheet table may be read directly into many popular spreadsheet programs.

Certain graphics types support font embedding. PDF files should always display and print properly, regardless of fonts installed on the system. PostScript files should also, if you have used the option to embed fonts. If you may want to edit the PostScript file, however, you should deselect the option to embed fonts, because programs such

as Adobe Illustrator may restrict your ability to edit a document using embedded fonts (due to potential copyright/licensing issues). To edit these files, be sure that all of the required fonts are installed on your computer (see **Font for data markers** below).

When importing AI graphics to Adobe Illustrator, the program may prompt you to update the legacy text before you can edit the file. In this case, choose "Update". You need to release the clipping mask before you attempt to edit individual elements of the plot. Use the "Ungroup" and "Group" functions when repositioning or modifying elements.

### 9.3 Font for data markers

We supply a special TrueType font "GWB Symbol Ext" to provide for data markers on scatter plots produced with **Act2**, **Tact**, **Gtplot**, **P2plot**, and **Xtplot**. The font is installed automatically on your computer when you install the GWB software.

The "GWB Symbol Ext" font is not subject to copyright, so you can share it freely. If, for example, you send graphics output to a colleague, you can send her the font to install on her machine. In this way, the data markers on the plots will appear correctly. You can download from the GWB website the extended font used beginning with GWB12, or the legacy "GWB Symbol" font from older versions of the software.

## Scatter Data

---

**Act2**, **Tact**, **Gtplot**, **P2plot**, and **Xtplot** can overlay data points (scatter data) on the diagrams that they produce. The preferred way to add scatter data is to import it from a **GSS** spreadsheet (".gss" file, see the **Using GSS** chapter of the **GWB Essentials Guide**). The old method of importing a specially formatted table from a text file still works, however.

To use this method to overlay a scatter plot of data points onto a diagram, first prepare a table of the data in a plain text (".txt") file created with an editor like Notepad.

The first line in the table is a header that names each column; e.g., Na+, pH, and so on. Enclose multiword names in quotes (e.g., "Mass solution", "Dissolved solids", etc.).

Subsequent lines contain the numeric data. A lack of data may be indicated by a string such as "n/d". To add error bars to the data points, enter a triplet of values separated by vertical bars (|). The values represent the minimum extreme of the error bar, the data point, and the maximum extreme. Exclude blank spaces from the triplet, or enclose it in quotes. The entry 0.5|2.0|3.5, for example, signifies a data point at 2.0 with an error bar extending from 0.5 to 3.5. The entry 0.5||3.5 prescribes the same error bar, omitting the symbol representing the data point.

The columns may be separated by any number of spaces or tabs. Comments may be placed anywhere in the table following a "#" sign. Comments extend from the # sign to the end of the line. You may also include, as separators in the table, blank lines or lines of dashes (-) underscores (\_), or equal signs (=).

You may represent individual data points with a special symbol from among the choices in **Figure 9.1**, a color from among the choices in **Figure 9.2**, and a point size. To do so, append any or all of the following to the data line in question: the symbol name, its color, and its point size, using a string such as "12pt". Beginning with **GWB12**, the names of the unfilled markers (box, circle, delta, del, caution, mobius, and pentagram) can be appended with a cardinal direction (-n, -s, -e, -w) to produce additional shapes (e.g. box-n specifies a box in which the "north", or upper half, is filled). The box and caution symbols can additionally be appended with intercardinal directions (-nw, -se, -sw, -ne).

To load the dataset, select **File** → **Open** → **Scatter Data...** Choosing the file selection dialog's **Edit** button allows you to modify the contents of the scatter data file.

■	square	□	box
●	blot	○	circle
▲	triangle	△	delta
▼	yield	▽	del
◆	diamond	◇	caution
•	bullet	◦	degree
★	star	☆	pentagram
⌘	hourglass	⊗	mobius
+	plus	×	cross

**Figure 10.1** Symbols for plotting scatter data.

To see the changes on the diagram, use the **Open** button to reload the data file after saving the changes. You may clear the scatter data by choosing the **OFF** button.

## 10.1 Act2 and Tact

To overlay a scatter plot of data points onto an **Act2** or **Tact** diagram, first prepare a table of the data in terms of log activities and fugacities. The first line identifies each of the table's columns in terms of a species or gas name, the ratio of two species or gases, or with the special labels "pH", "Eh", "pe" and "T(C)". The species or gas names should be those that appear on the diagram axes, or the original basis members of the thermodynamic dataset. When the axis variable is an activity ratio, the label should be formatted as numerator species<sup>power</sup>/denominator species<sup>power</sup>. A power of 1 does not need to be written explicitly. The activity ratio  $a_{Ca^{++}}/a_{H^+}^2$ , for example, should be written  $Ca^{++}/H^+^2$ .

Subsequent lines contain the numeric data in logarithmic form, except for the linear variables pH, Eh, pe, and temperature data. An example of a table dataset to be used with **Act2** is

pH	Na+	Ca++	HCO3-	
6.5	-4.3	-3.9	-3.3	red square 12pt
5.9	-3.2	-5.1	-4.0	# sample UI-4
6.8	-4.4	-3.6	n/d	

Once the table is prepared, click on **File** → **Open** → **Scatter Data...** to select the dataset. The program will read the data and project them onto the diagram. The **Act2** and **Tact** command

```
scatter dataset_name
```

serves the same purpose.

Aquamarine	Green yellow	Medium forest green	Red
Black	Grey	Medium turquoise	Regal blue
Blue	Grey (10%)	Medium slate blue	Salmon
Blue violet	Grey (20%)	Medium spring green	Sandy brown
Brown	Grey (30%)	Medium orchid	Sea green
Cadet blue	Grey (40%)	Medium violet red	Sienna
Coral	Grey (50%)	Medium goldenrod	Sky blue
Cornflower blue	Grey (60%)	Medium aquamarine	Slate blue
Cyan	Grey (70%)	Medium blue	Spring green
Dark slate blue	Grey (80%)	Medium sea green	Steel blue
Dark turquoise	Grey (90%)	Midnight blue	Tan
Dark orchid	Grey (95%)	Navy	Thistle
Dark green	Honeydew	Navy blue	Turquoise
Dark slate grey	Indian red	Old lace	Violet
Dark olive green	Khaki	Orange	Violet red
Dim grey	Light steel blue	Orange red	Wheat
Firebrick	Light grey	Orchid	White
Forest green	Light blue	Pale green	Yellow
Gold	Lime green	Pattens blue	Yellow green
Goldenrod	Magenta	Pink	
Green	Maroon	Plum	

**Figure 10.2** Color names for plotting scatter data.

## 10.2 Gtplot

**Gtplot** can overlay scatter data points on all of the plot types, except the pie and bar charts. To add scatter data to the “special” plots (ternary plot, Piper diagram, etc.), you specify the fluid composition, expressed in terms of thermodynamic components, as described below.

The first line in the table is a header that names each column; e.g., Temperature, pH, Carbon, Na<sup>+</sup>, and so on. Enclose multiword names in quotes (e.g., “Ionic strength”, “Dissolved solids”, etc.). You label the columns as follows:

- Components: Enter the component name (Ca<sup>++</sup>, SiO<sub>2</sub>(aq), etc.).
- Minerals: Enter the mineral name as it appears in the **Gtplot** menus (Quartz, Kaolinite, etc.).
- Species concentration: Enter the species name in parenthesis following the word “molality” (e.g., molality(Na<sup>+</sup>), molality(NaSO<sub>4</sub><sup>-</sup>), etc.).

- Species activity: Enter the species name in parenthesis following the word “activity” (e.g., activity(Na+), activity(NaHCO<sub>3</sub>), etc.).
- Species activity coefficients: Enter the species name in parenthesis following the word “gamma” (gamma(Na+), gamma(NaCl), etc.).
- Elemental composition: Enter the element name (Oxygen, Carbon, etc.).
- Fugacities: Enter the name of the gas (CO<sub>2</sub>(g), Steam, etc.).
- Mineral saturation: Enter the mineral name in parenthesis following the word “logQoverK” (e.g., logQoverK(Albite), "logQoverK(Albite high)", etc.).
- Isotopic composition: Enter the fluid or mineral name in parenthesis following the isotope label (2-H, 18-O, etc.). Examples: "2-H(Bulk-system)", 18-O(CO<sub>2</sub>(aq)).
- Sorbed fractions: Enter the component name in parenthesis following the word “Xsorbed” (e.g., Xsorbed(Pb<sup>++</sup>)).
- Chemical parameters, Physical parameters, and Reactant properties: Enter the variable name as it appears in the **Gtplot** menus (Eh, "Dissolved solids", etc.).

The scatter dataset should contain numerical values for any of the following:

- The composition of the fluid, minerals, sorbate, and bulk system, expressed in terms of thermodynamic components. The components are the original basis entries in the thermo dataset, plus any decoupled redox species.
- One of the other variables (pH, Dissolved solids, Carbonate alkalinity, etc.) passed to **Gtplot** from **SpecE8** or **React**. These variables are those that appear in diagrams, or are used to calculate unit conversions.
- The masses of minerals in the modeled system.
- The concentrations, activities and activity coefficients of the dissolved species (Fe<sup>++</sup>, Na<sup>+</sup>, etc.).
- The elemental composition of the fluid, minerals, and bulk system.
- The fugacities of gases in the fluid.
- The saturation indices ( $\log Q/K$ ) for various minerals.
- The stable isotopic compositions of the fluid, minerals in bulk, individual minerals, and the entire system.
- The fractions of the various components sorbed onto mineral surfaces.

You enter the numerical values in terms of the following units:



- Enter component masses in mg/kg. In this case, concentrations are expressed in terms of the component species, such as  $\text{HCO}_3$  or  $\text{SO}_4$ . A bicarbonate concentration, then, is entered directly in units of mg  $\text{HCO}_3/\text{kg}$ .
- Masses of minerals over the reaction path are entered in grams.
- The unit for species concentration is molality. The activities and the activity coefficients of dissolved species are unitless.
- Elemental compositions are entered in mg/kg. These values are expressed in terms of the elements, a convention not always followed when reporting anionic compositions. In creating a dataset, you need to convert analyses given in mg  $\text{HCO}_3/\text{kg}$  to mg C/kg, for example,  $\text{SO}_4$  to sulfur, and so on.
- Gas fugacities are unitless.
- Mineral saturation is entered as the saturation index,  $\log Q/K$ .
- Isotopic composition is entered in permil on the scale (e.g., SMOW) assumed in the **SpecE8** or **React** calculation.
- Sorbed fractions are entered as fractions.
- Values for the chemical and physical parameters and reactant properties are entered in their native units, which are the units shown when you first select a variable in the **XY Plot** dialog. The native unit for temperature, for example, is  $^{\circ}\text{C}$ , ionic strength and Eh are given in molal and volts, and  $f_{\text{O}_2}$  is entered as log fugacity. The native units for Mass solution and Mass H<sub>2</sub>O are kg; those for Carbonate alkalinity are mg/kg as  $\text{CaCO}_3$ .
- A lack of data may be indicated by a string such as "n/d".

An example of a table dataset to be used with **SpecE8** is

pH	Na+	Fe <sup>++</sup>	"Dissolved solids"
3.3	.15	5	1000
2.6	.12	15	800
6.	.5	n/d	200 # Buffalo Bayou
4.3	.75	0.1	450

To render these data, run **SpecE8** on a problem containing  $\text{Na}^+$  and  $\text{Fe}^{++}$  to produce a "SpecE8\_plot.gtp" file, start **Gtplot**, and load the scatter dataset with **File** → **Open** → **Scatter Data...** From the **Plot** menu, choose the **XY Plot**, for example, and set  $\text{Na}^+$  and  $\text{Fe}^{++}$  on the axes.

Now, if you choose an appropriate unit for the axis ( $\mu\text{g}/\text{kg}$ ,  $\text{mg}/\text{kg}$ ,  $\text{g}/\text{kg}$ , or wt%), the data points for the  $\text{Na}^+$  and  $\text{Fe}^{++}$  columns appear on the plot. You may choose another unit, such as molal concentration or grams, but you must first include in the scatter dataset columns for Mass H<sub>2</sub>O and Mass solution so that **Gtplot** can convert

the data from mg/kg to the plotting unit; if you do not do so, **Gtplot** assumes a value of 1 kg for both variables.

In “special” plots (Piper diagrams, etc.) showing the variable “HCO<sub>3</sub> + CO<sub>3</sub>”, you should enter a value in the scatter dataset for Carbonate alkalinity, in mg/kg as CaCO<sub>3</sub>, so that **Gtplot** can render the variable correctly. If you do not enter a value for alkalinity, the program will render the variable in terms of the reported HCO<sub>3</sub> concentration, if available, in which case the diagram produced will not account for the speciation of carbonate.

In order to see TDS represented as circles on Piper, ternary, and Durov diagrams, you should enter a value for “Dissolved solids” for each sample.

To assign symbols to each column of variables on an *xy* plot, add a line to the table with symbol names, using the choices in **Figure 9.1** (or a place holder such as “--”). Set symbol color using the choices in **Figure 9.2**. Similarly, set point size using fields like 15pt, 30pt, and so on. For example, adding the lines

--	--	circle	square
--	--	red	green
--	--	24pt	30pt

to the example above causes Fe<sup>++</sup> concentration to be represented by 24pt red circles, and Dissolved solids by 30pt green squares.

To assign symbols to each sample (i.e., each line in the dataset), append any or all of the following to the data line in question: the symbol name, its color, and its point size. The symbol specifications on the sample line will be used to represent each sample in all of the special plots. On an *xy* plot, you may choose whether to assign the symbol, its color, and its size according to the variable (by analyte) or the sample (by sample). For example, the table

pH	Na+	Fe <sup>++</sup>	"Dissolved solids"			
3.3	.15	5	1000	circle	24pt	red
2.6	.12	15	800	box	24pt	green
6.	.5	n/d	200	delta	24pt	blue
4.3	.75	0.1	450	caution	24pt	yellow
--	--	circle	square			
--	--	red	green			
--	--	24pt	30pt			

would allow great flexibility in how to depict the data on an *xy* plot.

An example of a table dataset to be used with **React** is

pH	Eh	Iron	Sulfur
3.3	.15	5	100
2.6	.12	15	80

6.	.5	n/d	20	# Buffalo Bayou
4.3	.75	0.1	45	

To render these data, run **React** on a problem containing iron and sulfur to produce a "React\_plot.gtp" file, start **Gtplot**, and load the scatter dataset with the **File** → **Open** → **Scatter Data...** option. From **Plot** → **XY Plot...** choose to plot the elemental composition of the fluid. (Similarly, if you had entered data in columns labeled Fe++ and SO4-, you would choose to plot components in the fluid.) Select either pH or Eh as the variable for the *x* axis, and iron, sulfur, or both, for the *y* axis.

Now, if you choose an appropriate unit for the *y* axis ( $\mu\text{g}/\text{kg}$ ,  $\text{mg}/\text{kg}$ ,  $\text{g}/\text{kg}$ , or wt%), the data points for the iron and sulfur columns appear plotted against pH or Eh. You may choose another unit, such as molal concentration or grams, but you must first include in the scatter dataset columns for Mass H2O and Mass solution so that **Gtplot** can convert the data from  $\text{mg}/\text{kg}$  to the plotting unit; if you do not do so, **Gtplot** assumes a value of 1 kg for both variables.

In a similar fashion, you could plot Eh against pH by choosing the "Chemical parameters" option from **Plot** → **XY Plot...** If scatter values lie outside the plot axis ranges, touch **Ctrl+Z**.

A second example of a scatter dataset to be used with **React** is

"Rxn progress"	activity(NaCl)	"logQoverK(Albite low)"
.3	5.13e-9	.2
.5	5.15e-9	.1
.7	5.17e-9	.15
.9	5.19e-9	n/d

To render these data, run **React** on a problem containing sodium and silicon to produce a "React\_plot.gtp" file, start **Gtplot**, and load the scatter dataset with the **File** → **Open** → **Scatter Data...** option. From **Plot** → **XY Plot...** choose to plot "Species activity". Select "Reaction Progress" as the variable for the *x* axis, and "NaCl" for the *y* axis. Choose to plot "Mineral Saturation" to show the scatter data for the saturation state of the mineral "Albite low".

### 10.3 P2plot

**P2plot** can overlay scatter data points on the 2D diagrams and cross-section plots it produces. The tables should be formatted like they are for **Gtplot**.

### 10.4 Xtplot

**Xtplot** can overlay scatter data points on all of the plot types, except the pie and bar charts. As with datasets for **Gtplot** and **P2plot**, the first line in the table is a header that names each column. Use the same labels presented for **Gtplot**, with the addition of the following variables specific to **Xtplot**:

- Label position and time as Length, Width, or Time.

An example of a table dataset to be used with **Xtplot** is

Time	pH	Zinc	Sulfur	
.15	3.3	5	100	
.30	2.6	15	80	
.50	6.	n/d	20	# Sample 98-010
.55	4.3	0.1	45	

To render these data, run **X1t** or **X2t** on a problem containing zinc and sulfur to produce an "X1t\_plot.xtp" or "X2t\_plot.xtp" file. Start **Xtplot** and load the scatter dataset with **File** → **Open** → **Scatter Data...** From **Plot** → **XY Plot...** choose to plot the elemental composition of the fluid versus time. (Similarly, if you had entered data in columns labeled "Fe++" and "SO4-", you would choose to plot components in the fluid.) Select zinc, sulfur, or both, as variables to plot on the *y* axis.

Now, if you choose an appropriate unit for the *y* axis ( $\mu\text{g}/\text{kg}$ ,  $\text{mg}/\text{kg}$ ,  $\text{g}/\text{kg}$ , or  $\text{wt}\%$ ), the data points for the zinc and sulfur columns appear plotted against time. You may choose another unit, such as molal concentration or grams, but you must first include in the scatter dataset columns for "Mass H2O" and "Mass solution" so that **Xtplot** can convert the data from  $\text{mg}/\text{kg}$  to the plotting unit.

Alternatively, choose to plot "Chemical parameters" and set pH as the *y* axis variable. In this case, the scatter values for the column labeled pH appear on the plot. If scatter values lie outside the plot axis ranges, touch **Ctrl+Z**.

A second example of a scatter dataset to be used with **Xtplot** is

Length	Width	activity(NaOH)	logQoverK(NaBr)
2e4	2e4	5e-12	-12
4e4	4e4	6e-12	-10
6e4	6e4	7e-12	-6
8e4	8e4	8e-12	n/d

To render these data, run **X1t** or **X2t** on a problem containing sodium to produce a "X1t\_plot.xtp" or "X2t\_plot.xtp" file, start **Xtplot**, and load the dataset with **File** → **Open** → **Scatter Data...** From **Plot** → **XY Plot...** choose to plot "Species activity" and set "NaOH" as the *y*-axis variable. Select "X position" as the variable for the *x* axis. Choose to plot "Mineral Saturation" to show the scatter data for the saturation state of "NaBr". You must include both "Length" (X position) and "Width" (Y position) values to show the scatter data on a mapview plot.

# Multiple Analyses

---

It is not uncommon to have stored in a spreadsheet the results of a number of chemical analyses that you would like to enter – one at a time – into one of the GWB applications. You might wish to use **SpecE8**, for example, to figure calcite saturation or CO<sub>2</sub> fugacity for a group of analyses.

You can store multiple analyses in a **GSS** spreadsheet, select one or more samples, then launch **SpecE8** or **React with Analysis** → **Launch...** An instance will be launched configured with the values in the first sample. Alternatively, you can add calculated analytes for all of your analyses to your **GSS** spreadsheet directly. See “Calculating analytes” in the **Using GSS** chapter of the **GWB Essentials Guide**.

If you have relatively few analyses in another type of spreadsheet, you may use the GWB’s “drag and drop” feature. Highlight data for each analysis in the spreadsheet, left-click, drag into the GWB app, and calculate the desired result. For details, refer to the **Importing data** section of the **Introduction** to the **GWB Essentials Guide**.

Given a large number of chemical analyses, this procedure becomes tedious. It is best in this case to prepare a short script that performs the operations automatically, adding the results to the spreadsheet. This chapter describes how to do so.

You may also want to take advantage of the “scatter data” feature of the GWB, which allows chemical analyses to be overlain as data points on diagrams produced by **Act2**, **Tact**, **Gtplot**, and **Xtplot**. For more information, refer to the **Scatter data** sections of the corresponding chapter (**Using Act2**, and so on) in the GWB documentation set.

## 11.1 Calculation procedure

Suppose you have a number of chemical analyses stored in an Excel spreadsheet, and you would like to add to the spreadsheet results calculated by one of the GWB applications. To do so, follow this procedure:

- Save the spreadsheet from Excel as a tab-delimited text file. Go to **File** → **Save As...** and choose “Text (Tab delimited) (\*.txt)” or “Unicode Text (\*.txt)” as the file type. Excel will create a new file with a “.txt” file extension.
- Prepare and run a GWB script, such as the one in the next section, that runs within the GWB application. The script takes the text file as input and produces

a new text file containing the original data as well as the calculation results. The **Control Scripts** chapter in this **GWB Reference Manual** describes how to prepare GWB scripts. An example of such a script, which you may take and modify for your purposes, is given in the next section.

- Open (**File** → **Open...**; choose “All Files (\*.\*)” as the file type) the resulting text file in Excel. You can now save this file as an Excel spreadsheet (a “.xls” file).

The next section carries you through an example of this procedure.

## 11.2 Example calculation

The files installed under directory “Scripts\Spreadsheet” within the GWB installation directory (e.g., “\Program Files\GWB”) provide an example of using a script to process multiple analyses from an Excel spreadsheet. To run the example, copy files “Spreadsheet.xls” and “Script.xls” from this directory to a convenient location on your computer, such as the “My Documents” folder.

The analyses are stored in file “Spreadsheet.xls”. Open this file in Excel by double-clicking on it. Save it as a tab-delimited text file, as described in the previous section. This creates a file “Spreadsheet.txt”.

You may examine this file with an editor such as Notepad. It looks like

ID	pH	HCO <sub>3</sub> <sup>-</sup>	SO <sub>4</sub> <sup>--</sup>	Cl <sup>-</sup>	Ca <sup>++</sup>	Mg <sup>++</sup>	Na <sup>+</sup>
GW-12	6.78	585.7	309	56	205.6	63.9	21.4
GW-13	6.78	585.7	311	56.2	214.9	66.8	22.6
GW-14	6.85	652.8	582	42.6	269.2	89	25.8
GW-15	7	558.2	400	65.4	216.2	65.7	32

(and so on)

The first line in the file contains column headers including “pH” and various basis species, and subsequent lines contain the numerical data. The headers will be used together with the numerical values to create SpecE8 commands such as

```
pH = 6.78
HCO3- = 585.7 mg/kg
```

File “Script.sp8” contains a **SpecE8** script that reads “Spreadsheet.txt”, calculates CO<sub>2</sub> fugacity and calcite saturation, and writes a file “Output.txt”. You can modify this script for your own purposes.

The script is shown below. For clarity, **SpecE8** commands within the script are listed in bold face and comment lines are in italics; the actual file, of course, is simply a text file.

```
script start
# Set up the input and output.
set in_id [open "Spreadsheet.txt" r]
set out_id [open "Output.txt" w]
```

```

fconfigure $out_id -encoding unicode

# First line contains column headers; check for Unicode.
gets $in_id headers
if ![string is ascii $headers] {
  close $in_id
  set in_id [open "Spreadsheet.txt " r]
  fconfigure $in_id -encoding unicode
  gets $in_id headers
}
puts $out_id "$headers\tf CO2\tCalcite SI"

# Loop through remaining lines.
gets $in_id aline
report set_digits 4
while {$aline != ""} { set i 0
  reset; balance on CI-

  # Set basis constraints from input data.
  foreach a [lrange $aline 1 end] {
    incr i 1
    if {[lindex $headers $i] == "pH"} {
      pH = $a
    } else {
      $a = [lindex $headers $i] mg/kg
    }
  }

  # Run SpecE8 calculation and write data + results.
  go
  foreach a [lrange $aline 0 end] {
    puts -nonewline $out_id "$a\t"
  }
  if {[report success]} {
    puts $out_id \
      "[report fugacity CO2(g)]\t[report SI Calcite]"
  } else {
    puts $out_id "Did not converge"
  }

  # Next line of input.
  gets $in_id aline
}
# Clean up.

close $out_id

```

```
close $in_id  
quit
```

Double click on file “Script.sp8” to start **SpecE8** and execute the script. The program will produce a file “Output.txt” that contains the original data with the values calculated for CO<sub>2</sub> fugacity and calcite saturation appended as new columns. The file looks like

D	pH	HCO <sub>3</sub> <sup>-</sup>	SO <sub>4</sub> <sup>--</sup>	...	Na <sup>+</sup>	f CO <sub>2</sub>	Calcite SI
GW-12	6.78	585.7	309		21.4	0.06537	0.2281
GW-13	6.78	585.7	311		22.6	0.06515	0.2427
GW-14	6.85	652.8	582		25.8	Did not converge	
GW-15	7	558.2	400		32	0.04134	0.4714

Open “Output.txt” in Excel by selecting **File** → **Open...** and choosing “All Files (\*.\*)” for the file type. Follow the Excel “Text Import Wizard”, accepting the default at each step: “Delimited” file type, “Tab” delimiter, “General” data format. The calculation results will appear as would any spreadsheet, which you may save as an Excel (“.xls”) file.



# Remote Control

---

You can run various GWB application programs not only by hand from the keyboard, but by “remote control” from a program or script you write. Note that this is now a legacy feature that has been replaced by the [Plug-in Feature](#), and is no longer supported. **Rxn**, **Act2**, **Tact**, **SpecE8**, **React**, **X1t**, and **X2t** can be run in this way. The program you write serves as the “master program”, which controls the GWB application as a “slave program”.

In writing a program of your own, for example, you might need to determine the saturation state of calcite in a fluid of arbitrary composition. Instead of developing code to calculate the distribution of mass and mineral saturation states in a fluid, you could invoke **SpecE8** from within your program and let it do the work for you.

Similarly, you could use the remote control feature to balance reactions with **Rxn**, calculate activity diagrams with **Act2**, or figure the results of irreversible reaction paths with **React**. In each case, you configure the GWB application by sending text commands, trigger the calculation, and then retrieve the calculation results to use for your own purposes.

You can transfer the results from the slave application to the master program with the “report” command, as described in the [Report Command](#) chapter in this **GWB Reference Manual**. Or, as is especially useful with **Act2** and **Tact**, you can copy calculation results such as activity diagrams to your computer’s clipboard, where they can be retrieved as graphical images. To do so, you use the “clipboard” command. Finally, you can simply read datasets, such as “SpecE8\_output.txt” produced by the GWB applications, into the master program.

Your program, the master program, controls a GWB application as a slave program through a interprocess communications device known as a “pipe”. (Pipes are not available in MS Windows 98 or ME, so you cannot use the remote control feature under these operating systems.) There are two ways to set up the communications. You can create two “unnamed pipes”, one for input to and the other for output from the GWB application. Or, you can establish a “named pipe”, which allows bidirectional data transfer.

Using a named pipe has a couple of advantages over unnamed pipes. First, the master program’s standard input and output streams are available for use in the normal manner. Second, by establishing two or more pipes with different names, any number of copies of the GWB application programs can be invoked simultaneously.

Sending data through a pipe is much like writing to a file, and receiving data is like reading from a file. Running a GWB application by remote control, therefore, involves little more than standard programming techniques already familiar to anyone with modest programming experience.

To run a GWB application by remote control, you start it from the master program using the “-pipe” command line option. This option is followed by the name of the pipe, or for an unnamed pipe the keyword “stdio”. In MS Windows, pipes are located in a pseudo-directory at the top level of the file system called “pipe”.

If the master program, for example, has created an unnamed pipe, it could invoke program **SpecE8** using the command

```
spece8 -pipe stdio
```

In this case, the standard output stream of the master serves as standard input to the slave, and the slave’s standard output stream is the master’s input stream.

Similarly, if the master program has created a named pipe called “\pipe\mypipe”, it could invoke program **SpecE8** by using the command

```
spece8 -pipe \pipe\mypipe
```

The master program could then communicate with **SpecE8** by writing to and reading from the pipe.

The sections below show examples of how the remote control feature can be implemented in the C++ programming and Tcl scripting languages, using named and unnamed pipes.

### 12.1 C++ program using unnamed pipes

In writing a program in C++, you will likely find it easiest to use a set of helper functions contained in file “RC\_helper.cpp”, a copy of which is installed in the “src” subdirectory of the GWB installation directory (e.g., in “\Program Files\GWB\src”). The helper functions in this file include:

<code>OpenGwbApplic</code>	Start the GWB program of interest.
<code>SendCommand</code>	Transmit a command to the GWB app, and, optionally, receive the results of the command.

There is a version of each function for unnamed and named pipes. By using these functions, the programmer can avoid worrying about the details of communication between the master and slave programs.

If you `#include` the header file “RC\_helper.h” at the top of your master program, the helper functions will be available. Of course, you can modify and extend these functions for your own purposes, if you wish. The program must also be compiled with the “RC\_helper.cpp” file, also provided in the same location.

In the following example, included in the “src” subdirectory, a console program invokes **React** using unnamed pipes to integrate a kinetic rate law for quartz dissolution

at 100°C. Array “script” is a vector of pointers to the commands needed to configure **React** and trigger the calculation.

The program opens **React**, sends it the commands in array “script”, uses the “report” command to request the calculation result, which it extracts from **React’s** response using “sscanf”. Note that since the program uses unnamed pipes, output to the console is sent via the “stderr” output stream.

```
/* RC_example1.cpp */

#include "Program Files/GWB/src/RC_helper.h"
#include <stdio.h>

char* script[] = {
    "reset",
    "time begin = 0 days, end = 5 days",
    "T = 100",
    "SiO2(aq) = 1 umolal",
    "react 5000 g Quartz",
    "kinetic Quartz rate_con = 2.e-15 surface = 1000",
    "go",
    ""
};

int main(int argc, char* argv[])
{
    char line[200];
    char discard[20];
    char** command;
    double SI_Quartz;

    fprintf(stderr, "Starting program React.\n");
    OpenGwbApplic("\\Program Files\\gwb\\react.exe");

    for (command = script; **command; command++)
        SendCommand(*command);

    SendCommand("report SI Quartz", line, sizeof(line));
    sscanf(line, "%lg", &SI_Quartz);
    fprintf(stderr, "Value of SI Quartz is %g.\n", SI_Quartz);

    SendCommand("quit");

    fprintf(stderr, "press return to exit> ");
    gets_s(discard);
    return 0;
}
```

## 12.2 C++ program using named pipes

As a second example of a master program written in C++, we open two copies of **React** as slave programs; the copies run simultaneously. To do so, we establish two pipes, using the “Pipe” class defined in “RC\_helper.h”. In this case, the standard I/O channels are available to the program, so we need not direct console messages to “stderr”.

```
/* RC_example2.cpp */

#include "/Program Files/GWB/src/RC_helper.h"
#include <stdio.h>

char* script1[] = {
    "reset",
    "time begin = 0 days, end = 5 days",
    "T = 100",
    "SiO2(aq) = 1 umolal",
    "react 5000 g Quartz",
    "kinetic Quartz rate_con = 2.e-15 surface = 1000",
    "go",
    ""
};

char* script2[] = {
    "reset",
    "time begin = 0 days, end = 5 days",
    "T = 100",
    "SiO2(aq) = 1 umolal",
    "react 5000 g Quartz",
    "kinetic Quartz rate_con = 2.e-15 surface = 750",
    "go",
    ""
};

int main(int argc, char* argv[])
{
    char line[200];
    char discard[20];
    char** command1;
    char** command2;
    double SI_Quartz;
    Pipe pipe1("pipe1");
    Pipe pipe2("pipe2");

    printf("Open two copies of React.\n");
    OpenGwbApplic(pipe1,
```

```

        "\\Program Files\\gwb\\react.exe");
OpenGwbApplic(pipe2,
        "\\Program Files\\gwb\\react.exe");

for (command1 = script1, command2 = script2;
    **command1 || **command2;
    **command1 ? command1++ : 0, **command2 ? command2++ : 0) {
    if (**command1)
        SendCommand(pipe1, *command1);
    if (**command2)
        SendCommand(pipe2, *command2);
}

SendCommand(pipe1, "report SI Quartz", line, sizeof(line));
sscanf(line, "%lg", &SI_Quartz);
printf("SI Quartz for 1000 cm2/g is %g.\n", SI_Quartz);

SendCommand(pipe2, "report SI Quartz", line, sizeof(line));
sscanf(line, "%lg", &SI_Quartz);
printf("SI Quartz for 7500 cm2/g is %g.\n ", SI_Quartz);

SendCommand(pipe1, "quit");
SendCommand(pipe2, "quit");

printf("press return to exit> ");
gets_s(discard);
return 0;
}

```

## 12.3 Tcl script using unnamed pipes

You may find it especially useful to invoke GWB applications from within another application or calculation environment, such as a spreadsheet, word processor, or mathematical interpreter. You can do so, as long as the environment has scripting abilities and can open pipes.

As an example, we repeat the first example above in the Tcl scripting language. As in the C++ example, a number of helper functions are available in file "RC\_helper.tcl", installed with the GWB in subdirectory src. The complete Tcl script is given below.

```

source RC_helper.tcl

set cmdlist {
    reset
    {time begin = 0 days, end = 5 days}
    {T = 100}
    {SiO2(aq) = 1 umolal}
}

```

```

    {react 5000 g Quartz}
    {kinetic Quartz rate_con = 2.e-15 surface = 750}
    go
  }

  OpenGwbApplic {/Program Files/gwb/react/react.exe}
  foreach cmd $cmdlist {
    SendCommand $cmd
  }
  SendCommand {report SI Quartz} line
  puts "SI Quartz is $line."
  SendCommand quit

```

## 12.4 Perl script using unnamed pipes

As a final example, we show how to run **React** by remote control from a Perl script. The example below uses the object oriented Perl module "RC\_helper.pm", included in the "src" subdirectory of the GWB installation.

```

#!/usr/bin/env perl
use strict;
use warnings;
use lib "\\Program Files\\Gwb\\src";
use RC_helper;

my $script = <<SCRIPT;
reset
time begin = 0 days, end = 5 days
T = 100
SiO2(aq) = 1 umolal
react 5000 g Quartz
kinetic Quartz rate_con = 2.e-15 surface = 1000
go
SCRIPT

print "Starting program React.\n";
my $react = RC_helper->new("\\Program Files\\Gwb\\react.exe");
for my $command (split /\n/, $script) {
    $react->send_command($command);
}
my $SI_Quartz = $react->send_command("report SI Quartz");
print "Value of SI Quartz is $SI_Quartz\n";

$react->send_command("quit");

```

# Index

---

- acidity, [102](#)
- activity, [42](#), [98](#)
- activity coefficients, data for calculating, [13](#)
- alkalinity, [42](#), [101](#)
- angle, [103](#)
- aqueous, [42](#)
- aqueous species, in thermodynamic dataset, [15](#), [17](#)
- arbitrary reaction definition, in surface dataset, [35](#)
- arbitrary reaction definition, in thermodynamic dataset, [25](#)
  
- basis, [42](#)
- basis species, in surface dataset, [30](#)
- basis species, in thermodynamic dataset, [15](#)
- biomass, [42](#)
- boltzman, [42](#)
- bulk\_volume, [42](#)
  
- C++ plug-in, [64](#)
- C++ program, [122](#), [124](#)
- C++ programs, [67](#)
- calculation procedure, multiple analyses, [117](#)
- calculator, [5](#)
- carbon, [103](#)
- cat\_area, [42](#)
- cd-music model, in surface dataset, [34](#)
- charge, [42](#)
- charged uncomplexed sites, in surface dataset, [34](#)
- chlorinity, [42](#)
- clipboard, [105](#)
- coef\_dispersion, [42](#)
- colloids, [42](#)
- colonies per volume, [103](#)
- color, [103](#)
  
- command line interface, [3](#)
- compressibility, [102](#)
- concentration, [42](#), [97](#)
- configuration, [42](#)
- constraints, [42](#)
- contact\_area, [42](#)
- control script, example, [61](#)
- control scripts, [59](#)
- control statements in scripts, [60](#)
- corrosivity, [103](#)
- couples, [42](#)
- custom plug-in, [90](#)
  
- database, [44](#)
- Deltat, [44](#)
- density, [101](#)
- deuterium (<sup>2</sup>H), [103](#)
- diffusion and dispersion coefficients, [102](#)
- discharge, [44](#)
- distance, [98](#)
- distribution coefficients, [102](#)
  
- EC, [44](#)
- efflux, [44](#)
- Eh, [44](#)
- electrical conductivity, [102](#)
- electrical potential, [98](#)
- elements, [44](#)
- elements, in thermodynamic dataset, [15](#)
- End line, in surface dataset, [34](#)
- end\_members, [44](#)
- energy, [102](#)
- energy content, [102](#)
- equil\_eqn, [44](#)
- equil\_favors, [44](#)
- equil\_temp, [44](#)
- exchange capacity, [102](#)

## Index

---

- exchange\_capacity, 44
- FA, 44
- FD, 44
- flow rate, 100, 101
- foaming agents, 103
- font for data markers, 108
- formulae for aqueous species, in thermodynamic dataset, 25
- Fortran plug-in, 73
- Fortran programs, 77
- free electron, in thermodynamic dataset, 18
- free electron, in thermodynamic dataset, 25
- freeflowing, 44
- fugacity, 44, 98
- fugacity coefficients, in thermodynamic dataset, 25
  
- gamma, 44
- gas\_pressure, 44
- gases, 44
- gases, in thermodynamic dataset, 20
- get\_default\_units, 44
- get\_units, 44
- graphics output, 105
  
- hardness, 46
- hardness\_carb, 46
- hardness\_ncarb, 46
- header data, 12, 28
- header variables, in thermodynamic dataset, 13, 24
- heat capacity, 102
- heat source, 102
- history substitution, 3
- hyd\_pot, 46
  
- imbalance, 46
- imbalance\_error, 46
- inert\_volume, 46
- influx, 46
- initial lines, 12, 28
- initial lines, in thermodynamic dataset, 24
- IS, 46
- isotopes, 46, 56
- iterations, 46
  
- Kd, 46
- keyboard shortcuts, 7
  
- Legacy features, 2
- legacy formats, surface datasets, 34
- legacy formats, thermodynamic datasets, 24
- legacy temperature expansion, for virial coefficients, 22
- logfO2, 46
- logk, 46
- logks, 46
- logQoverK, 46
  
- mass, 46
- mass\_reacted, 48
- mass\_remaining, 48
- mineral\_mass, 48
- mineral\_volume, 48
- minerals, 48
- minerals, in thermodynamic dataset, 19
- mixing\_fraction, 48
- mobility, 48
- multiple analyses, 117
- multiple analyses, example calculation, 118
- mv, 48
- mw, 48
  
- named pipes, 124
- naqueous, 48
- nbasis, 48
- ncolloids, 48
- ncouples, 48
- nelements, 48
- nend\_members, 48
- new temperature expansion, for SIT coefficients, 23
- new temperature expansion, for virial coefficients, 22, 23
- ngases, 48
- nisotopes, 48
- nlogks, 48
- nminerals, 48
- Nnode, 48
- nreactants, 48
- nsolid\_solutions, 48
- nsorbed, 50
- nsorbing\_surfaces, 50
- nstagnant, 50
- nsurf\_species, 50
- Nx, 50
- Ny, 50



- odor, 103
- online documentation, 6
- options, 50
- oxide components, in thermodynamic dataset, 20
- oxygen, 103
- oxygen demand, 103
  
- pe, 50
- percent removal, 102
- Perl script, 126
- permeability, 50, 102
- pH, 50, 98
- Pitzer coefficients, in thermodynamic dataset, 21
- plug-in feature, 63, 64, 73, 77, 84, 90
- PMC, 103
- polydentate sorption, in surface dataset, 34
- polyfit, 50
- polynomial expansions, in thermodynamic datasets, 11
- pore volumes, 102
- porosity, 50
- pressure, 50, 102
- principal temperatures, in thermodynamic dataset, 13, 24
  
- PV, 50
- Python plug-in, 84
- Python programs, 87
  
- Q/K, 102
- QoverK, 50
  
- radioactive emission per volume, 103
- radioactive exposure over time, 103
- rate\_con, 50
- ratecon\_unit, 50
- reactant\_area, 50
- reactant\_type, 50
- reactants, 50
- reaction, 50
- reaction rate, 98–100
- reactions, in surface dataset, 29
- reactions, in thermodynamic dataset, 14
- redox couples, in thermodynamic dataset, 25
- redox coupling reactions, in thermodynamic dataset, 16
- remote control, 121
- report command, 37
  
- results, 66, 76, 86, 93
- rxn\_rate, 52
  
- saving images, 106
- scatter data, 109
- scripts, interaction with application, 60
- sections, in a surface dataset, 27
- sections, in a thermodynamic dataset, 9
- set\_digits, 52
- set\_node, 52
- set\_units, 52
- SI, 46
- Sionst, 52
- SIS, 52
- SIT activity model, in thermodynamic dataset, 24
- SIT coefficients, in thermodynamic dataset, 21
- site density units, in surface dataset, 35
- solid solutions, in thermodynamic dataset, 19, 25
- solid\_solutions, 52
- soln\_compressibility, 52
- soln\_density, 52
- soln\_expansivity, 52
- soln\_mass, 52
- soln\_viscosity, 52
- soln\_volume, 52
- sorb\_area, 52
- sorbed, 52
- sorbing minerals, in surface dataset, 30
- sorption capacity, 102
- sorption\_capacity, 52
- special characters, 5
- species pairs and triplets, virial coefficients for, 22
- species, in surface dataset, 29
- species, in thermodynamic dataset, 14
- specific discharge, 101
- spelling completion, 3
- stagnant, 52
- startup files, 6
- success, 52
- sulfur, 103
- surf\_charge, 52
- surf\_charge0, 52
- surf\_chargedb, 52
- surf\_charged, 52
- surf\_potential, 52

## Index

---

- surf\_potential0, [52](#)
- surf\_potentialb, [52](#)
- surf\_potentiald, [52](#)
- surf\_species, [54](#)
- surf\_type, [54](#)
- surface charge, [102](#)
- surface datasets, [27](#)
- surface species, in surface dataset, [31](#)
- surfaces, [54](#)
- system commands, [6](#)
  
- T, [54](#)
- T-table expansions, in thermodynamic datasets, [10](#)
- Tcl license agreement, [62](#)
- Tcl script, [125](#)
- TDS, [54](#)
- temperature, [54](#), [103](#)
- temperature expansion, in thermodynamic dataset, [24](#)
- Temperature expansions, [10](#), [28](#)
- temperature expansions, for virial coefficients, [21](#)
- temperature expansions, in surface dataset, [35](#)
- temperature range of validity, virial coefficients, [22](#)
- temps, [54](#)
- Tend, [54](#)
- text size in the GWB windows, [7](#)
- thermal conductivity, [102](#)
- thermal expansivity, [102](#)
- thermal transmissivity, [102](#)
- thermo data line, in surface dataset, [34](#)
- thermodynamic datasets, [9](#)
- three-layer model, in surface dataset, [34](#)
- Time, [54](#)
- time, [98](#)
- Tionst, [46](#)
- total\_biomass, [54](#)
- total\_reacted, [54](#)
- TPF, [54](#)
- tritium ( $^3\text{H}$ ), [103](#)
- Tstart, [54](#)
- turbidity, [103](#)
  
- unit conversion, [97](#)
- unnamed pipes, [122](#), [125](#), [126](#)
  
- velocity, [54](#), [101](#)
- virial coefficients, in thermodynamic dataset, [21](#)
- viscosity, [102](#)
  
- Watact, [54](#)
- watertype, [54](#)
- Wmass, [54](#)
  
- xcoef\_dispersion, [54](#)
- xdischarge, [54](#)
- Xfree, [54](#)
- Xi, [54](#)
- xpermeability, [54](#)
- xsorbed, [54](#)
- xss, [54](#)
- xvelocity, [54](#)
- xycoef\_dispersion, [54](#)
  
- ycoef\_dispersion, [54](#)
- ydischarge, [54](#)
- ypermeability, [56](#)
- yvelocity, [56](#)