Extended Aerosol Inorganics Model (*E-AIM*)

Specifications of subroutines **EAIMstr** and **EAIMvar**

(these Fortran routines can be used to call the *E-AIM* model within other programs, using a dynamic link library)

Version 1.0 (7 January 2015): describes use for inorganic systems only.

This manual is part of a zip file package that also contains the libraries and example programs.

Author: Simon L. Clegg

University of East Anglia

Email: s.clegg@uea.ac.uk

Phone: +44 (0)1603 593185

Skype: slclegg

Table of Contents

1. Introduction

This document describes the specification of two Fortran subroutine versions of the Extended Aerosol Inorganics Model (*E-AIM*, http://www.aim.env.uea.ac.uk/aim/aim.php). Either one of the subroutines can be called within other programs to carry out calculations for *E-AIM* Models I-IV, similar to those on the *Simple*, *Comprehensive*, and *Aqueous Solution* pages on the website. The subroutines are provided as a dynamic link library.

The subroutines described in sections 4 and 5 of this document have interfaces – i.e., sets of arguments – that are intended to be suitable for use in both Fortran and non-Fortran programming environments. (The arguments are simple strings, integer and real scalar variables, and Fortran 'explicit shape' arrays only.) Two alternative versions of the subroutines are described in section 6. These have fewer arguments, but can only be linked to another Fortran program which has been compiled using the Intel compiler. The information that has to be provided for every calculation is essentially the same as described on the website for *Batch* mode input (e.g., see http://www.aim.env.uea.ac.uk/aim/model2/model2d.php).

The first of the two subroutines that can be called from the dynamic link library is **EAIMstr**. For this routine most of the input data are entered in a text string argument. The data that must be provided in the string are the same as for the web-based *Batch* mode input referred to above. The second, alternative, subroutine is called **EAIMvar**. For this routine the same input data are required, but are entered in real and integer variables and arrays (not using a text string). Both routines will give the same results. The only differences are, first, that for **EAIMvar** the user can switch off calculations of solution and solid densities, and surface tensions, if they are not needed (in **EAIMstr** they are always calculated, if possible). This may yield a small increase in speed. Second, **EAIMvar** may be preferable for use in iterative calculations because the direct input of real numbers – rather than via a text string – makes it easier to avoid a loss of precision.

So far, the routines have been called successfully from other Fortran programs, and have been run within Mathematica, using its NETLink facility.

2. Libraries containing the *E-AIM* subroutines

The *E-AIM* subroutines are provided in the form of dynamic link libraries (dlls), which have the extension .dll. There are four different versions of the library, all of which contain the routines described in sections 4 and 5 of this document. The libraries have the following names:

- EAIM_d32.dll – the model code runs in double precision, and has been compiled for execution on Windows machines with a 32 bit version of the Windows operating system.
- EAIM_d64.dll – the model code runs in double precision, and has been compiled for execution on Windows machines with a 64 bit version of the Windows operating system.
- EAIM_q32.dll – the model code runs in extended (quad) precision, and has been compiled for execution on Windows machines with a 32 bit version of the Windows operating system.
- EAIM_q64.dll – the model code runs in extended (quad) precision, and has been compiled for execution on Windows machines with a 64 bit version of the Windows operating system.

The libraries have been compiled using the Intel Fortran compiler, version 13.1.3 or later. They are supplied with the corresponding .lib and .exp files, which are needed if the dll is going to be used by another Fortran program. The libraries all contain both **EAIMstr** and **EAIMvar**. They are supplied with two example programs (Test_str.for and Test_var.for), which are described in Appendix 2.

An alternative copy of the library, which has versions of EAIMstr and EAIMvar contained within a Fortran module, is described in section 6. (The name of this library is EAIM_module_q64.dll.)

What is the difference between the double precision and quad precision versions of the model and subroutines? The calculation of the amounts of dissolved $OH^-(aq)$ in a solution that is acidic, or partitioning of $H_2SO_4(g)$ into the gas phase (H_2SO_4 has very low volatility), requires a precision of greater than 15 digits ("double precision") in order to obtain an accurate result. This is inherent in the method used to solve for the equilibrium state of the chemical system. The "extended precision" versions of the model, in which real numbers have about 32 digits of precision, are thus the most general in terms of the kinds of calculations they can do. These are the ones that run on the *E-AIM* website. However, the extended precision code runs at least a factor of 5 slower than the double precision code which is just as good for many types of problem in which the amounts of the different variables (the moles of each species) do not differ by tens of orders of magnitude.

3. What the *E-AIM* subroutines do

When the *E-AIM* routines are first invoked (called) they will carry out a series of checks to determine whether the arguments have the correct sizes, and whether some integer arguments have been set correctly. If these tests are failed, the routines will return (with blank or zero outputs) and with an error message in the string argument **Messages**, and error codes in the string **errorFlags**. Note that these tests are carried out only on the first calls of the routine, and not on later calls.

If the *E-AIM* routines pass the basic checks noted above, they will then open a file called **eaim.err**, which should appear in the same directory as the executable that calls the dll. If the model code detects an error - in the input data for example - a message will be written at the end of the **eaim.err** file (i.e., appended) with a time stamp, and execution of the routines will stop. There will be no other alert or warning, so if the routine does not appear to work, then the error file should be checked for messages. The error file can be deleted by the user if required, because the *E-AIM* routines will create a new **eaim.err** file if one does not already exist.

The code will also look for a file called **eaim.org** (in the same directory as the executable program calling the dll). If the file is found, the number of organic compounds to be included in the calculation is read from it, followed by the thermodynamic properties of each of these compounds. Note that the amounts of these organic compounds, for each problem, must be specified in the input arguments to the subroutine together with the amounts of the inorganic species, as described in sections 4 and 5 below. If the **eaim.org** file does not exist (or if the number of organic species entered in it is zero), the problem will be treated as "inorganic only". This document does not yet contain a description of how to include organic compounds in an *E-AIM* calculation (contact Simon Clegg if you are interested).

Next, the subroutines read the input data for the problem from real, integer, and string arguments that are described below. If these data fail internal tests, or are incomplete, then an explanatory error message and a time stamp will be written at the end of the file **eaim.err**. Execution of the code will then stop, without any other messages or warnings. If the internal tests are passed, the *E-AIM* calculation will be carried out and the results returned to the calling (sub)program in the INTENT(OUT) arguments of the routines.

4. Specification of the *E-AIM* subroutine **EAIMstr**

This section describes the input and output arguments of EAIMstr. The colour codes used below are as follows: **red** – an argument that contains input data for the calculation which must be provided by the user; **blue** – an argument of INTENT(OUT) that contains the results of the calculation; **black** – an integer argument of INTENT(IN) which specifies the length of a string argument, or the size of an array argument. Note that in the specification below REAL (KIND=8) quantities are "double precision" (with approx. 15 digits of precision), and INTEGER (KIND=4) are 4 byte integers (which have a range of ±2147483648).

```
SUBROUTINE EAIMstr(ModelNumber, iOptions, iSizeiOptions, Input, lenInput,
                   iOutput, iSizeiOutput, Output, iOutputLabel,
                   iSizeOutput, lenOutputLabel, Messages, lenMessages,
                   UsageRecord, lenUsageRecord, ErrorFlags, lenErrorFlags)

INTEGER (KIND=4), INTENT(IN) :: iSizeiOptions, iSizeiOutput, iSizeOutput,
                                lenInput, lenOutputLabel, lenMessages,
                                lenUsageRecord, lenErrorFlags, ModelNumber,
                                iOptions(iSizeiOptions)
CHARACTER (LEN=lenInput), INTENT(IN) :: Input

REAL (KIND=8),    INTENT(OUT) :: Output(iSizeOutput)
INTEGER (KIND=4), INTENT(OUT) :: iOutput(iSizeiOutput),
INTEGER (KIND=4) ::              iOutputLabel(lenOutputLabel, iSizeOutput)
CHARACTER (LEN=lenMessages),    INTENT(OUT) :: Messages
CHARACTER (LEN=lenUsageRecord), INTENT(OUT) :: UsageRecord
CHARACTER (LEN=lenErrorFlags),  INTENT(OUT) :: errorFlags
```

4.1 Arguments

1: **ModelNumber** - INTEGER.

On entry: the reference number of the *E-AIM* model to be used in the calculation.

Constraint: 1 ≤ ModelNumber ≤ 4.

Details: This scalar variable selects the *E-AIM* model to be used and must have one of the following values: 1 - Model I; 2 - Model II; 3 - Model III; 4 - Model IV.

Notes: The value will be read when the routine is first called only, or when input argument **iOption(4)** = 1.

2: **iOptions(iSizeIoptions)** - INTEGER array, of size iSizeOptions.

On entry: sets options for the calculation of solution densities and surface tensions, whether thermodynamic consistency tests will be carried out on the results, and whether initialization of the model will be carried out on the current call of the routine. (This initialization is carried out automatically on the first call, irrespective of the option setting.)

Constraint: all values of the array must be either 0 (option off), or 1 (option on).

Details: this is an integer array that is used to specify options to control or include/exclude different elements of the calculation. Possible values are as follows:

```
iOptions(1) = 0, or 1. For EAIMstr this option has no effect. Please use EAIMvar if it
                       is required. (A value of 1 means that densities and volumes of
                       the liquid phase(s) and solid(s) will be calculated, if possible.
                       A value of 0 means that they will not be calculated. Any other
                       value will be trapped as an error.)

iOptions(2) = 0, or 1. For EAIMstr this option has no effect. Please use EAIMvar if it
                       is required. (A value of 1 means that the surface tension(s) of
                       the liquid phase(s) and solid(s) will be calculated if possible.
                       A value of 0 means that they will not be calculated. Any other
                       value will be trapped as an error.)

iOptions(3) = 0, or 1. A value of 0 means that a series of tests for the thermodynamic
                       consistency of the result of the model calculation will *not* be
                       carried out. This may result in slightly faster execution. A value
                       of 1 means that the tests will be carried out, and any failures
                       indicated in the argument errorFlags (see below).

iOptions(4) = 0, or 1. When the subroutine is first called, a series of initialisations
                       are always carried out (these are necessary for the code to function
                       correctly). A value of 0 means that on later calls these initialisations
                       will not be carried out (this is how the model is expected to be run).
                       A value of 1 means that the initializations will be done. This option
                       may be useful for test purposes.

iOptions(5) = 0, or 1. A value of 1 means that a usage record will be written to the
                       string argument UsageRecord (see below for details). A value of
                       zero means no record will be written, and UsageRecord will be blank.
```

3: **iSizeiOptions** – INTEGER.

On entry: the dimension of array **iOptions** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiOptions, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

4: **Input** - CHARACTER, a string of length lenInput.

On entry: contains the input data for the problem to be solved.

Details: this character string contains the input data for the problem to be solved. The data for each problem are specified in the same way as for *Batch* input for the web-based model. This is described in Appendix 1 of this document, and on the following web page: http://www.aim.env.uea.ac.uk/aim/model2/input2d.html. For chemical systems containing only inorganic compounds, only items **a – u** from the descriptions should be entered into **Input**. (If your system contains organic compounds then additional files will be needed to run the model. Please contact Simon Clegg.)

5: **lenInput** – INTEGER.

On entry: the length of character string Input as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenInput, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

6: **iOutput(iSizeiOutput)** – INTEGER, an array of size iSizeiOutput.

On exit: the elements of this array contain information about the results of the chemical calculation.

Details: each element of the array contains the data listed below.

iOutput(1) : an integer from 1 to 3, meaning the following:

0 - the result contains neither an aqueous nor a hydrophobic liquid phase.

1 - the result contains an aqueous phase (but not a hydrophobic liquid phase).

2 - the result contains a hydrophobic liquid phase (but not an aqueous phase).

3 - the result contains both an aqueous phase and a hydrophobic liquid phase.

iOutput(2) : NC, the maximum number of cations in the system.

iOutput(3) : NA, the maximum number of anions in the system.

iOutput(4) : NN, the maximum number of neutral species in the system (including water).

iOutput(5) : NG, the maximum number of gases in the system.

iOutput(6) : NS, the maximum number of solids in the system.

7: **iSizeiOutput** – INTEGER.

On entry: the size of integer array **iOutput** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiOutput, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

8: **Output(iSizeOutput) –** REAL, an array of size iSizeOutput.

On exit: contains the results of the calculation.

Details: this array will contain the results of the *E-AIM* calculation. The quantities contained in each of the elements of array **Output** are given below. See also the output array **iOutputLabels**, in which integer-coded labels corresponding to the output quantities are placed.

Note the following definitions used below: NSOL = NC + NA + NN
NSOL2 = 2 * NSOL
NSOL3 = 3 * NSOL
NSOL4 = 4 * NSOL

Output(1)		- RH, the relative humidity (as a fraction)
Output(2)		- T, temperature (K)
Output(3)		- P, system pressure (atm.)
Output(4)		- V, system volume (m^3)
Output(5)		- density of the aqueous phase (g cm^{-3})
Output(6)		- volume of the aqueous phase (cm^3)
Output(7)		- surface tension of the aqueous phase (mN m^{-1})
Output(8)		- density of the hydrophobic phase (g cm^{-3})
Output(9)		- volume of the hydrophobic phase (cm^3)
Output(10)		- surface tension of the hydrophobic phase (mN m^{-1})
Output(11:	10+NC)	- moles of each cation (aqueous phase)
Output(11+NC:	10+NC+NA)	- moles of each anion (aqueous phase)
Output(11+NC+NA:	10+NSOL)	- moles of each neutral (aqueous phase)
Output(11+NSOL:	10+NSOL+NC)	- act. coeff. of each cation (aqueous phase)
Output(11+NSOL+NC:	10+NSOL+NC+NA)	- act. coeff. of each anion (aqueous phase)
Output(11+NSOL+NC+NA:	10+NSOL2)	- act. coeff. of each neutral (aqueous phase)
Output(11+NSOL2:	10+NSOL2+NC)	- moles of each cation (hydrophobic phase)
Output(11+NSOL2+NC:	10+NSOL2+NC+NA)	- moles of each anion (hydrophobic phase)
Output(11+NSOL2+NC+NA:	10+NSOL3)	- moles of each neutral (hydrophobic phase)
Output(11+NSOL3:	10+NSOL3+NC)	- act. coeff. of each cation (hydrophobic phase)
Output(11+NSOL3+NC:	10+NSOL3+NC+NA)	- act. coeff. of each anion (hydrophobic phase)
Output(11+NSOL3+NC+NA:	10+NSOL4)	- act. coeff. of each neutral hydrophobic phase)
Output(11+NSOL4:	10+NSOL4+NG)	- moles of each gas
Output(11+NSOL4+NG:	10+NSOL4+2*NG)	- equilibrium partial pressure of each gas (atm)
Output(11+NSOL4+2*NG:	10+NSOL4+2*NG+NS)	- moles of each solid
Output(11+NSOL4+2*NG+NS:	10+NSOL4+2*(NG+NS))	- volume of each solid (cm^3)
Output(11+NSOL4+2*(NG+NS):	10+NSOL4+2*NG+3*NS)	- saturation ratio of each solid

Notes: zero or a negative number will be assigned to quantities that are not relevant, such as properties of the hydrophobic phase where one does not exist, or quantities that could not be calculated (for example densities or surface tensions of chemical systems for which there are insufficient data).

9: **iSizeOutput** – INTEGER.

On entry: the size of real array **Output** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be ≥ minSizeOutput, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

Notes: a test of the size of the array **Output** is carried out in the *E-AIM* routine. This will stop, with an error message, if a larger size than the current value of **iSizeOutput** is needed.

10: **iOutputLabel(lenOutputlabel, iSizeOutput)** – INTEGER, a two dimensional array with a first dimension of size lenOuputLabel, and a second dimension of size iSizeOutput

On exit: each column **iOutputLabel(:,*i*)** contains an integer-coded label that describes the contents of the same element **Output(*i*)** of the results of the calculation. Values are assigned in EAIMstr to this argument when the routine is first called. On later calls it will only be re-assigned for iOptions(4) = 1.

Constraints: it is expected that all values are ≥0 and ≤127.

Details: these labels for the results contained in the **Output** array are encoded by their positions in the ASCII character table. This is done using the intrinsic Fortran function IACHAR, which is described in the Intel Fortran documentation as follows: *"returns the position of a character in the ASCII character set, even if the processor's default character set is different. In Intel Fortran, IACHAR is equivalent to the ICHAR function."* For example, the first output quantity is the relative humidity (RH), thus **iOutputLabel(1,1)** = 82, and **iOutputLabel(2,1)** = 72. The integer 82 is the position of upper case R in the ASCII table, and 72 is the position of upper case H. Thus this label is 'RH' when converted back to character form. The unused positions in the array are filled with blanks (position 32 in the ASCII table), thus **iOutputLabel(3:,*i*)** in this example are all set equal to 32.

Notes: this array of integer-coded labels is written only on the first call of the routine, or if **iOptions(4)** = 1. This integer coding is used because in some programming environments an array of strings (i.e., a two dimensional character array) is awkward to handle.

11: **lenOutputLabel** – INTEGER.

On entry: the first dimension of two dimensional integer array **iOutputLabel**, as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be ≥ minLenOutputLabel, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

12: **Messages** - CHARACTER, a string of length lenMessages.

On exit: contains error or warning messages relating to faults encountered in the main *E-AIM* routine, otherwise it will be blank.

Details: some testing of array sizes and input values is carried out in the *E-AIM* subroutine. If the tests are failed, the subroutine will return an explanatory message in this string (and no other results). The string should be checked after the *E-AIM* routine has been called.

Notes: where there is a failure, or an input error, that is detected within the *E-AIM* code for the chemical model an error message will be written to the file **eaim.err.** This is likely to be in the same directory as executable that calls the *E-AIM* subroutine (see section 3). Execution of the *E-AIM* code will then stop.

13: **lenMessages** – INTEGER.

On entry: the length of character string **Messages** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenMessages, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

14: **UsageRecord** - CHARACTER, a string of length lenUsageRecord.

On exit: a statement that includes the current date and time, and the time taken (in seconds) to execute the call of the *E-AIM* subroutine.

Details: the record is only written if the value **iOptions(5)** = 1 when the routine is called. Otherwise, **UsageRecord** will be blank. Here is a typical example record:

```
" E-AIMTr: dll(   )(   )     1        - 26-05-14 16:23:40 - Elapsed: 1.8720 - Organics:  2 -"
```

The first entry, E-AIMTr, refers to the model being used (Models I & II - E-AIMTr; Model III - E-AIM25; Model IV - E-AIMFr). The contents of the two sets of parentheses are intentionally blank, and the number “1” refers to the number of calculations carried out since the first call of the routine (thus, it is a counter). The date and time follow, and then the elapsed time (seconds) for the current call of the routine. The final value in the record is the number of organic compounds that have been included in the model system (2, in this particular example).

15: **lenUsageRecord** – INTEGER.

On entry: the length of character string **UsageRecord** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenUsageRecord, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 4.2.

16: **errorFlags** - CHARACTER, a string of length lenErrorFlags.

On exit: contains a set of single character flags indicating the success or failure of the *E-AIM* call, and the results of thermodynamic consistency tests.

Details: the first character in the string is always assigned a value, and the second character will be assigned a value if the chemical model has been called and returned a result. The remaining characters of the string are reserved for error flags which indicate failures of tests of the model result (which will be carried out if the value of **iOption(3)** is set to unity.

errorFlags(1:1) = '0' for a successful call of the *E-AIM* routine, and '1' for a failure in
which there is an error in one or more of the arguments (an array or string

set to the wrong size, for example). In this case the content of the string **Messages** should be written out, because this is likely to explain the error.

errorFlags(2:2) = the integer iFail, which is a flag set by the solver used to calculate equilibrium. A value of '0' indicates complete success. Experience shows that '6' and '1' also occur for correct answers. If iFail takes other values then the result is doubtful - please contact the author.

errorFlags(3:10) = if these characters are all blank then the result of the *E-AIM* calculation has passed a set of internal tests for thermodynamic consistency, and is likely to be correct. Failures of the internal tests are indicated by one or more of the following single character flags (which may appear in any order):

L - one or more variables in the calculation are within a small factor of their lower bounds (at the final result). Experience suggests that this means that there are errors in the calculated equilibria and therefore concentrations and/or partial pressures for some minor species.
W – water activity or relative humidity.
G - equilibrium between the gases and the liquid and/or solid phases;
S – saturation of the aqueous and/or hydrophobic phases with respect to one or more of the solids present.
H – the concentration of hydrogen ion in the liquid phase(s) is low enough that gas/liquid partitioning of NH_3 may not be accurately estimated (this applies in *E-AIM* calculations that do not include the dissociation of NH_4^+(aq) and water).
P – Gibbs' phase rule.
R – other reactions in and between the liquid phase(s).
A – calculated activities of organic compounds in the system are greater than unity, relative to the pure liquid reference state.

17: **lenErrorFlags** – INTEGER.

On entry: the length of character string **errorFlags** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenErrorFlags, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given below.

4.2 Internal EAIMstr parameters

The sizes of input and output arrays of the *E-AIM* subroutine are tested within the routine to make sure they are greater than or equal to certain lengths (for strings) and sizes (for arrays). The following parameters are defined in the source code of the *E-AIM* subroutine. They set minimum permitted lengths and sizes of the arguments of the routine. The values as currently set should be OK for most problems. They are used for testing the validity of the arguments.

minLenInput	: the length of the character string argument **Input** must be greater than or equal to this value.
minSizeOutput	: the number of elements of the double precision array argument **Output**, and the second dimension of the integer array **iOutputLabel**, must be greater than or equal to this value.
minLenOutputLabel	: the length of each element (i.e., the size of the first dimension) of the integer array argument **iOutputLabel** must be greater than or equal to this value.
minLenMessages	: the length of the character string argument **Messages** must be greater than or equal to this value.
minSizeiOutput	: the size of integer array argument **iOutput** must be greater than or equal to this value.
minLenUsageRecord	: the length of the character string argument **UsageRecord** must be greater than or equal to this value.
minSizeiOptions	: the size of the integer array argument **iOptions** must be greater than or equal to this value.
minLenErrorFlags	: the length of the character string argument **ErrorFlags** must be greater than or equal to this value.

The values of the above parameters that specify these minimum lengths (minLen…) and sizes (minSize…), are as follows:

```
   minLenInput        = 250,      minSizeiOutput     = 6,
   minSizeOutput      = 200,      minLenUsageRecord  = 150,
   minLenOutputLabel  = 50,       minSizeiOptions    = 5,
   minLenMessages     = 350,      minLenErrorFlags   = 10
```

4.3 An example call of EAIMstr

Here we describe a call of EAIMstr for a system containing 0.1 moles H_2SO_4, 1.0 moles $(NH_4)_2SO_4$, and 0.5 moles of NH_4NO_3 in a system of 1.0 m^3 volume at a total pressure of 1.0 atm. (These amounts of the salts are equivalent to 0.2 moles H^+, 2.5 moles NH_4^+, 1.1 moles SO_4^{2-}, and 0.5 moles NO_3^-). The ambient temperature is 15 ^oC (288.15 K), and the equilibrium relative humidity is fixed at 70% (0.7). The dissociation of water is switched on. The gases NH_3 and HNO_3 are allowed to partition into the gas phase, but not H_2SO_4. All solids can form. Model II is used for this calculation. When EAIMstr is called, the values of the input arguments should be as follows:

ModelNumber = 2

iOptions(1) = 0 or 1 (this option has no effect for EAIMstr, see EAIMvar description)

iOptions(2) = 0 or 1 (as above)

iOptions(3) = 1

iOptions(4) = 0 or 1 (the value has no effect on the first call)

iOptions(5) = 1

Input = “288.15 1.0 1.0   1 1   0.70   0.2 2.5 0. 1.1 0.5 0. 0. 0. 0.   0 0 0 3 0   0”

This completes the description of the problem. The remaining input arguments specify the length and size of the strings and arrays that describe the problem and the results. They are as follows: **iSizeiOptions** = 5, **lenInput** = 250, **iSizeiOutput** = 6, **iSizeOutput** = 200, **lenOutputLabel** = 50, **lenMessages** = 350, **lenUsageRecord** = 150, **lenErrorFlags** = 10. The assigned values are the minimum lengths and sizes allowed, and are listed at the end of the previous section.

This problem is the first one in the text file Test_str.dat, which is the input data file for the program Test_str.for that is included with this manual. The program is intended to demonstrate, in the simplest way possible, how to call the EAIMstr subroutine from the supplied dynamic link library.

After EAIMstr has been called successfully with the above input data, the values of the output variables are as given below. Note that the integer-coded labels (**iOutputLabel**) for each of the quantities in **Output** have been converted back to characters, and that quantities in **Output** which are zero are not listed. (When the test program Test_str is run, the output below will be found in the file Test_str.res.)

```
  iOutput(1) = 1        iOutput(4) = 2
  iOutput(2) = 3        iOutput(5) = 6
  iOutput(3) = 6        iOutput(6) = 30

  i    Output(i)    iOutputLabel(1:,i)(a)
  1    0.7000E+00   RH
  2    0.2881E+03   T
  3    0.1000E+01   P
  4    0.1000E+01   V
  5    0.1311E+01   aqueous phase density (g cm-3)
  6    0.1328E+03   aqueous phase volume (cm3)
  7    0.8755E+02   aqueous phase surface tension (mN m-1)
 11    0.1866E-01   nH(aq)
 12    0.1244E+01   nNH4(aq)
 14    0.1813E+00   nHSO4(aq)
 15    0.2905E+00   nSO4(aq)
 16    0.5000E+00   nNO3(aq)
 19    0.1038E-15   nOH(aq)
 20    0.4175E+01   nH2O(aq)
 21    0.4291E-09   nNH3(aq)
 22    0.1352E+01   fH(aq)
 23    0.4225E+00   fNH4(aq)
 25    0.4685E+00   fHSO4(aq)
 26    0.1894E-01   fSO4(aq)
 27    0.2276E+00   fNO3(aq)
 30    0.1609E+02   fOH(aq)
 31    0.1075E+01   fH2O(aq)
 32    0.1535E+01   fNH3(aq)
 55    0.5039E+00   nH2O(g)
 56    0.1286E-05    nHNO3(g)
 58    0.2492E-08    nNH3(g)
 61    0.1177E-01    pH2O(g) (atm.)
 62    0.3005E-07    pHNO3(g) (atm.)
 64    0.5823E-10    pNH3(g) (atm.)
 65    0.1285E-20    pH2SO4(g) (atm.)
 76    0.6282E+00    n(NH4)2SO4
106    0.4690E+02    vol. (NH4)2SO4 (cm^3)
129    0.2658E-08    ratio H2SO4.H2O
130    0.8601E-08    ratio H2SO4.2H2O
131    0.3122E-06    ratio H2SO4.3H2O
132    0.2717E-05    ratio H2SO4.4H2O
133    0.2666E-13    ratio H2SO4.6.5H2O
134    0.7221E-05    ratio HNO3.H2O
135    0.1132E-03    ratio HNO3.3H2O
136    0.1000E+01    ratio (NH4)2SO4
137    0.3061E+00    ratio (NH4)3H(SO4)2
138    0.2054E-01    ratio NH4HSO4
139    0.5294E+00    ratio NH4NO3
140    0.5366E+00    ratio 2NH4NO3.(NH4)2SO4
141    0.3501E+00    ratio 3NH4NO3.(NH4)2SO4
142    0.3247E-01    ratio NH4NO3.NH4HSO4
155    0.1176E-04    ratio HNO3.2H2O
```

UsageRecord:[(b)]

```
" E-AIMTr: dll(   )(   )     1        - 16-06-14 11:46:36 - Elapsed:   0.2930 - Organics:  0 -"
```

errorFlags:

```
"00          "
```

Messages:

```
""
```

Notes

(a) As stated above, the labels have been converted back from integer codes to characters. As an example, for output number 2 (temperature, T) the corresponding value of **iOutputLabel(1:,2)** (i.e., the second column of the integer array) is 84 followed by (**lenOutputLabel – 1**) instances of 32. The integer 84 is the position in the ASCII table of upper case letter T, and 32 is the position of the "blank" character in the table.

(b) If you repeat this example the date, time, and elapsed time reported in **UsageRecord** will of course differ from those here.

5. Specification of the *E-AIM* subroutine **EAIMvar**

This section describes the input and output arguments of EAIMvar. The colour codes used below are as follows: **red** – an argument that contains input data for the calculation which must be provided by the user; **blue** – an argument of INTENT(OUT) that contains the results of the calculation; **black** – an integer argument of INTENT(IN) which specifies the length of a string argument, or the size of an array argument. Note that in the specification below REAL (KIND=8) quantities are “double precision” (with approx. 15 digits of precision), and INTEGER (KIND=4) are 4 byte integers (which have a range of ±2147483648).

```
SUBROUTINE EAIMvar(ModelNumber, iOptions, iSizeiOptions,
                   T, P, V,
                   iWaterCase, iDissocOption, WaterValue,
                   InorganicMoles, iSizeInorgMoles,
                   iGasOptions, iSizeiGasOptions, nSolidOptions,
                   iSolidOptions, iSizeiSolidOptions,
                   nOrganicCmpnds, OrganicMoles, iSizeOrgMoles,
                   iOrganicOptions, iSizeiOrgOptions,
                   iOutput, iSizeiOutput, Output, iOutputLabel,
                   iSizeOutput, lenOutputLabel,
                   Messages, lenMessages, UsageRecord, lenUsageRecord,
                   errorFlags, lenErrorFlags)

 INTEGER (KIND=4), INTENT(IN) :: iSizeiOptions, iSizeInorgMoles,
                                 iSizeOrgMoles, iSizeiGasOptions,
                                 iSizeiSolidOptions, iSizeiOrgOptions, iSizeOutput,
                                 iSizeiOutput, lenOutputLabel, lenMessages,
                                 lenUsageRecord, lenErrorFlags
 INTEGER (KIND=4), INTENT(IN) :: ModelNumber, iOptions(iSizeiOptions),
                                 nOrganicCmpnds
 REAL (KIND=8),    INTENT(IN) :: T, P, V, WaterValue,
                                 InorganicMoles(iSizeInorgMoles)
 REAL (KIND=8)                :: OrganicMoles(iSizeOrgMoles)
 INTEGER (KIND=4), INTENT(IN) :: iWaterCase, iDissocOption,
                                 iGasOptions(iSizeiGasOptions), nSolidOptions
 INTEGER (KIND=4)             :: iSolidOptions(2,iSizeiSolidOptions),
                                 iOrganicOptions(iSizeiOrgOptions)
 REAL (KIND=8),    INTENT(OUT) :: Output(iSizeOutput)
 INTEGER (KIND=4), INTENT(OUT) :: iOutput(iSizeiOutput)
 INTEGER (KIND=4)              :: iOutputLabel(lenOutputLabel, iSizeOutput)
 CHARACTER (LEN=lenMessages),    INTENT(OUT) :: Messages
 CHARACTER (LEN=lenUsageRecord), INTENT(OUT) :: UsageRecord
 CHARACTER (LEN=lenErrorFlags),  INTENT(OUT) :: errorFlags
```

5.1 Arguments

1: **ModelNumber** - INTEGER.

On entry: the reference number of the *E-AIM* model to be used in the calculation.

Constraint: 1 $\le$ ModelNumber $\le$ 4.

Details: This scalar variable selects the *E-AIM* model to be used and must have one of the following values: 1 - Model I; 2 - Model II; 3 - Model III; 4 - *M*odel IV.

Notes: The value will be read when the routine is first called only, or when input argument **iOption(4)** = 1. (This argument is identical for both EAIMstr and EAIMvar.)

2: **iOptions(iSizeiOptions)** - INTEGER, an array of size iSizeiOptions.

On entry: sets options for the calculation of solution densities and surface tensions, whether thermodynamic consistency tests will be carried out on the results, and whether initialization of the model will be carried out on the current call of the routine. (This initialization is carried out automatically on the first call, irrespective of the option setting.)

Constraint: all values of the array must be either 0 (option off), or 1 (option on).

Details: this is an integer array that is used to specify options to control or include/exclude different elements of the calculation. (This argument is identical for both EAIMstr and EAIMvar.) Possible values are as follows:

iOptions(1) = 0, or 1. A value of 1 means that densities and volumes of the liquid phase(s) and solid(s) will be calculated, if possible. A value of 0 means that they will <u>not</u> be calculated. Any other value will be trapped as an error.

iOptions(2) = 0, or 1. A value of 0 means that the surface tension(s) of the liquid phase(s) and solid(s) will be calculated if possible. A value of 0 means that they will <u>not</u> be calculated. Any other value will be trapped as an error.

iOptions(3) = 0, or 1. A value of 0 means that a series of tests for the thermodynamic consistency of the result of the model calculation will *not* be carried out. This may result in slightly faster execution. A value of 1 means that the tests will be carried out, and any failures indicated in the argument **errorFlags** (see below).

iOptions(4) = 0, or 1. When the subroutine is first called, a series of initialisations are always carried out (these are necessary for the code to function correctly). A value of 0 means that on later calls these initialisations will not be carried out (this is how the model is expected to be run). A value of 1 means that the initializations <u>will</u> be done. This option may be useful for test purposes, or perhaps when calling in a programming environment in which data internal to routine **EAIMvar** are not saved for some reason.

iOptions(5) = 0, or 1. A value of 1 means that a usage record will be written to the string argument **UsageRecord** (see below for details). A value of zero means no record will be written, and **UsageRecord** will be blank.

3: **T** - REAL.

On entry: the temperature (K) at which the calculation will be carried out.

Constraints: the valid range is 180 to 330 K (for Models I, II and IV), or 298.15 K only (Model III).

4: **P** - REAL.

On entry: the system pressure (atm.).

Constraints: must be >0. It is recommended that this value is set at 1.0.

5: **V** - REAL.

On entry: the system volume (m^3).

Constraints: must be >0. The code has not been tested for very large or small values of **V**, which is set at 1.0 in the web-based version of *E-AIM*.

6: **iWaterCase** - INTEGER.

On entry: the way in which water is to be treated in the model (held in the condensed phase, specified as a fixed RH, or equilibrated between vapour and condensed phases).

Constraints: must be 1, 2, or 3.

Details: this argument and the associated **WaterValue** (below) should be considered together. **iWaterCase** = 1 means that the relative humidity of the system (specified as a fraction) is to be fixed to **WaterValue**. Alternative **iWaterCase** values are 2, for which **WaterValue** must be the total number of moles of water in the system. Here the program will solve for the equilibrium distribution of water between the vapour and condensed phases and will give the calculated relative humidity as an output. For **iWaterCase** = 3, **WaterValue** is again the total number of moles of water in the system, but in this case it is not allowed to partition into the vapour, and remains in the condensed phase as a liquid, ice, or water of hydration.

7: **iDissocOption** - INTEGER.

On entry: determines whether or not the dissociation of water in the liquid phase is included in the calculations.

Constraints: must be -1, 0, or 1.

Details: A value of -1 means that dissociation is not calculated for any input conditions. A value of zero means that if the numbers of moles of H^+ and OH^- are both entered as zero (see item 9 below), then H_2O dissociation in the aqueous phase will remain off, and neither ion will be a variable in the calculations. If a non-zero value of either H^+ or OH^- is entered, then the other quantity will be made a variable and water dissociation will be on. A value of 1 means that water dissociation will always be calculated, and both H^+(aq) and OH^-(aq) will be made variables in the calculation, even if their input amounts are zero.

8: **WaterValue** - REAL.

On entry: the number of moles of water present in the chemical system, or the ambient relative humidity (expressed as a fraction, not a percentage), according to the value of **iWaterCase**.

Constraints: for **iWaterCase** = 2 or 3, **WaterValue** must be > 0. For **iWaterCase** = 1 (fixed RH), 0.1 $\leq$ **WaterValue** $\leq$ 0.999.

Details: where **iWaterCase** is equal to 1, **WaterValue** is the equilibrium relative humidity in the chemical system, expressed as a fraction (thus, 80% RH is entered as 0.80). For **iWaterCase** equal to 2 or 3, **WaterValue** is the number of moles of water in the chemical system, either constrained to the condensed phase only (**iWaterCase** = 3), or allowed to partition between the condensed and vapour phases (**iWaterCase** = 2).

9: **InorganicMoles(iSizeInorgMoles)** - REAL, an array of size iSizeInorgMoles.

On entry: the numbers of moles of H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, Br^-, OH^- and NH_3 present in the system.

Constraints: all values must be $\geq$ 0.0, and the amounts of cations and anions must be charge balanced.

Details: the numbers of moles of each species must be present in elements 1:9 of the array in the order: H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, Br^-, OH^- and NH_3. The valid combinations of non-zero species for each model are: Model I - H^+, SO_4^{2-}, NO_3^-, Cl^-, and Br^-; Model II - H^+, NH_4^+, SO_4^{2-}, NO_3^-, OH^- and NH_3; Models III and IV - H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, OH^-, and NH_3.

10: **iSizeInorgMoles** – INTEGER.

On entry: the dimension of array **InorganicMoles** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeInorganicMoles, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

11: **iGasOptions(iSizeiGasOptions)** – INTEGER, an array of size iSizeiGasOptions.

On entry: flags that control the treatment of the inorganic gases HNO_3, HCl, NH_3, H_2SO_4, and HBr in the chemical system (if they can be formed).

Constraints: each value can be equal to 0, 3, or 4.

Details: the integer flags must be present in elements 1:5 of the array, and apply to the gases HNO_3, HCl, NH_3, H_2SO_4, and HBr, respectively. The flags control how the gas phase species are treated. Taking HNO_3(g) as an example, a value of '3' means that it is assumed not to occur and so all NO_3^- (and associated H^+) remain in the condensed phase(s). A value of '4' means, again, assume no formation of HNO_3(g) but report its equilibrium partial pressure over the liquid phase (if it exists, and also contains both H+ and NO_3^-). A value of '0' means that HNO_3(g) can be formed and will be partitioned between the phases if the two ions are present. The same applies to the other gases.

Notes: for systems containing NH_3 that are likely to contain a near-neutral or alkaline aqueous phase it is advisable to turn on the dissociation of water.

12: **iSizeiGasOptions** – INTEGER.

On entry: the size of array **iGasOptions** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiGasOptions, which is an internal parameter of the *E-AIM* subroutine. A table of the current values of these internal parameters is given in section 5.2.

13: **nSolidOptions** - INTEGER.

On entry: the number of inorganic solids for which options will be specified in the array **iSolidOptions** (see below).

Constraints: must be $\geq$0 and $\leq$30.

Details: it is possible to switch off the formation of individual solids (as many as needed) in the model in order to investigate the properties of supersaturated solutions. See the next entry for further details.

14: **iSolidOptions(2,iSizeiSolidOptions)** – INTEGER, an array with a first dimension of size 2, and a second dimension of size iSizeiSolidOptions

On entry: if **nSolidOptions** is >0, the element **iSolidOptions(1,*i*)** should contain the reference number of the solid whose option value is being set, and **iSolidOptions(2,*i*)** should contain the value of the option itself, for the *i*th solid for which an option is being set. If no solid options are being set then the array can be left blank.

Details: the reference numbers of the inorganic solids in the model are given in the list below:

Ref:	Solid	Ref:	Solid	Ref:	Solid
1	H2O(ice)	10	(NH4)2SO4	18	Na2SO4
2	H2SO4	11	(NH4)3H(SO4)2	19	Na2SO4.10H2O
3	H2SO4.H2O	12	NH4HSO4	20	Na3H(SO4)2
4	H2SO4.2H2O	13	NH4NO3	21	NaHSO4.H2O
5	H2SO4.3H2O	14	2NH4NO3.(NH4)2SO4	22	NaHSO4
6	H2SO4.4H2O	15	3NH4NO3.(NH4)2SO4	23	NaH3(SO4)2.H2O
7	H2SO4.6.5H2O	16	NH4NO3.NH4HSO4	24	Na2SO4.(NH4)2SO4.4H2O
8	HNO3.H2O	17	NH4Cl	25	NaNO3
9	HNO3.3H2O			26	NaNO3.Na2SO4.H2O
				27	NaCl
				28	HCl.3H2O
				29	HNO3.2H2O
				30	NaCl.2H2O

The permitted option values (i.e., possible values of **iSolidOptions(2,*i*)**) for each solid ***i*** are 3, which means exclude the solid from the calculation, or 4 which has the same meaning except that the degree of saturation of any liquid phase with respect to the solid will be calculated.

The easiest way to explain the use of these options is by example. Suppose we wish to switch off the formation of just one solid, $(NH_4)_2SO_4$, in a calculation. In this case **nSolidOptions** = 1, **iSolidOptions(1,1)** = 10 (the reference number of the solid in the table above, and

iSolidOptions(2,1) = 3 (or 4, if we want to know the saturation ratio). Next consider a second calculation in which we want to switch off the formation of both NH_4NO_3 and $(NH_4)_2SO_4$, and want to know the saturation ratio of the aqueous phase with respect to NH_4NO_3 but not $(NH_4)_2SO_4$. In this case the values to be entered are: **nSolidOptions** = 2, **iSolidOptions(1,1)** = 10, **iSolidOptions(2,1)** = 3, **iSolidOptions(1,2)** = 13 (the reference number of NH_4NO_3), and **iSolidOptions(2,2)** = 4. Solids can be entered in this list in any order.

15: **iSizeiSolidOptions** – INTEGER.

On entry: the size of the second dimension of array **iSolidOptions** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiSolidoptions, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

16: **nOrganicCmpnds** - INTEGER.

On entry: the number of organic compounds that are included in the chemical system (and which have properties defined in the file **eaim.org**).

Constraints: must be $\geq$0. If this number is >0 then the file eaim.org must be present and the number of organic compounds entered there must be the same as **nOrganicCmpnds**.

Details: if interested, please contact Simon Clegg. This document is complete only for the description of calculations for inorganic systems.

17: **OrganicMoles(iSizeOrgMoles)** – REAL, an array of size iSizeOrgMoles.

On entry: the numbers of moles of each of the **nOrganicCmpnds** organic compounds present in the chemical system.

Constraints: for each organic compound *i* that is present, **OrganicMoles(*i*)** must be $\geq$0.0. If **nOrganicCmpnds** = 0 this array can be left blank.

Details: if interested, please contact Simon Clegg. This document is complete only for the description of calculations for inorganic systems.

18: **iSizeOrgMoles** – INTEGER.

On entry: the size of array **OrganicMoles** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraints: must be $\geq$ minSizeOrganicMoles, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

19: **iOrganicOptions(iSizeiOrgOptions)** - INTEGER array, of size iSizeiOrgOptions.

On entry: this array must contain a set of options that determine how each organic compound in the system will be treated.

Constraints: permitted option values are 0, 3, and 4. In systems where there are no organic compounds (i.e., **nOrganicCmpnds** = 0) this array can be left blank.

Details: if interested, please contact Simon Clegg. This document is complete only for the description of calculations for inorganic systems.

20: **iSizeOrgOptions** – INTEGER.

On entry: the dimension of array **iOrganicOptions** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiOrganicOptions, which is a value set internally in the *E-AIM* subroutine. A list of the current values is given in section 5.2.

21: **iOutput(iSizeiOutput)** – INTEGER, an array of size iSizeiOutput.

On exit: the elements of this array contain information about the results of the chemical calculation.

Details: each element of the array contains the data listed below. (This argument is identical for both EAIMstr and EAIMvar.)

iOutput(1) : an integer from 1 to 3, meaning the following:

0 - the result contains neither an aqueous nor a hydrophobic liquid phase.

1 - the result contains an aqueous phase (but not a hydrophobic liquid phase).

2 - the result contains a hydrophobic liquid phase (but not an aqueous phase).

3 - the result contains both an aqueous phase and a hydrophobic liquid phase.

iOutput(2) : NC, the maximum number of cations in the system.

iOutput(3) : NA, the maximum number of anions in the system.

iOutput(4) : NN, the maximum number of neutral species in the system (including water).

iOutput(5) : NG, the maximum number of gases in the system.

iOutput(6) : NS, the maximum number of solids in the system.

22: **iSizeiOutput** – INTEGER.

On entry: the size of integer array **iOutput** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minSizeiOutput, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

23: **Output(iSizeOutput) –** REAL, an array of size iSizeOutput.

On exit: contains the results of the calculation.

Details: this array will contain the results of the *E-AIM* calculation. The quantities contained in each of the elements of array **Output** are given below. (This argument is identical for both EAIMstr and EAIMvar.) See also the output array **iOutputLabels**, in which integer-coded labels corresponding to the output quantities are placed.

Note the following definitions used below: NSOL = NC + NA + NN
NSOL2 = 2 * NSOL
NSOL3 = 3 * NSOL
NSOL4 = 4 * NSOL

Output(1)	- RH, the relative humidity (as a fraction)
Output(2)	- T, temperature (K)
Output(3)	- P, system pressure (atm.)
Output(4)	- V, system volume (m^3)
Output(5)	- density of the aqueous phase (g cm^{-3})
Output(6)	- volume of the aqueous phase (cm^3)
Output(7)	- surface tension of the aqueous phase (mN m^{-1})
Output(8)	- density of the hydrophobic phase (g cm^{-3})
Output(9)	- volume of the hydrophobic phase (cm^3)
Output(10)	- surface tension of the hydrophobic phase (mN m^{-1})

Output(11:	10+NC)	- moles of each cation (aqueous phase)
Output(11+NC:	10+NC+NA)	- moles of each anion (aqueous phase)
Output(11+NC+NA:	10+NSOL)	- moles of each neutral (aqueous phase)
Output(11+NSOL:	10+NSOL+NC)	- act. coeff. of each cation (aqueous phase)
Output(11+NSOL+NC:	10+NSOL+NC+NA)	- act. coeff. of each anion (aqueous phase)
Output(11+NSOL+NC+NA:	10+NSOL2)	- act. coeff. of each neutral (aqueous phase)
Output(11+NSOL2:	10+NSOL2+NC)	- moles of each cation (hydrophobic phase)
Output(11+NSOL2+NC:	10+NSOL2+NC+NA)	- moles of each anion (hydrophobic phase)
Output(11+NSOL2+NC+NA:	10+NSOL3)	- moles of each neutral (hydrophobic phase)
Output(11+NSOL3:	10+NSOL3+NC)	- act. coeff. of each cation (hydrophobic phase)
Output(11+NSOL3+NC:	10+NSOL3+NC+NA)	- act. coeff. of each anion (hydrophobic phase)
Output(11+NSOL3+NC+NA:	10+NSOL4)	- act. coeff. of each neutral hydrophobic phase)
Output(11+NSOL4:	10+NSOL4+NG)	- moles of each gas
Output(11+NSOL4+NG:	10+NSOL4+2*NG)	- equilibrium partial pressure of each gas (atm)
Output(11+NSOL4+2*NG:	10+NSOL4+2*NG+NS)	- moles of each solid
Output(11+NSOL4+2*NG+NS:	10+NSOL4+2*(NG+NS))	- volume of each solid (cm^3)
Output(11+NSOL4+2*(NG+NS):	10+NSOL4+2*NG+3*NS)	- saturation ratio of each solid

Notes: zero or a negative number will be assigned to quantities that are not relevant, such as properties of the hydrophobic phase where one does not exist, or quantities that could not be calculated (for example densities or surface tensions of chemical systems for which there are insufficient data).

24: **iSizeOutput** – INTEGER.

On entry: the dimension of real array **Output** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be ≥ minSizeOutput, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

Notes: a further test of the size of the array Output is carried out in the *E-AIM* routine, which will stop with an error message if a larger size than the current value of **iSizeOutput** is needed.

25: **iOutputLabel(lenOutputlabel, iSizeOutput)** – INTEGER, a two dimensional array with a first dimension of size lenOuputLabel, and a second dimension of size iSizeOutput.

On exit: each column **iOutputLabel(:,*i*)** contains an integer-coded label that describes the contents of the same element **Output(*i*)** of the results of the calculation. Values are assigned in EAIMvar to this argument when the routine is first called. On later calls it will only be re-assigned for iOptions(4) = 1.

Constraints: it is expected that all values are ≥0 and ≤127.

Details: these labels for the results contained in the **Output** array are encoded by their positions in the ASCII character table. This is done using the intrinsic Fortran subroutine IACHAR, which is described in the Intel Fortran documentation as follows: *"returns the position of a character in the ASCII character set, even if the processor's default character set is different. In Intel Fortran, IACHAR is equivalent to the ICHAR function."* For example, the first output quantity is the relative humidity (RH), thus **iOutputLabel(1,1)** = 82, and **iOutputLabel(2,1)** = 72. The integer 82 is the position of upper case R in the ASCII table, and 72 is the position of upper case H. Thus this label is 'RH' when converted back to character form. The unused positions in the array are filled with blanks (position 32 in the ASCII table), thus **iOutputLabel(3:,*i*)** in this example are all set equal to 32. (This argument is identical for both EAIMstr and EAIMvar.)

Notes: this array of integer-coded labels is written only on the first call of the routine, or if **iOptions(4)** = 1. This integer coding is used because in some programming environments an array of strings (i.e., a two dimensional character array) is awkward to handle.

26: **lenOutputLabel** – INTEGER.

On entry: the first dimension of two dimensional integer array **iOutputLabel**, as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be ≥ minLenOutputLabel, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

27: **Messages** - CHARACTER, a string of length lenMessages.

On exit: contains error or warning messages relating to faults encountered in the main *E-AIM* routine, otherwise it will be blank.

Details: some testing of array sizes and input values is carried out in the *E-AIM* subroutine. If the tests are failed, the subroutine will return an explanatory message in this string (and no other results). The string should be checked after the *E-AIM* routine has been called. (This argument is identical for both EAIMstr and EAIMvar.)

Notes: where there is a failure, or an input error, that is detected within the code for the chemical model an error message will be written to the **eaim.err** file. This is likely to be in the same directory as the executable that calls *E-AIM* (see section 3). Execution of the *E-AIM* code will then stop, without any other warnings or messages.

28: **lenMessages** – INTEGER.

On entry: the length of character string **Messages** as declared in the (sub)program from which the E-AIM subroutine is called.

Constraint: must be $\geq$ minLenMessages, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

29: **UsageRecord** - CHARACTER, a string of length lenUsageRecord.

On exit: a statement that includes the current date and time, and the time taken (in seconds) to execute the call of the *E-AIM* subroutine.

Details: the record is only written if the value **iOptions(5)** = 1 when the routine is called. Otherwise, **UsageRecord** will be blank. Here is a typical example record:

```
" E-AIMTr: dll(   )(   )     1        - 26-05-14 16:23:40 - Elapsed: 1.8720 - Organics:  2 -"
```

The first entry, E-AIMTr, refers to the model being used (Models I & II - E-AIMTr; Model III - E-AIM25; Model IV - E-AIMFr). The contents of the two sets of parentheses are intentionally blank, and the number "1" refers to the number of calculations carried out since the first call of the routine (it is a counter). The date and time follow, and then the elapsed time (seconds) for the current call of the routine. The final number in the record is the number of organic compounds that have been included in the model system (2, in this particular example). (This argument is identical for both EAIMstr and EAIMvar.)

31: **lenUsageRecord** – INTEGER.

On entry: the length of character string **UsageRecord** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenUsageRecord, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

32: **errorFlags** - CHARACTER, a string of length lenErrorFlags.

On exit: contains a set of single character flags indicating the success or failure of the *E-AIM* call, and the results of any thermodynamic consistency tests.

Details: the first character in the string is always assigned a value, and the second character will be assigned a value if the chemical model has been called and returned a result. The remaining characters of the string are reserved for error flags which indicate failures of tests of the model result (which will be carried out if the value of argument **iOption(3)** is set to unity. (This argument is identical for both EAIMstr and EAIMvar.)

errorFlags(1:1) = '0' for a successful call of the *E-AIM* routine, and '1' for a failure in which there is an error in one or more of the arguments (an array or string set to the wrong size, for example). In this case the content of the string **Messages** should be written out, because this is likely to explain the error.

errorFlags(2:2) = the integer iFail, which is a flag set by the solver used to calculate equilibrium. A value of '0' indicates complete success. Experience shows that '6' and '1' also occur for correct answers. If iFail takes other values then the result is doubtful - please contact the author.

errorFlags(3:10) = if these characters are all blank then the result of the *E-AIM* calculation has passed a set of internal tests for thermodynamic consistency, and is likely to be correct. Failures of the internal tests are indicated by one or more of the following single character flags (which may appear in any order):

L - one or more variables in the calculation are within a small factor of their lower bounds (at the final result). Experience suggests that this may mean that there are errors in the calculated equilibria (hence concentrations and/or partial pressures for some minor species.

W – water activity or relative humidity;

G - equilibrium between the gases and the liquid and/or solid phases;

S – saturation of the aqueous and/or hydrophobic phases with respect to one or more of the solids present;

H – the concentration of hydrogen ion in the liquid phase(s) is low enough that gas/liquid partitioning of NH_3 may not be accurately estimated (this applies to the *E-AIM* models that do not include the dissociation of NH_4^+(aq) and water);

P – Gibbs' phase rule;

R – other reactions in and between the liquid phase(s);

A – calculated activities of organic compounds in the system are greater than unity, relative to the pure liquid reference state.

33: **lenErrorFlags** – INTEGER.

On entry: the length of character string **ErrorFlags** as declared in the (sub)program from which the *E-AIM* subroutine is called.

Constraint: must be $\geq$ minLenErrorFlags, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given in section 5.2.

5.2 Internal EAIMvar parameters

The sizes of input and output arrays of the *E-AIM* subroutine are tested within the routine to make sure they are greater than or equal to certain lengths (for strings) and sizes (for arrays). The following parameters are defined in the source code of each *E-AIM* subroutine. They set minimum permitted lengths and sizes of the arguments of the routine. The values as currently set should be OK for most problems. They are only used for testing the validity of the arguments, and not their contents.

minSizeInorganicMoles	= NCmaxBase + NAmaxBase + (NNmaxExt-1). These three parameters have values of 3, 6, and 2. Thus, minSizeInorganicMoles = 10.
minSizeOrganicMoles	= 1, or the value of argument nOrganicCmpnds, whichever is greater.
minSizeiGasOptions	= NGmaxBase-1. The value of this parameter is 6, thus minSizeiGasOptions = 5
minSizeiSolidOptions_2	= NSmaxBase. The value of this parameter is 30.
minSizeiOrganicOptions	= 6*minSizeOrganicMoles, thus minSizeiOrganicOptions is equal to the greater of 6 and 6*nOrganicCmpnds.
minSizeOutput	: the number of elements of the double precision array argument **Output**, must be greater than or equal to this value.
minLenOutputLabel	: the length of each element (i.e., the size of the first dimension) of the integer array argument **iOutputLabel** must be greater than or equal to this value.
minLenMessages	: the length of the character string argument **Messages** must be greater than or equal to this value.
minSizeiOutput	: the size of integer array argument **iOutput** must be greater than or equal to this value.
minLenUsageRecord	: the length of the character string argument **UsageRecord** must be greater than or equal to this value.
minSizeIOptions	: the size of the integer array argument **iOptions** must be greater than or equal to this value.
minLenErrorFlags	: the length of the character string argument **errorFlags** must be greater than or equal to this value.

The values of the above parameters that set these minimum lengths (minLen…) and sizes (minSize…), are as follows:

```
   minSizeiOptions   = 5
   minSizeInorganicMoles  = 10, (calculated as NCmaxBase + NAmaxBase + (NNmaxBase - 1))
   minSizeiGasOptions     = 5, (calculated as NGmaxBase - 1)
   minSizeiSolidOptions_2 = 30 (equal to NSmaxBase)
   minSizeOrganicMoles    = MAX(1, nOrganicCmpnds)
   minSizeiOrganicOptions = MAX(6, 6*minSizeOrganicMoles)
   minSizeOutput     = 200
   minSizeiOutput    = 6
   minLenOutputLabel = 50
   minLenMessages    = 350
   minLenUsageRecord = 150
   minLenErrorFlags  = 10
```

5.3 An example call of EAIMvar

Here we describe a call of EAIMvar for the same system described in section 4.3 (for EAIMstr): 0.1 moles H_2SO_4, 1.0 moles $(NH_4)_2SO_4$, and 0.5 moles of NH_4NO_3 in a system of 1.0 m^3 volume at a total pressure of 1.0 atm. (These amounts of the salts are equivalent to 0.2 moles H^+, 2.5 moles NH_4^+, 1.1 moles SO_4^{2-}, and 0.5 moles NO_3^-). The ambient temperature is 15 ^oC (288.15 K), and the equilibrium relative humidity is fixed at 70% (0.7). The dissociation of water is switched on. The gases NH_3 and HNO_3 are allowed to partition into the gas phase, but not H_2SO_4. All solids can form. Model II is used for this calculation. When EAIMvar is called, the values of the input arguments should be as follows:

ModelNumber = 2

iOptions(1) = 0 or 1 (controls calculation of densities)

iOptions(2) = 0 or 1 (controls calculation of surface tensions)

iOptions(3) = 1

iOptions(4) = 0 or 1 (the value has no effect on the first call)

iOptions(5) = 1

T = 288.15, **P** = 1.0, **V** = 1.0 (temperature, pressure, and volume of the system)

iWaterCase = 1, **iDissocOption** = 1, **WaterValue** = 0.70

InorganicMoles(1:9) = 0.2, 2.5, 0.0, 1.1, 0.5, 0.0, 0.0, 0.0, 0.0

iGasOptions(1:5) = 0, 0, 0, 3, 0

nSolidOptions = 0

 (array **iSolidOptions(:,:)** can be left undefined in this case because nSolidOptions = 0)

nOrganicCmpnds = 0

 (array **iOrganicOptions(:)** can be left undefined in this case because nOrganicCmpnds = 0)

The real array **InorganicMoles** contains the numbers of moles of the following species (in this order): H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, Br^-, OH^- and NH_3. The integer array **iGasOptions** contains the options values for the following gases (in this order): HNO_3, HCl, NH_3, H_2SO_4, and HBr.

This completes the description of the problem. The remaining input arguments specify the length and size of the strings and arrays that describe the problem and the results. They are as follows: **iSizeiOptions** = 5, **lenInput** = 250, **iSizeiOutput** = 6, **iSizeOutput** = 200, **lenOutputLabel** = 50, **lenMessages** = 350, **lenUsageRecord** = 150, **lenErrorFlags** = 10. The assigned values are the minimum lengths and sizes allowed, and are listed at the end of the previous section.

This problem is the first one in the text file Test_var.dat, which is the input data file for the program Test_var.for that is included with this manual. The program is intended to demonstrate, in the simplest way possible, how to call the EAIMvar subroutine from the supplied dynamic link library (see Appendix 2 for a description).

After EAIMvar has been called successfully with the above input data, the values of the output variables will be the same as for the example for EAIMstr described in section 4.3. Note that the integer-coded labels (**iOutputLabel**) for each of the quantities in **Output** have been converted back to characters, and that quantities in **Output** which are zero are not listed. (When the test program Test_var is run, the output will be found in the file Test_var.res.)

6. Module versions of the *E-AIM* subroutines

The use of a dynamic link library in which the two routines EAIMstr and EAIMvar are contained in a Fortran module enables fewer arguments to be used for both routines: essentially just those needed for input and output of data, and not those that specify lengths and sizes of the arguments. However, the resulting dll is less portable because it requires the user to compile his/her program with the same Fortran compiler used to produce the dynamic link library.

This section describes the EAIMstr and EAIMvar routines in the “module” version of the dll. Test programs that call this dll, which is named EAIM_module_q64.dll, are described in Appendix 3.

6.1 Specification of the *E-AIM* subroutine EAIMstr (module version)

This section describes the input and output arguments of the module version of EAIMstr. The colour codes used below are as follows: **red** – an argument that contains input data for the calculation which must be provided by the user; **blue** – an argument of INTENT(OUT) that contains the results of the calculation. Note that in the specification below REAL (KIND=8) quantities are “double precision” (with approx. 15 digits of precision), and INTEGER (KIND=4) are 4 byte integers (which have a range of ±2147483648).

```
SUBROUTINE EAIMstr(ModelNumber, iOptions, Input,
                   iOutput, Output, OutputLabel, Messages,
                   UsageRecord, ErrorFlags)

INTEGER (KIND=4),  INTENT(IN) :: ModelNumber, iOptions(:)
CHARACTER (LEN=*), INTENT(IN) :: Input
INTEGER (KIND=4),  INTENT(OUT) :: iOutput(:)
REAL (KIND=8),     INTENT(OUT) :: Output(:)
CHARACTER (LEN=*)              :: OutputLabel(:)
CHARACTER (LEN=*), INTENT(OUT) :: Messages, UsageRecord, errorFlags
```

6.1.1 Arguments

1: **ModelNumber** - INTEGER.

On entry: the reference number of the *E-AIM* model to be used in the calculation.

Constraint: 1 $\leq$ ModelNumber $\leq$ 4.

Details: this scalar variable selects the *E-AIM* model to be used and must have one of the following values: 1 - Model I; 2 - Model II; 3 - Model III; 4 - Model IV.

Notes: the value will be read when the routine is first called only, or when input argument **iOption(4)** = 1.

2: **iOptions(:)** - INTEGER, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeiOptions, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.1.2.

On entry: sets options for the calculation of solution densities and surface tensions, whether thermodynamic consistency tests will be carried out on the results, and whether initialization of the model will be carried out on the current call of the routine. (This initialization is carried out automatically on the first call, irrespective of the option setting.)

Constraint: all values of the array must be either 0 (option off), or 1 (option on).

Details: this is an integer array that is used to specify options to control or include/exclude different elements of the calculation. Possible values are the same as for the non-module version of EAIMstr and are described in section 4.1.

3: **Input** - CHARACTER (LEN=*), a string of arbitrary length. This length must be must be $\geq$ minLenInput, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.1.2.

On entry: contains the input data for the problem to be solved.

Details: the data for each problem are specified in the same way as for *Batch* input for the web-based model. This is described in Appendix 1 of this document, and on the following web page: http://www.aim.env.uea.ac.uk/aim/model2/input2d.html. For chemical systems containing only inorganic compounds, only items **a – u** from the descriptions should be entered into **Input**. (If your system contains organic compounds then additional files will be needed to run the model. please contact Simon Clegg.)

4: **iOutput(:)** – INTEGER, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeiOutput, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.1.2.

On exit: the elements of this array contain information about the results of the chemical calculation.

Details: each element of the array contains the same data as for the non-module version of EAIMstr, which is described in section 4.1.

5: **Output(:) –** REAL, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeOutput, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.1.2.

On exit: contains the results of the calculation.

Details: this array will contain the results of the *E-AIM* calculation. See also the output array **OutputLabels**, in which labels describing each output quantity are placed. The contents of array **Output** are the same as for the non-module version of EAIMstr, and are described in section 4.1.

6: **OutputLabel(:)** – CHARACTER (LEN=*), a one dimensional assumed shape character array. The size of the array must be $\geq$ minSizeOutput, which is an internal parameter of the subroutine. The length of each element of the array must be $\geq$ minLenOutputLabel, which is also an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.1.2.

On exit: each element **OutputLabel(*i*)** contains a text label that describes the contents of the same element **Output(*i*)** of the results of the calculation. Values are assigned in EAIMstr to this argument when the routine is first called. On later calls it will only be re-assigned for iOptions(4) = 1.

Constraints: none.

Details: this array of labels is written only on the first call of the routine, or if **iOptions(4)** = 1.

7: **Messages** - CHARACTER (LEN=*), a string of arbitrary length. This length must be $\geq$ minLenMessages, which is an internal parameter of the subroutine. A list of the current values of these internal parameters is given at the end of section 6.1.2.

On exit: contains error or warning messages relating to faults encountered in the main *E-AIM* routine, otherwise it will be blank.

Details: some testing of array sizes and input values is carried out in the *E-AIM* subroutine. If the tests are failed, the subroutine will return an explanatory message in this string (and no other results). The string should be checked after the *E-AIM* routine has been called.

Notes: where there is a failure, or an input error, that is detected within the *E-AIM* code for the chemical model an error message will be written to the file **eaim.err** that is likely to be in the same directory as the executable program that calls the routine (see section 3). Execution of the *E-AIM* code will then stop.

8: **UsageRecord** - CHARACTER (LEN=*), a string of arbitrary length. This length must be $\geq$ minLenUsageRecord, which is an internal parameter of the subroutine. A list of the current values of these internal parameters is at the end of section 6.1.2.

On exit: a statement that includes the current date and time, and the time taken (in seconds) to execute the call of the *E-AIM* subroutine.

Details: the record is only written if the value **iOptions(5)** = 1 when the routine is called. It is the same as for the non-module version of EAIMstr and is described in section 4.1.

9: **errorFlags** – CHARACTER (LEN=*), a string of arbitrary length. This length must be ≥ minLenErrorFlags, which is an internal parameter of the subroutine. A list of the current values of these internal parameters is at the end of section 6.1.2.

On exit: contains a set of single character flags indicating the success or failure of the *E-AIM* call, and the results of thermodynamic consistency tests.

Details: the first character in the string is always assigned a value, and the second character will be assigned a value if the chemical model has been called and returned a result. The remaining characters of the string are reserved for error flags which indicate failures of tests of the model result (which will be carried out if the value of **iOption(3)** is set to unity. The flag values are the same as for the non-module version of EAIMstr, and are described in section 4.1.

6.1.2 Internal EAIMstr parameters

The sizes of input and output arrays of the subroutine are tested internally to make sure they are greater than or equal to certain lengths (for strings) and sizes (for arrays). The following parameters are defined in the source code of the module version of EAIMstr. They set minimum permitted lengths and sizes of the arguments of the routine. The values as currently set should be OK for most problems. They are only used for testing the validity of the arguments.

The parameters listed below are the same as those for the non-module version of EAIMstr (see section 4.2), with the exceptions that minSizeOutput and minLenOutputLabel now refer to the character array OutputLabel (which replaces the integer array iOutputLabel in the non-module version of the routine).

minLenInput	: the length of the character string argument **Input** must be greater than or equal to this value.
minSizeOutput	: the number of elements of the double precision array argument **Output,** and the character array **OutputLabel**, must be greater than or equal to this value.
minLenOutputLabel	: the length of each element of the character array argument **OutputLabel** must be greater than or equal to this value.
minLenMessages	: the length of the character string argument **Messages** must be greater than or equal to this value.
minSizeiOutput	: the size of integer array argument **iOutput** must be greater than or equal to this value.
minLenUsageRecord	: the length of the character string argument **UsageRecord** must be greater than or equal to this value.
minSizeiOptions	: the size of the integer array argument **iOptions** must be greater than or equal to this value.
minLenErrorFlags	: the length of the character string argument **ErrorFlags** must be greater than

or equal to this value.

The values of the above parameters that specify these minimum lengths (minLen…) and sizes (minSize…), are as follows:

```
   minLenInput        = 250,      minSizeiOutput     = 6,
   minSizeOutput      = 200,      minLenUsageRecord  = 150,
   minLenOutputLabel  = 50,       minSizeiOptions    = 5,
   minLenMessages     = 350,      minLenErrorFlags   = 10
```

6.1.3 Calling EAIMstr

No example is given here, because the specification of the problem, and the results, are essentially identical to those for the non-module version of EAIMstr. See Appendix 3 for a description of the test program for this routine, and coding requirements.

6.2 Specification of the subroutine EAIMvar (module version)

This section describes the input and output arguments of the module version of EAIMvar. The colour codes used below are as follows: **red** – an argument that contains input data for the calculation which must be provided by the user; **blue** – an argument of INTENT(OUT) that contains the results of the calculation. Note that in the specification below REAL (KIND=8) quantities are "double precision" (with approx. 15 digits of precision), and INTEGER (KIND=4) are 4 byte integers (which have a range of ±2147483648).

```
SUBROUTINE EAIMvar(ModelNumber, iOptions, T, P, V,
                   iWaterCase, iDissocOption, WaterValue,
                   InorganicMoles, iGasOptions, nSolidOptions, iSolidOptions,
                   nOrganicCmpnds, OrganicMoles, iOrganicOptions,
                   iOutput, Output, OutputLabel,
                   Messages, UsageRecord, errorFlags)

INTEGER (KIND=4), INTENT(IN) :: ModelNumber, iOptions(:), nOrganicCmpnds
REAL (KIND=8),    INTENT(IN) :: T, P, V, WaterValue, InorganicMoles(:)
REAL (KIND=8)                :: OrganicMoles(:)
INTEGER (KIND=4), INTENT(IN) :: iWaterCase, iDissocOption, iGasOptions(:),
                                nSolidOptions
INTEGER (KIND=4)              :: iSolidOptions(:,:), iOrganicOptions(:)
REAL (KIND=8),    INTENT(OUT) :: Output(:)
INTEGER (KIND=4), INTENT(OUT) :: iOutput(:)
CHARACTER (LEN=*)             :: OutputLabel(:)
CHARACTER (LEN=*), INTENT(OUT):: Messages, UsageRecord, errorFlags
```

6.2.1 Arguments

1: **ModelNumber** - INTEGER.

On entry: the reference number of the *E-AIM* model to be used in the calculation.

Constraint: 1 $\leq$ ModelNumber $\leq$ 4.

Details: This scalar variable selects the *E-AIM* model to be used and must have one of the following values: 1 - Model I; 2 - Model II; 3 - Model III; 4 - *M*odel IV.

Notes: The value will be read when the routine is first called only, or when input argument **iOption(4)** = 1.

2: **iOptions(:)** – INTEGER, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeiOptions, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On entry: sets options for the calculation of solution densities and surface tensions, whether thermodynamic consistency tests will be carried out on the results, and whether initialization of the model will be carried out on the current call of the routine. (This initialization is carried out automatically on the first call, irrespective of the option setting.)

Constraint: all values of the array must be either 0 (option off), or 1 (option on).

Details: this is an integer array that is used to specify options to control or include/exclude different elements of the calculation. The possible values are the same as for the non-module version of EAIMvar, and are described in section 5.1.

3: **T** - REAL.

On entry: the temperature (K) at which the calculation will be carried out.

Constraints: the valid range is 180 to 330 K (for Models I, II and IV), or 298.15 K only (Model III).

4: **P** - REAL.

On entry: the system pressure (atm.).

Constraints: must be >0. It is recommended that this value is set at 1.0.

5: **V** - REAL.

On entry: the system volume (m^3).

Constraints: must be >0. The code has not been tested for very large or small values of **V**, which is set at 1.0 m^3 in the web-based version of *E-AIM*.

6: **iWaterCase** - INTEGER.

On entry: the way in which water is to be treated in the model (held in the condensed phase, specified as a fixed RH, or equilibrated between vapour and condensed phases).

Constraints: must be 1, 2, or 3.

Details: this argument and the associated **WaterValue** (below) should be considered together. **iWaterCase** = 1 means that the relative humidity of the system (specified as a fraction) is to be fixed to **WaterValue**. Alternative **iWaterCase** values are 2, for which **WaterValue** must be the total number of moles of water in the system. Here the program will solve for the equilibrium distribution of water between the vapour and condensed phases and will give the calculated relative humidity as an output. For **iWaterCase** = 3, **WaterValue** is again the total number of moles of water in the system, but in this case it is not allowed to partition into the vapour, and remains in the condensed phase as a liquid, ice, or water of hydration.

7: **iDissocOption** - INTEGER.

On entry: determines whether or not the dissociation of water in the liquid phase is included in the calculations.

Constraints: must be -1, 0, or 1.

Details: A value of -1 means that dissociation is not calculated for any input conditions. A value of zero means that if the numbers of moles of H^+ and OH^- are both entered as zero (see item 9 below), then H_2O dissociation in the aqueous phase will remain off, and neither ion will be a variable in the calculations. If a non-zero value of either H^+ or OH^- is entered, then the other quantity will be made a variable and water dissociation will be on. A value of 1 means that water dissociation will always be calculated, and both H^+(aq) and OH^-(aq) will be made variables in the calculation, even if their input amounts are zero.

8: **WaterValue** - REAL.

On entry: the number of moles of water present in the chemical system, or the ambient relative humidity (expressed as a fraction, not a percentage), according to the value of **iWaterCase**.

Constraints: for **iWaterCase** = 2 or 3, **WaterValue** must be > 0. For **iWaterCase** = 1 (fixed RH), 0.1 $\leq$ **WaterValue** $\leq$ 0.999.

Details: where **iWaterCase** is equal to 1, **WaterValue** is the equilibrium relative humidity in the chemical system, expressed as a fraction (thus, 80% RH is entered as 0.80). For **iWaterCase** equal to 2 or 3, **WaterValue** is the number of moles of water in the chemical system, either constrained to the condensed phase only (**iWaterCase** = 3), or allowed to partition between the condensed and vapour phases (**iWaterCase** = 2).

9: **InorganicMoles(:)** – REAL, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeInorganicMoles, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On entry: the numbers of moles of H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, Br^-, OH^- and NH_3 present in the system.

Constraints: all values must be $\geq$ 0.0, and the amounts of cations and anions must be charge balanced.

Details: the numbers of moles of each species must be present in elements 1:9 of the array in the order: H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, Br^-, OH^- and NH_3. The valid combinations of non-zero species for each model are: Model I - H^+, SO_4^{2-}, NO_3^-, Cl^-, and Br^-; Model II - H^+, NH_4^+, SO_4^{2-}, NO_3^-, OH^- and NH_3; Models III and IV - H^+, NH_4^+, Na^+, SO_4^{2-}, NO_3^-, Cl^-, OH^-, and NH_3.

10: **iGasOptions(:)** – INTEGER, a one dimensional assumed shape array. The size of the array must be $\geq$ minSizeiGasOptions, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On entry: flags that control the treatment of the inorganic gases HNO_3, HCl, NH_3, H_2SO_4, and HBr in the chemical system (if they can be formed).

Constraints: each value can be equal to 0, 3, or 4.

Details: the integer flags must be present in elements 1:5 of the array, and apply to the gases HNO_3, HCl, NH_3, H_2SO_4, and HBr, respectively. The flags control how the gas phase species are treated. Taking HNO_3(g) as an example, a value of '3' means that it is assumed not to occur and so all NO_3^- (and associated H^+) remain in the condensed phase(s). A value of '4' means, again, assume no formation of HNO_3(g) but report its equilibrium partial pressure over the liquid phase (if it exists, and also contains both H+ and NO_3^-). A value of '0' means that HNO_3(g) can be formed and will be partitioned between the phases if the two ions are present. The same applies to the other gases.

Notes: for systems containing NH_3 that are likely to contain a near-neutral or alkaline aqueous phase it is advisable to turn on the dissociation of water.

11: **nSolidOptions** - INTEGER.

On entry: the number of inorganic solids for which options will be specified in the array **iSolidOptions** (see below).

Constraints: must be $\geq$0 and $\leq$30.

Details: it is possible to switch off the formation of individual solids (as many as needed) in the model in order to investigate the properties of supersaturated solutions. See the next entry for further details.

12: **iSolidOptions(2,:)** – INTEGER, a two dimensional assumed shape array. The first dimension must be of size 2, and a second dimension of size $\geq$ minSizeiSolidOptions_2, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On entry: if **nSolidOptions** is >0, the element **iSolidOptions(1,*i*)** should contain the reference number of the solid whose option value is being set, and **iSolidOptions(2,*i*)** should contain the value of the option itself, for the *i*th solid for which an option is being set. If no solid options are being set then the array can be left blank.

Details: for the reference numbers of the inorganic solids in the model, and examples of the use of this option, see section 5.1.

13: **nOrganicCmpnds** - INTEGER.

On entry: the number of organic compounds that are included in the chemical system (and which have properties defined in the file **eaim.org**).

Constraints: must be ≥0. If this number is >0 then the file **eaim.org** must be present and the number of organic compounds entered there must be the same as **nOrganicCmpnds**.

Details: if interested, please contact Simon Clegg. This document is complete only for the description of calculations for inorganic systems.

14: **OrganicMoles(:)** – REAL, a one dimensional assumed shape array. If nOrganicCmpnds = 0, then the size of the OrganicMoles must be ≥ 1 (which is the value of internal parameter minSizeOrganicMoles); and if nOrganicCmpnds > 0 then its size ≥ nOrganicCmpnds.

On entry: the numbers of moles of each of the **nOrganicCmpnds** organic compounds present in the chemical system.

Constraints: for each organic compound *i* that is present, **OrganicMoles(*i*)** must be ≥0.0. If **nOrganicCmpnds** = 0 this array can be left blank.

Details: if interested, contact the author. This document is complete only for the description of calculations for inorganic systems.

15: **iOrganicOptions(:)** – INTEGER, a one dimensional assumed shape array.

On entry: this array must contain a set of options that determine how each organic compound in the system will be treated. Permitted option values are 0, 3, and 4. In systems where there are no organic compounds (i.e., **nOrganicCmpnds** = 0) this array can be left blank.

Constraint: must be ≥ minSizeiOrganicOptions, which is a value set internally in the *E-AIM* subroutine. A list of the current values is given in section 6.2.2.

Details: if interested, please contact Simon Clegg. This document is complete only for the description of calculations for inorganic systems.

16: **iOutput(:)** – INTEGER, a one dimensional assumed shape array. The size of the array must be ≥ minSizeiOutput, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On exit: the elements of this array contain information about the results of the chemical calculation.

Details: each element of the array contains the same data as for the non-module version of EAIMvar, which is described in section 5.1.

17: **Output(:) –** REAL, a one dimensional assumed shape array. The size of the array must be ≥ minSizeiOutput, which is an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On exit: contains the results of the calculation.

Details: this array will contain the results of the *E-AIM* calculation. See also the output array **OutputLabels**, in which labels describing each output quantity are placed. The contents of array **Output** are the same as for the non-module version of EAIMvar, and are described in section 5.1.

18: **OutputLabel(:)** – CHARACTER (LEN=*), a one dimensional assumed shape character array. The size of the array must be $\geq$ minSizeOutput, which is an internal parameter of the subroutine. The length of each element of the array the must be $\geq$ minLenOutputLabel, which is also an internal parameter of the subroutine. A list of the current values of the internal parameters is given at the end of section 6.2.2.

On exit: each element **OutputLabel(*i*)** contains a text label that describes the contents of the same element **Output(*i*)** of the results of the calculation. Values are assigned in EAIMvar to this argument when the routine is first called. On later calls it will only be re-assigned for iOptions(4) = 1.

Constraints: none.

Details: this array of labels is written only on the first call of the routine, or if **iOptions(4)** = 1.

19: **Messages** - CHARACTER (LEN=*), a string of arbitrary length. This length must be $\geq$ minLenMessages, which is an internal parameter of the *E-AIM* subroutine. A list of the current values of these internal parameters is given at the end of section 6.2.2.

On exit: contains error or warning messages relating to faults encountered in the main *E-AIM* routine, otherwise it will be blank.

Details: some testing of array sizes and input values is carried out in the *E-AIM* subroutine. If the tests are failed, the subroutine will return an explanatory message in this string (and no other results). The string should be checked after the *E-AIM* routine has been called.

Notes: where there is a failure, or an input error, that is detected within the *E-AIM* code for the chemical model an error message will be written to the file **eaim.err** that is likely to be in the same directory as the executable program that calls *E-AIM*. Execution of the *E-AIM* code will then stop.

20: **UsageRecord** - CHARACTER (LEN=*), a string of arbitrary length. This length must be $\geq$ minLenUsageRecord, which is an internal parameter of the subroutine. A list of the current values of these internal parameters is at the end of section 6.2.2.

On exit: a statement that includes the current date and time, and the time taken (in seconds) to execute the call of the *E-AIM* subroutine.

Details: it is the same as for the non-module version of EAIMvar and is described in section 5.1.

21: **errorFlags** – CHARACTER (LEN=*), a string of arbitrary length. This length must be $\geq$ minLenErrorFlags, which is an internal parameter of the subroutine. A list of the current values of these internal parameters is in section 6.2.2.

On exit: contains a set of single character flags indicating the success or failure of the *E-AIM* call, and the results of thermodynamic consistency tests.

Details: the first character in the string is always assigned a value, and the second character will be assigned a value if the chemical model has been called and returned a result. The remaining characters of the string are reserved for error flags which indicate failures of tests of the model result (which will be carried out if the value of **iOption(3)** is set to unity. The flag values are the same as for the non-module version of EAIMvar, and are described in section 5.1.

6.2.2 Internal EAIMvar parameters

The sizes of input and output arrays of the *E-AIM* subroutine are tested within the routine to make sure they are greater than or equal to certain lengths (for strings) and sizes (for arrays). The following parameters are defined in the source code of the *E-AIM* subroutine. They set minimum permitted lengths and sizes of the arguments of the routine. The values as currently set should be OK for most problems. They are only used for testing the validity of the arguments, and not their contents.

The parameters listed below are the same as those for the non-module version of EAIMvar (see section 5.2), with the exception that minSizeOutput and minLenOutputLabel now refer to the character array OutputLabel (which replaces the integer array iOutputLabel in the non-module version of the routine).

minSizeInorganicMoles = NCmaxBase + NAmaxBase + (NNmaxExt-1). These three parameters have values of 3, 6, and 2. Thus, minSizeInorganicMoles = 10.

minSizeOrganicMoles = 1, or the value of argument nOrganicCmpnds, whichever is greater.

minSizeiGasOptions = NGmaxBase-1. The value of this parameter is 6, thus minSizeiGasOptions = 5

minSizeiSolidOptions_2 = NSmaxBase. The value of this parameter is 30.

minSizeiOrganicOptions = 6*minSizeOrganicMoles, thus minSizeiOrganicOptions is equal to the greater of 6 and 6*nOrganicCmpnds.

minSizeOutput : the number of elements of the double precision array argument **Output**, and character array **OutputLabel**, must be greater than or equal to this value.

minLenOutputLabel : the length of each element of the character array argument **OutputLabel** must be greater than or equal to this value.

minLenMessages : the length of the character string argument **Messages** must be greater than or equal to this value.

minSizeiOutput : the size of integer array argument **iOutput** must be greater than or equal to this value.

minLenUsageRecord	:	the length of the character string argument **UsageRecord** must be greater than or equal to this value.
minSizeIOptions	:	the size of the integer array argument **iOptions** must be greater than or equal to this value.
minLenErrorFlags	:	the length of the character string argument **errorFlags** must be greater than or equal to this value.

The values of the above parameters that set these minimum lengths (minLen…) and sizes (minSize…), are as follows:

```
   minSizeiOptions   = 5
   minSizeInorganicMoles  = 10, (calculated as NCmaxBase + NAmaxBase + (NNmaxBase - 1))
   minSizeiGasOptions     = 5, (calculated as NGmaxBase - 1)
   minSizeiSolidOptions_2 = 30 (equal to NSmaxBase)
   minSizeOrganicMoles    = MAX(1, nOrganicCmpnds)
   minSizeiOrganicOptions = MAX(6, 6*minSizeOrganicMoles)
   minSizeOutput     = 200
   minSizeiOutput    = 6
   minLenOutputLabel = 50
   minLenMessages    = 350
   minLenUsageRecord = 150
   minLenErrorFlags  = 10
```

6.2.3 Calling EAIMvar

No example is given here, because the specification of the problem, and the results, are essentially identical to those for the non-module version of EAIMvar. See Appendix 3 for a description of the test program for this routine, and coding requirements.

7. Files

This document, the dll files, and test programs, are provided in a zip file with the following directory structure and contents:

\ (top level): ReadMe.txt

\Manual: pdf copy of this manual.

\Non-Module: the non-module versions of the dll and test programs (sections 4 and 5 of this document), in the following subdirectories:

\Non-Module\Libraries: the .dll, .lib, and .exp files for the non-module based versions of EAIMstr and EAIMvar, as described in section 2 of this document.

\Non-Module\Test_str (64 bit): the test program Test_str.for, including the input data file, results file, and executable, for testing the non-module version of routine EAIMstr. The file

MakeTest_str is the command file for the Intel Fortran compiler to compile and link the test program.

\Non-Module\Test_var (64 bit): the same as above, except that these files are for testing the non-module version of EAIMvar.

\Module: the module versions of the dll and test programs, as described in section 6 of this document, in the following subdirectories:

\Module\Libraries: the .dll, .lib, and .exp files for the module-based versions of EAIMstr and EAIMvar, compiled to run in extended precision internally and on the 64 bit version of Windows (as indicated by the “q64” in the name). The directory also contains the eaim_routines.mod file that is needed for compilation of the test programs.

\Module\Test_str (64 bit): the test program Test_str.for, including the input data file, results file, and executable, for testing the module version of routine EAIMstr. The file MakeTest_str is the command file for the Intel Fortran compiler to compile and link the test program.

\Module\Test_var (64 bit): the same as above, except that these files are for testing the module version of EAIMvar.

Appendix 1. Specifying Input Data for Routine **EAIMstr**, Using the String Argument **Input**

This description has been adapted from the one on the *E-AIM* website for batch input, and is essentially the same. The text string argument **Input** must contain the following information, and in this order:

* temperature (in K)
* system pressure (in atm., generally set to 1.0)
* system volume (in m^3, generally set to 1.0)
* water case number (1, 2 or 3)
* water dissociation option (-1, 0 or 1),
* the fixed RH or total moles of water depending on the value of the water case
* the numbers of moles of inorganic ions, and NH_3
* integers options controlling the formation of solids and trace gas equilibration.
* amounts of user-defined organic species, if any (properties are entered in the **eaim.org** file), followed by 6 groups of integer options which control their behaviour.

The sections below explain how to specify problems in which there are only inorganic chemical species, and also those which contain organic compounds. At this time the full description of how to include organic compounds (which requires the use of data files in addition to the *E-AIM* subroutines) has not been written. Therefore only subsection A is relevant.

A. INORGANIC AMOUNTS AND OPTIONS

An example of the contents of text string **Input** are given below for a typical problem. The line has been split into two, and the letters beneath each number are for explanatory purposes and are not part of the input. Successive values should be separated by one or more spaces.

```
298.15D0  1.000  1.000   1   0   0.420   1.0    2.0    0.0   1.0    1.0
  a         b      c     d   e     f      g      h      i     j      k

0.0    0.0    0.0    0.0        3   3   3   3   3       0
 l      m      n      o         p   q   r   s   t       u
```

The meaning of each value is now described (and whether each entry is an integer, or real number, is also noted):

a - temperature (K). (REAL). Valid range is between 180 K and 330 K (Models I, II, and IV), or 298.15 K only (Model III).

b - system pressure (here 1 atm). (REAL). It is recommended at present that this value is left unchanged.

c - system volume (here 1 m^3). (REAL).

d, **f** - water case (**d**, INTEGER) and associated value (**f**, REAL). **d** = 1 means that the relative humidity of the system is to be fixed to the value of **f**, which in this example is 42% (specified as the fraction 0.42).

Alternative water options are case **d** = 2, in which case **f** must be the total number of moles of water in the system. Here the program will solve for the equilibrium distribution of water between the vapour and condensed phases and will give the calculated RH as an output. For **d** = 3, **f** is again the total number of moles of water in the system, but in this case it is not allowed to partition into the vapour, and remains in the condensed phase as a liquid, ice, or water of hydration.

e - this is the water dissociation option. (INTEGER). A value of -1 means that dissociation is not calculated for any input conditions. A value of zero means that if both numbers of moles of H^+ and OH^- are zero on input, then H_2O dissociation will remain off, and neither ion will be a variable in the calculations. If either H^+ or OH^- is present on input, then the other will be made a variable and water dissociation will be on. A value of 1 means that water dissociation will always be calculated, and both H^+(aq) and OH^-(aq) will be made variables in the calculation, even if their input amounts are zero.

g - o - these values (REAL) are, in order, the numbers of moles of H^+, NH4$^+$, Na^+, SO_4^{2-}, NO3$^-$, Cl^-, Br^-, OH^- and NH_3 present (see elsewhere for valid combinations of ions for Models I-IV).

p - this controls how gas phase HNO_3 is treated by the program. (INTEGER). A value of '3' means that it is assumed not to occur, hence all NO_3^- (and associated H^+) remain in the condensed phase(s). A value of '4' means, again, assume no formation of HNO_3(g) but report its equilibrium partial pressure over the liquid phase (if it exists, and also contains both H^+ and NO_3^-). A value of '0' means that HNO_3(g) can be formed and will be partitioned between the phases if the ions are present. No other values of **n** are permitted.

q - as **p** above, for the gas HCl.

r - as **p** above, for the gas NH3.

s - as **p** above, for the gas H_2SO_4.

t - as **p** above, for the gas HBr.

u - the number of solids whose options are to be individually entered. (INTEGER). In the above example, **u** is zero, which means that the program will look for all the possible solids that can form and include them in the equilibration.

If **u** is equal to 1 or more, then it must be followed by the reference numbers and associated option values (both integers) of the **u** solids to be treated specially. For example, '1 10 3' instead of '0' would mean:

* 1 solid phase
* reference number = 10 (i.e., $(NH_4)_2SO_4$))
* option = 3 (means exclude from calculation).

The only other available option is '4', which means exclude from the calculation but report the saturation ratio.

Another example, '2 10 3 13 4' instead of '0' means:

* 2 solids
* 1st solid reference number = 10 (i.e., $(NH_4)_2SO_4$)
* 1st solid option = 3 (means exclude from calculation).
* 2nd solid reference number = 13 (i.e., NH_4NO_3)
* 2nd solid option = 4 (means exclude from calculation, but report the saturation ratio)

Reference numbers of all the solids in the model are given below (this is the same list as in section 5.1):

Ref:	Solid	Ref:	Solid	Ref:	Solid
1	H2O(ice)	10	(NH4)2SO4	18	Na2SO4
2	H2SO4	11	(NH4)3H(SO4)2	19	Na2SO4.10H2O
3	H2SO4.H2O	12	NH4HSO4	20	Na3H(SO4)2
4	H2SO4.2H2O	13	NH4NO3	21	NaHSO4.H2O
5	H2SO4.3H2O	14	2NH4NO3.(NH4)2SO4	22	NaHSO4
6	H2SO4.4H2O	15	3NH4NO3.(NH4)2SO4	23	NaH3(SO4)2.H2O
7	H2SO4.6.5H2O	16	NH4NO3.NH4HSO4	24	Na2SO4.(NH4)2SO4.4H2O
8	HNO3.H2O	17	NH4Cl	25	NaNO3
9	HNO3.3H2O			26	NaNO3.Na2SO4.H2O
				27	NaCl
				28	HCl.3H2O
				29	HNO3.2H2O
				30	NaCl.2H2O

B. USER-SPECIFIED ORGANIC AMOUNTS AND OPTIONS

The following quantities are entered, in the order given below, for each of the N organic solutes specified in the **eaim.org** file. They should be placed after the last of the inorganic options for the problem, on the same line. (If there are no organics, this entire section can be ignored.)

A.	moles of each solute, (for organics 1 -> N)	(REAL)
B.	gas options, (1 -> N)	(INTEGER)
C.	solid options, (1 -> N)	(INTEGER)
D.	mixed solid options, (1 -> N)	(INTEGER)
E.	dissociation options, (1 -> N)	(INTEGER)
F.	liquid/liquid equilibrium options (1->N)	(INTEGER)
G.	organic salt options (1 -> N)	(INTEGER)

The integer options values for items **A** to **F** above are as follows:

0 - allow the gas/solid/mixed solid to form, the dissociation (of organic acid or base) to occur, or the liquid/liquid equilibrium to be calculated if the appropriate quantities have been set in the **eaim.org** file. For example, if a value of the vapour pressure or Henry's

law constant has been specified then if the option value is 0the compound will be allowed to partition into the gas phase. If the thermodynamic quantity has not been specified then the option value will have no effect, and can safely be left at zero. In the case of dissociation, of an organic acid or base, a value of 0 means that the program will calculate the equilibrium.

3 - prevent (switch off) gas/solid/mixed solid/dissociation, or a liquid/liquid equilibration, from occurring. So, to use the above examples, a value of 3 for the gas option of the organic prevents it partitioning into the gas phase even if the compound has been assigned a Henry's law constant or vapour pressure. For liquid/liquid equilibrium this option constrains any compound that would otherwise be able to partition between both liquid phases to exist only in the aqueous phase.

4 - this option value has the same effect as a value of 3, but the equilibrium vapour pressure or solid saturation ratio will be calculated and included in the output.

The "mixed solid" formation option, item **D**, should be set to 3 (ie., switched off) as this feature has not been developed and tested. Note: currently this option is reset to 3 internally in *E-AIM*, and the user input is ignored.

Item **G**, the organic salt options: The model is able to calculate the formation of aminium sulphate, nitrate, and/or chloride salts in systems containing amines (that have been neutralised to aminium ions by H^+) and the three inorganic anions. (Of course the user must also have entered the appropriate equilibrium constants in **eaim.org**.) The options for all three salts are represented by a single integer as follows. The option value itself is the sum of the following codes:

1 - switch off formation of aminium sulphate salt
2 - switch off formation of aminium nitrate salt
4 - switch off formation of aminium chloride salt

So, if the formation of all three salts is to be switched off then the integer option value is 1 + 2 + 4 = 7. If only the sulphate and chloride salts are to be switched off then the option value will be 1 + 4 = 5. If only the chloride salt is to be switched off then the value of the option is 4. If all salts are to be allowed to form then the option value is 0 (zero). This value (i.e., 0) should also be used when the organic is not an amine and this option is therefore irrelevant.

Here is an example of how to input the organic amounts and options:

Below is input needed (i.e., following item '**s**' in section 2 above) for a system containing 2 organic compounds (1.0 mole of the first, and 2.5 moles of the second). The first organic is an amine (and can form solid sulphate, nitrate, and chloride salts). Partitioning of the first organic into the gas phase is switched off (item B), as is the aqueous phase dissociation of the second (item E). The formation of the aminium nitrate salt (for the first organic only) is switched off (item G):

```
1.0 2.5   3 0   0 0   0 0   0 3   0 0   2 0
   A        B     C     D     E     F     G
```

The letters indicate which pairs of numbers correspond to items **A-G** above.

C. EXAMPLE OF SYSTEM CONTAINING BOTH INORGANIC AND ORGANIC COMPOUNDS

The line of input below specifies a chemical system containing 0.1 mol m^{-3} of H_2SO_4, 0.15 mol m^{-3} of $(NH_4)_2SO_4$, and three organic compounds for which the amounts are 0.05 0.075 and 0.9 mol m^{-3}. The thermodynamic properties of the three organic compounds must be specified in file eaim.org, and the details of how to do this are described in the header of that text file.

In this system we fix the RH at 0.55 (55%), and prevent the partitioning of H_2SO_4 from the condensed phase into the gas phase. We also switch off the formation of the solid salt NH_4HSO_4(s). The second organic compound is an acid, which dissociates in the aqueous phase, and in this calculation we prevent the dissociation from occurring (ie, we switch it off). Finally, for this example, the third organic compound is an amine and the formation of aminium sulphate salt is also switched off.

Here is the single line of input, split into three parts:

```
  298.15D0  1.000  1.000    1   0   0.55     0.2    0.3    0.0   0.25   0.0
    a         b      c      d   e     f       g      h      i     j      k

  0.0    0.0    0.0    0.0         0   0   0   3   0      1 12 4
   l      m      n      o          p   q   r   r   t      u

  0.05 0.075 0.09  0 0 0  0 0 0   0 0 0  0 3 0  0 0 0  0 0 1
          A          B      C       D      E      F      G
```

The letters below each section of the line are not part of the data, but are added here to help the user refer to the notes and definitions above.

Appendix 2. Simple Test Programs (non-module routines)

The dll files are supplied with two Fortran test programs, **Test_str.for** (which calls routine **EAIMstr**), and **Test_var.for** (which calls **EAIMvar**). Each program has its own data file (Test_str.dat, and Test_var.dat), which it reads to obtain the input data a small number of problems. Users can alter, or add to, these problems in the data files as needed.

In both the examples provided, the first problems being solved are those described in sections 4.3 and 5.3.

Using the Intel Fortan compiler, the test programs were compiled and linked to the libraries with the following command: “ifort @makeTest”, where makeTest is the name of a command file (a simple text file) that contains all the options needed for compiling and linking the test program(s) with the dynamic link library. Using the program Test_var.for as an example, these options are:

```
## (1) compile options for the test program
#  ----------------------------------------
   /double_size:64 /integer_size:32 /check:all /traceback /O1 /static

## (2) output (executable) file name
#  ----------------------------
    /exe:Test_var.exe

## (3) the name of the source file containing the test program
#  -------------------  --------------------------------------
   Test_var.for

## (4) link to the E-AIM dll library
#  ---------------------------------
   /link EAIM_q64.lib
```

(All the lines starting # are comments within the command file). The first two compile options in (1) set the sizes used for double precision and integer variables in the test program. The values listed above (64 and 32) are defaults, so these options can be omitted. The /check:all and /traceback options set comprehensive error checking in the test program (useful if you alter it, and need to debug). The option /O1 sets a low level of code optimization (this could be given a higher value), and /static causes the executable to link to all libraries statically. These libraries include those provided with the compiler – which you don’t normally see or have to specify – and static binding means that the executable should run on computers on which the compiler software hasn’t been installed.

In (2) above the name of the executable test program is set to Test_var.exe, and in (3) the name of the Fortran file to be compiled is given.

The /link option in (4) specifies that the test program will be linked to the dynamic link library EAIM_q64. **Important note**: the three files EAIM_q64.lib, EAIM_q64.dll, and EAIM_q64.exp should be present in the same directory as the test program when it is compiled, and in the same directory as the executable Test_var.exe when it is run.

Appendix 3. Simple Test Programs (module routines).

In this case a single dynamic link library is provided (EAIM_module_q64.dll), compiled so that *E-AIM* runs internally in extended precision and for use on machines running 64 bit Windows. The names of the files containing the test programs are the same as for the non-module case (Appendix 2), though the code has been changed as needed by the module-based EAIMstr and EAIMvar routines. There are two main changes: first, the inclusion of the statement ‘USE EAIM_routines’ at the beginning of the program; second, the smaller numbers of arguments needed for EAIMstr and EAIMvar compared to the non-module versions. The procedure for compiling and running the programs is the same as for the non-module case, except that the file eaim_routines.mod (which is included with the library files) must be present in the same directory as the source code of the test programs when they are compiled.

Appendix 4. Notes on Running ***E-AIM*** Subroutines in Different Programming Environments.

It is intended that this section will contain notes describing users’ experiences running the *E-AIM* routines in different environments (being called from programs written in languages other than Fortran). Basically it will be a “hints and tips” section. The simplest case is likely to be where the routines are called by other Fortran programs.

1. Calling from Mathematica (Windows)

This has been done successfully by Scot Martin at Harvard (smartin@seas.harvard.edu), for use on his *Aerosol Calculator* website (http://aerosolcalculator.seas.harvard.edu/webMathematica/SingleParticle.jsp).