PLASMAKIN Manual

Nuno R. Pinhão
ITN
Nuclear and Technological Institute

Estrada Nacional 10
2686-953 Sacavém
Portugal
Email: npinhao@itn.pt
URL: www.itn.pt/GE

PLASMAKIN Manual

by Nuno R. Pinhão

Reference Manual version 1.0 for *PLASMAKIN* V3.0 Edition
Published 2001
Copyright © 2001 by Nuno R. Pinhão

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, and with the Front-Cover Texts being "Preface" and with the Back-Cover Texts being "Appendix". A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

Preface ..... **i**
Abstract i
Intended Audience i
Other Sources of Information i
Conventions Used in this Manual i
1. Introduction **1**
What is *PLASMAKIN* 1
Limitations of *PLASMAKIN* 3
Units 4
2. Overview **1**
Usage 1
Data Model 2
3. Public Parameters and Variables **4**
`KTable` 6
`NCasc` 7
`NCh` 7
`NKea` 8
`NKgas` 9
`NKPhEm` 9
`NKReac` 10
`NKRImp` 11
`NMaxReacSpc` 11
`NnC` 12
`NnTV` 13
`NnV` 13
`NnVX` 14
`NPhot` 14
`NReverse` 15
`NSKea` 15
`NSpecies` 16
`NKSurf` 17
4. Derived Data Type Definitions **18**
DATA_COLUMN 19
PHYS_PROPERTY 19
5. Procedures **21**
`pkCleanData` 22

pkGetParticle ... 23
pkGetPhotonEmission ... 25
pkGetPowerLosses ... 27
pkGetReacCoef ... 28
pkGetReverseCoef ... 30
pkGetSources ... 32
pkGetValue ... 33
pkIsPhotoElec ... 35
pkReadBaseData ... 36
pkReadChemReactions ... 37
pkReadData ... 39
pkReadSpecies ... 40
pkSetPhoton ... 41
pkSetReacCoef ... 43
pkSetValue ... 44
SetRate ... 46
6. Data Input Format ... 48
PLASMAKIN_DATA ... 49
CHEM_SPECIES ... 50
CHEM_REACTION ... 54
References ... 58
A. Error Messages ... 59
B. GNU Free Documentation License ... 62

List of Figures

F-1. Relationships between *PLASMAKIN* and other program units or data files. 1

List of Examples

E-1. `KTable`: Using `KTable` to control the printing sort order of variables. 6
E-2. `pkCleanData`: Deallocating *PLASMAKIN* data. 23
E-3. `pkGetParticle`: Reading the names of all species into an array. 24
E-4. `pkGetParticle`: Example showing how the returned values depend on the data type of the `Value` argument. 24
E-5. `pkGetPhotonEmission`: Getting the photon emission distribution in a 1D discharge 26
E-6. `pkGetPowerLosses`: Inquiring the power losses using the keyword form for dummy arguments. 28
E-7. `pkGetReacCoef`: Inquiring the electron collision rates. 30
E-8. `pkGetReverseCoef`: Evaluating the electron superelastic cross sections from the corresponding forward process cross section. 31
E-9. `pkGetSources`: Computing the source terms in a 1D discharge taking account of gas temperature gradients. 33
E-10. `pkGetValue`: Inquiring the gas temperature and density. 35
E-11. `pkIsPhotoElec`: Computing the photoelectron emission in a 1D discharge model. 36
E-12. `pkReadBaseData`: Reading the basic data in the input file. 37
E-13. `pkReadChemReactions`: Reading the chemical reactions in the input file. 39
E-14. `pkReadData`: Reading the input file. 40
E-15. `pkReadSpecies`: Reading the basic data in the input file. 41
E-16. `pkSetPhoton`: Setting the photon density prior to computing the source terms for other species. 42
E-17. `pkSetReacCoef`: Modifying the value of the 'reverse' property. 44
E-18. `pkSetReacCoef`: Updating temperature dependent rate coefficients or coefficients defined through an external routine: 44
E-19. `pkSetValue`: Setting the ratio of the number of molecules in the two first vibrational levels 46
E-20. PLASMAKIN_DATA: Examples of PLASMAKIN_DATA **NAMELIST** syntax. 50
E-21. CHEM_SPECIES: Example of CHEM_SPECIES **NAMELIST** syntax. 53
E-22. CHEM_REACTION: Example of CHEM_REACTION **NAMELIST** syntax. 56

Preface

Abstract

PLASMAKIN is a software library to handle physical and chemical data used in plasma physics modeling and to compute kinetics data from the reactions taking place in the gas or at the surfaces — particle production and loss rates, photon emission rates and energy exchange rates.

This manual describes all the public variables and routines in *PLASMAKIN* and the data input syntax. It gives all the information needed to use the *PLASMAKIN* library in programs, to write data input files and to understand the errors messages produced by *PLASMAKIN*. However it does not explains the physics and chemistry concepts underlying the library. For that purpose the indicated references should be consulted.

Intended Audience

This manual is intended for programmers with a good background in physics and chemistry or conversely, physicists or chemists with programming skills. In any case a basic understanding of the Fortran 90/95 language is needed.

The reader should be familiar with the physics and chemistry concepts underlying the library and be able to consult the article describing *PLASMAKIN* [Pinhão01].

Other Sources of Information

The main reference for the physics and chemistry processes included in *PLASMAKIN* is an article published in Computer Physics Communications [Pinhão01].

The *PLASMAKIN* library and a sample program and data file can be obtained from the CPC library (http://www.cpc.uk).

The author maintains a mailing list on *PLASMAKIN* news, errors and updates. This mailing list can be subscribed sending an email to the author (mailto:npinhao@itn.pt?subject=Subscribe pk list).

Conventions Used in this Manual

In this manual the following conventions are used:

`expression`

Terms in bold and courier font indicates information supplied by the user

{option1 | option2...}

Braces enclose a list from which you must choose one, but only one, item.

[option] [option1 | option2...]

Square brackets enclose optional items or a list of optional exclusive items.

...

A horizontal ellipsis means that the preceding item can be repeated

`call pkReadData`

Courier font is used for code samples

value, Ivalue, Lvalue, Rvalue, Cvalue

These words indicate, respectively, a generic type, a Fortran INTEGER, LOGICAL, REAL or CHARACTER value

IN, OUT, INOUT

These terms classify arguments that, respectively, are used to provide data to a procedure but are not modified inside it; are used to pass data from the procedure back to the calling program but are undefined on entry; or can both provide data to the procedure and return data to the calling program.

Chapter 1. Introduction

A large number of problems in plasma physics involve the consideration of several chemical species [1] and reactions. The solution of such problems invariably requires the ability to read, classify, sort and manipulate particles and reactions and, frequently, the evaluation of source terms in chemical kinetic equations, the energy transferred by the electrons to the other particles, the gas heating due to collisions or the photon emission spectra.

The handling of these data frequently represents a significant fraction of codes, development and maintenance time and is a source of errors. It would clearly be advantageous to have a package able to deal with these data independently of the number or nature of the species and chemical reactions involved and of the problem being solved or the method used. Such a package could be used as a "black box" moving the description of particles and reactions from code to a data file, isolating the evaluation of terms related to the chemical kinetics from the remaining program, allowing the user to concentrate on the physical problem and on the algorithm to solve it. Once the code developed, it would allow an easy and fast modification and testing of chemical models. *PLASMAKIN* was developed to fulfill this purpose.

What is *PLASMAKIN*

PLASMAKIN is a collection of routines to parse an input file describing the species and reactions present in a discharge, handle this data and to compute chemical kinetics data characterizing the discharge. The library supports any number of species and reactions, is independent of the number of dimensions of the problem and can be used both in stationary and time-dependent problems. Although *PLASMAKIN* has been written for plasma physics problems it can be used or adapted to any field dealing with chemical kinetics.

The user intervention is limited to the preparation of an input file and in making the appropriate calls to *PLASMAKIN* procedures in a driver program. The input file uses a language close to the physical and chemical notation and the parser is able to detect errors in this file and to deduce the values of several properties, reducing the amount of data that must be written in the file.

A large number of species properties are supported and fully described in CHEM_SPECIES. Vibrational series can be indicated and the corresponding initial relative populations are estimated using a Gordiets-Treanor distribution [*Gordiets72*]:

$$\delta_{s,v} = \begin{cases} \exp\left[-v\left(\frac{\epsilon_{1,0}}{k_B T_v} - (v-1)\frac{\zeta_e \theta}{T_g}\right)\right] & , \quad v \le v^* \\ \delta_{s,v^*} \cdot \frac{v^*}{v} & , \quad v^* \le v \le v^{**} \\ \exp\left(-\frac{\epsilon_v}{k_B T_g}\right) & , \quad v > v^{**} \end{cases}$$

Unimolecular, bimolecular and termolecular reactions can be included.

Optical emission and absorption are examples of unimolecular reaction important in plasma physics. Radiation imprisonment can also be accounted including a constant or space dependent escape factor [*Molisch98*].

Cascade levels (influencing the population of lower levels by spontaneous emission but whose density is not followed) are supported and the corresponding branching ratios are automatically evaluated.

A large number of rate coefficients have an Arrhenius temperature dependence. However reactions in plasma can have more complex temperature dependencies or, as is the case for electron collision reactions, depend on the electron temperature. To accommodate this, rate coefficients with a power series dependence on temperature in the exponential term can be used in *PLASMAKIN*:

$$k_i = \alpha_i^0 T^{\beta_i^0} exp\left(\sum_{j=1}^{5} \frac{\alpha_i^j}{T^{\beta_i^j}}\right)$$

where T is the electron temperature for electron collision reactions and the gas temperature for other cases. The initial temperature is read in the data file and can be changed during program execution.

As the rate coefficient for forward and reverse reactions are related by the principle of detailed balancing, *PLASMAKIN* automatically evaluates the reverse rates from the corresponding forward rates.

A large number of reactions with vibrational levels are formally identical, the only difference being the vibrational number(s). To avoid the need of writing all this reactions *PLASMAKIN* is able to read a single description of a group of reactions involving species from a vibrational series, create the corresponding individual reactions and compute the rate coefficients.

Finally, to compute rate coefficients known as a function of the reduced field E/N or mean energy (as is frequently the case for electron rate coefficients), rate coefficients depending on vibrational quantum numbers or other expressions, a user supplied routine can be used.

Termolecular Reactions - Two types of pressure-dependent reactions are included: recombination reactions and chemically activated bimolecular reactions. Reaction rates are calculated using the [*Lindemann22*] and [*Troe83*] formulation. In both cases the 3rd body concentration can be set proportional to the total pressure of to the partial pressure of a selected gas species.

Surface reaction can be included and the indication of surface names allows differentiating the reactions according to the surface.

Energy losses - The energy exchanged in reactions is calculated from the values of the formation enthalpy at the gas temperature. As the gas temperature in cold plasmas is within a few hundred degrees to T_0, the formation enthalpy at gas temperature is not much higher than the standard value and is approximated by:

$$H_f(T_g) = H_f^{\ominus}(T_0) + C_p \Delta T$$

A throughout description of the equations and expressions used in *PLASMAKIN* can be found in [*Pinhão01*].

The package is written in standard Fortran 95 as a MODULE unit. However, to allow an easy integration with FORTRAN 77 code only the FORTRAN 77 intrinsic data types are used for public data and *PLASMAKIN* procedures can be called by FORTRAN 77 code. The usage of Fortran standard naming conventions, error diagnostics and error messages eases the usage of the package.

Limitations of *PLASMAKIN*

Although *PLASMAKIN* is prepared to deal with a large diversity of properties and types of chemical reactions and to respond to different problems found in plasma physics, some reaction types are not yet supported or functions found in other chemical kinetic libraries or programs [*Kee96*, *Carver97*] have not been included.

The following functions have been intentionally left out:

- It does not include any particle or chemical reaction database. In fact, although there are some reference databases, the number of gases and reactions covered in those databases is still small. Also, different procedures and levels of approximation for the description of species and reactions are common in this field. Consequently the author has decided to leave to the user the selection of data and has instead provided enough flexibility in the data structures in order to accommodate different ways of data representation.
- *PLASMAKIN* does not provide either any differential equations solver or any other integration procedure. As the number of numerical procedures available is relatively large and in continuous evolution, the selection of the appropriate algorithm is largely problem dependent and many are freely available and well documented [2] the choice of the appropriate algorithm was again left to the user.
- *PLASMAKIN* does not includes an electron Boltzmann equation solver. Tough the electron and the chemical kinetics are interconnected - the rate coefficients depend on the electron kinetics and the former depends on species concentration - the available methods of solution for the electron Boltzmann equation are valid only in clearly defined conditions and there is no "universal" solver. Again, for problems where both kinetics must be solved together, the choice of the appropriate electron Boltzmann equation solver was left to the user. Nevertheless an electron kinetics library, build on top of *PLASMAKIN*, and solving the homogeneous electron Boltzmann equation in the classical two term approximation is available from the author (mailto:npinhao@itn.pt?subject=Inquire on Boltzlib) for non-commercial use.

- Finally *PLASMAKIN* does not use some approximations common in other fields - i.e. the "family approach" used in atmospheric chemistry - as they are seldom used in "gaseous electronics" problems.

The following capabilities are been considered for the next version:

- Support for ionic excited states.
- Account as the 3rd body in three body collisions, of species with different efficiencies in promoting the reaction.
- Support for surface reactions with different adsorbed atoms.
- Evaluation of surface coverage and sticking coefficients.
- Interfaces for C, Pascal and Python mixed language programming.

Units

With a few exceptions the physical properties are handled in *PLASMAKIN* using S.I. units. The exceptions are the energy, the electron temperature, where in both cases eV is used, and the vibrational frequency where for historical reasons cm^{-1} is used [3].

However in the input file the values of several physical properties can also be indicated in other units. In this case the unit used must be indicated after the value, according to the syntax:

```
property = value, 'units'
```

The units available for each property are listed in the corresponding reference page.

When these units are found the corresponding value are converted to S.I. units and the conversion factors are saved. When the calling program inquires the value of a given physical property the corresponding conversion factor is used to back-convert to the initial units. Thus the value passed to the calling program is always in the units used in the data file.

The conversion operations and conversion factors are always hidden from the calling program. This allows some flexibility on data input and output and at the same time frees the programmer from elementary but error-prone operations.

Notes

1. In this manual each quantum level is considered as an independent species

2. See, for instance, GAMS or NetLib.
3. In spectroscopy the vibrational frequency is measured in units of ν/c, where *c* is the velocity of light.

Chapter 2. Overview

Usage

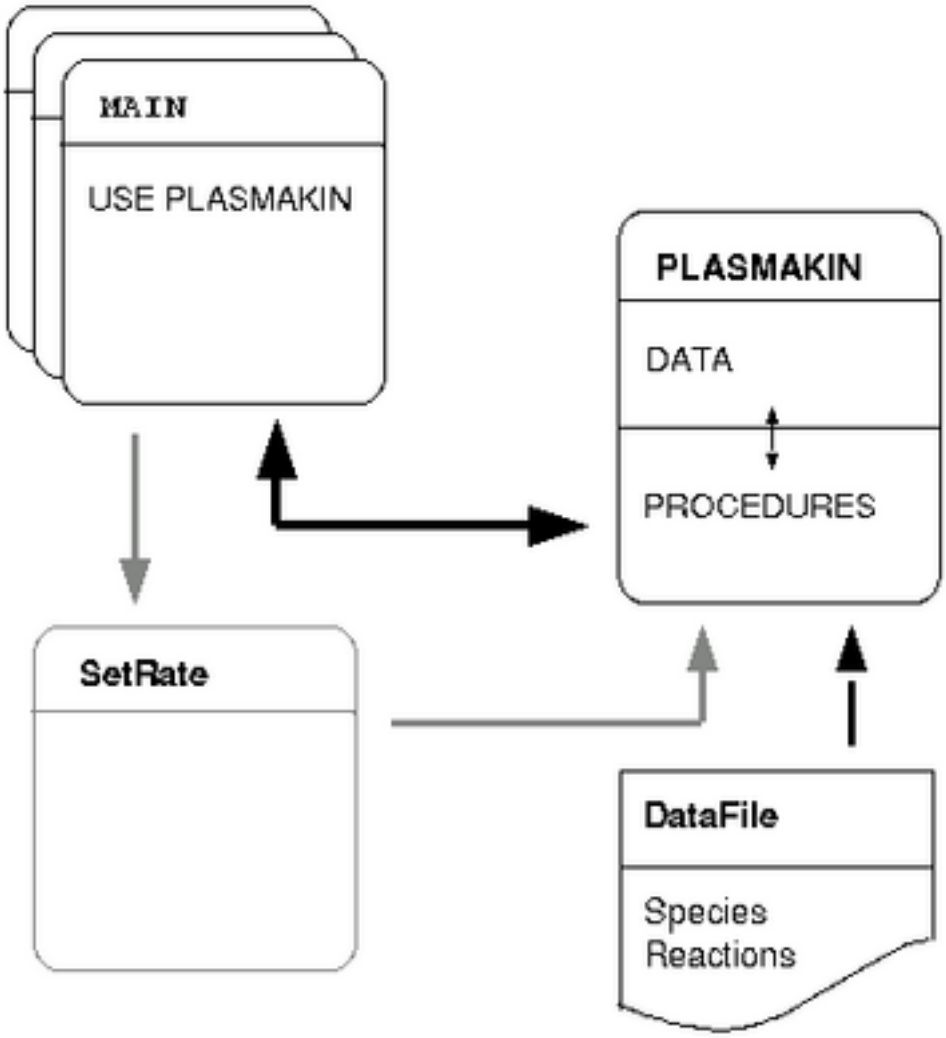
The *PLASMAKIN* library is a Fortran 90/95 MODULE unit with public and private data and procedures. The library is accessible trough a standard **USE** instruction:

```
USE PLASMAKIN
```

The module reads his data from an input file and can call an optional user supplied routine `SetRate`, to evaluate complex rate reaction expressions, but is otherwise, self-contained.

Figure F-1 represents the relationships between *PLASMAKIN* and other routines or data files. The arrows indicate the direction of information flow.

Figure F-1. Relationships between *PLASMAKIN* and other program units or data files.



Data Model

PLASMAKIN makes extensive use of derived data types. However only two of them are public and even so they are not intended to be used outside the *PLASMAKIN* MODULE. The only reason why they are public is because they are used in **NAMELIST** reading.

The hiding of data structures guarantees that future modification of the data model has limited, if any, impact in the calling programs. It also considerably simplifies mixed language programming.

Public data include parameters, scalars and arrays variables, the above derived types and procedure interfaces.

With the indicated exception, all the public variables and arguments in *PLASMAKIN* procedures have intrinsic Fortran data types. This allows the use of the library with FORTRAN 77 conforming code with little modifications [1].

Particle or reaction properties are only accessible through access routines allowing a strict control of which data can be inquired or modified by the calling program.

The calling program should not modify the value of any of the public variables as they are extensively used by *PLASMAKIN* to internal processing. The possibility of modification of any of those variables is a potential source of errors. Again this could be avoided by the use of access routines but it would be cumbersome and introduce time penalties as these variables are frequently used in the calling program. The author has decided to rely on the user experience to avoid this kind of errors.

Notes

1. You need off course a Fortran 95 compiler and the Fortran 90/95 USE instruction to invoke *PLASMAKIN*

3. Public Parameters and Variables

With the exception of the array `KTable(:)` and the **PARAMETER** `NMaxReacSpc`, all the other variables are counters representing the number of species or chemical reactions satisfying some condition.

These counters are interrelated and other useful values can be deduced from the available counters. The following values have not been explicitly defined and can be useful:

- Nº of neutral excited species = `NnVX - NnV`
- Nº of electron collision direct processes = `NKea - NSKea`
- Nº of gas collision processes between atoms and molecules = `NKgas - NKea`
- Nº of reverse gas collision processes between atoms and molecules = `NReverse - NSKea`

When the input file is read the species and chemical reactions are classified and sorted. The new order follows the classification shown in Figure F-2 and Figure F-3 for, respectively, species and chemical reactions.

Figure F-2. Representation of the order of species in *PLASMAKIN*. The electrons have the highest index of the charged species. The region (a) represents the neutral, excited species.

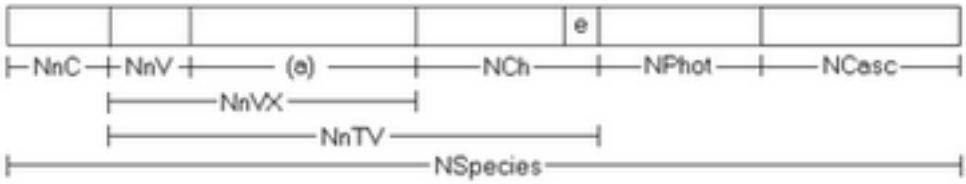
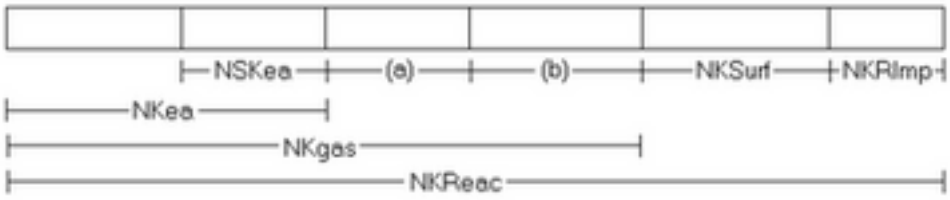


Figure F-3. Representation of the order of reactions in *PLASMAKIN*. The region (a) and (b) represent respectively, the number of direct and reverse gas collision processes between atoms and

molecules.



Caution

The values of these counters are set by *PLASMAKIN* and should not be modified by the user program.

KTable

Name

KTable — Translation table for reaction coefficient indexes

Type and Attributes

Integer array with dimension NKReac

Description

KTable is a translation table to map the internal *PLASMAKIN* reaction indexes to the corresponding reaction indexes in the input file.

When the input file is read the reactions are classified and sorted to speed-up further operations. However the interpretation of several results depends on the relationship with the input file reactions. The array KTable holds this information. A typical use of the array KTable is in write instructions as in the example bellow.

Example

Example E-1. KTable: Using KTable to control the printing sort order of variables.

```
REAL(8), ALLOCATABLE :: KRate(:)
...
ALLOCATE( KRate(NKReac) )
...
CALL pkGetReacCoef( 'value', KRate(:), 1 )                    ❶
print *, KRate(:)                                             ❷
print *, KRate( KTable(:) )                                   ❸
```

❶ The array KRate holds the values of the reaction rate coefficients sorted by *PLASMAKIN*.

❷ This instruction prints the reaction coefficient values in the order used in *PLASMAKIN*.

❸ This instruction prints the reaction coefficient values in the same order as reactions are listed in the input file

`NCasc`

Name

`NCasc` — Number of cascade levels

Type and Attributes

Integer

Description

`NCasc` is the number of cascading species - excited species decaying only by spontaneous emission and affecting the population of lower levels, but whose population is not followed.

`NCasc` is calculated counting in the input file the number of species with the *`cascade`* flag set to **`T`**.

NCh

Name

NCh — Number of charged species

Type and Attributes

Integer

Description

NCh is the number of different types of charged species, including positively and negatively charged ions and the electrons.

NCh is calculated counting in the input file the number of species with the value of *charge* different from **0**.

NKea

Name

NKea — Number of coefficients for electron collision reactions

Type and Attributes

Integer

Description

NKea is the number of different types of reactions with electrons.

`NKea` is calculated counting the number of reactions indicated in the input file where the name of at least one species in the *`reactants`* list is **`'e'`**.

NKgas

Name

`NKgas` — Number of gas phase reactions

Type and Attributes

Integer

Description

`NKgas` is the number of different types of reactions taking place in the gas.

`NKgas` is calculated counting in the input file the number of reactions with the value of *`locus`* equal to the default value **`'gas'`**.

NKPhEm

Name

`NKPhEm` — Number of photon emitting reactions

Type and Attributes

Integer

Description

`NKPhEm` is the number of different types of reactions emitting photons.

`NKPhEm` is calculated counting the number of reactions with a photon in the *`products`* list.

NKReac

Name

`NKReac` — Total number of reactions

Type and Attributes

Integer

Description

`NKReac` is the total number of different types of reactions, including surface reactions and radiation imprisonment reactions.

`NKReac` is calculated counting the number of reactions defined in CHEM_REACTION **NAMELIST** groups. The number of reactions counted in each instruction depends on the value of the *`opposing`* property and on the type of species indicated in the *`reactants`* and *`products`* lists:

- If the *`opposing`* property equals **T** the number of reactions is doubled;
- If none of the species indicated in the *`reactant`* or *`product`* lists represents a vibrational series and *`opposing`* = **F**, the instruction represents one reaction. However if any of the species represents a vibrational series, all the corresponding reactions are considered and the number of reactions is given by the equation

$$\prod_i (k_i^{max} - k_i^{min} + 1) \times \left(v_i^{max} - v_i^{min} + 1 - \frac{|k_i^{max} + k_i^{min}|}{2} \right)$$

where the product is on the number of vibrational species on the side of the equation with the highest number of vibrational species. For clarity let this side be called `S` and the other side `F` [1].

v_i^{min} and v_i^{max} are, respectively, the minimum and maximum allowed values for the vibrational quantum number for species `i` in side `S`. If no other limits are imposed they are the minimum and maximum vibrational values for that series. k_i^{min} and k_i^{max} are limiting values for the `products` vibrational quantum number.

Notes

1. In most reactions `S` is the `reactants` side and `F` the `products` side.

   However when the number of vibrational species in `products` is higher (as in the recombination reaction $A^+ + B^- + M \rightarrow AB(v) + M$) the above equation is valid only if side `S` is the `products` side.

NKRImp

Name

`NKRImp` — Number of radiation imprisonment reactions

Type and Attributes

Integer

Description

`NKRImp` is the number of different types of radiation imprisonment reactions.

`NKRImp` is calculated counting the number of reactions with a photon in the `reactants` list but none in the `products` list.

NMaxReacSpc

Name

`NMaxReacSpc` — Maximum number of particles taking part or produced on a reaction.

Type and Attributes

Integer, parameter

Description

`NMaxReacSpc` is the maximum number of *`reactants`* or *`products`* in a reaction.

In *PLASMAKIN* this number is a parameter constant equal to **4**. However if this number is not appropriate, the user only has to modify this value in the declaration of `NMaxReacSpc` in the *PLASMAKIN* source file and recompile the library.

NnC

Name

`NnC` — Number of neutral species with constant concentration.

Type and Attributes

Integer

Description

`NnC` is the number of neutral species with constant concentration.

`NnC` is calculated counting the number of species with the value *`constant`* = **`T`**.

NnTV

Name

`NnTV` — Total number of species with variable concentration.

Type and Attributes

Integer

Description

`NnTV` is the number of species whose concentration changes and must correspond to the number of continuity equations on the model.

`NnTV` is calculated adding the number of charged species `NnC` with the number of neutral species with variable concentration, `NnVX`.

NnV

Name

`NnV` — Number of non-excited neutrals with variable concentration.

Type and Attributes

Integer

Description

`NnV` is the number of non-excited neutral species whose concentration changes.

`NnV` is calculated counting the number of species with *`energy`* = **`0`** among the `NnVX` species.

NnVX

Name

`NnVX` — Total number of neutrals with variable concentration.

Type and Attributes

Integer

Description

`NnVX` is the number of neutral atomic or molecular species with changing concentration.

`NnVX` is calculated counting the number of neutral, non-constant and non-cascade species.

NPhot

Name

`NPhot` — Number of different photon species

Type and Attributes

Integer

Description

`NPhot` is the number of different photons species.

`NPhot` is calculated counting the number of species with the word **`photon`** in *`name`*.

`NReverse`

Name

`NReverse` — Number of reverse reaction.

Type and Attributes

Integer

Description

`NReverse` is the number of reverse reaction processes.

`NReverse` is calculated counting the number of reverse reactions defined through CHEM_REACTION **NAMELIST** groups with *`opposing`* = **`T`**.

NSKea

Name

`NSKea` — Number of super-elastic electron collision reactions.

Type and Attributes

Integer

Description

`NSKea` is the number of superelastic collision reactions.

`NSKea` is calculated counting the number of reverse reactions where one of the *`reactants`* has the name **`'e'`**.

NSpecies

Name

`NSpecies` — Total number of species

Type and Attributes

Integer

Description

`NSpecies` includes all the species defined in the input file, including photons. However the third-body in three-body collisions, identified by the generic name **`'M'`** is not counted.

`NSpecies` is calculated counting the number of species defined in CHEM_SPECIES **NAMELIST** instructions including off course the vibrational series.

`NKSurf`

Name

`NKSurf` — Number of surface reactions.

Type and Attributes

Integer

Description

`NKSurf` includes all the reactions defined in CHEM_REACTION **NAMELIST** instructions with the value of *`locus`* different from the default value **`'gas'`**

4. Derived Data Type Definitions

The following two derived data types are public as they are used in data reading. However the user has no need to use them to communicate with *PLASMAKIN* as all the data read by *PLASMAKIN* can be inquired through access routines using only the intrinsic Fortran data types. These derived data types are documented here only to help understanding the data input format used by *PLASMAKIN*.

DATA_COLUMN

Name

DATA_COLUMN — Identifies the location of input data on an external file.

Structure Members

```
TYPE DATA_COLUMN
  CHARACTER(20)   ::  name                ! Name of datafile
  INTEGER         ::  index               ! Data index (column or block nº)
END TYPE
```

Description

The DATA_COLUMN derived type is used in the input file to address data located in other files. The string represents a file name and the index identifies the location of data in that file.

Frequently the index s used to identify a single column or data block. However if more than one column or data block must be read - as in the case of (x_1[, x_2, ...], y) values - the calling program is responsible for selecting the correct columns or data blocks based on this index. Usually the values are organized in consecutive columns or data blocks but the used is free to choose how to organize the data. The drawback is that he is also responsible by reading this data.

Example

In the following example the variable ***data_file*** has type DATA_COLUMN and is used to input the filename **'eTransport.txt'** and the index value **1**. In this case the file holds the electron transport properties (mobility and diffusion coefficient) as a function of the reduced electrical field.

```
&CHEM_SPECIES name = 'e', charge = -1, data_file = 'eTransport.txt',1 /
```

PHYS_PROPERTY

Name

PHYS_PROPERTY — Represents a physical property

Structure Members

```
TYPE PHYS_PROPERTY
  REAL(double)    ::  value              ! value of physical property
  CHARACTER(10)   ::  units              ! units of physical property
END TYPE PHYS_PROPERTY
```

Description

The PHYS_PROPERTY derived type is used in the *PLASMAKIN* input file to read physical properties. Two fields compose it, one for the numerical value and the other for the unit.

It is used only when a given property can be indicated in more than one unit. Otherwise only the numerical value of the property is used.

Example

In the following example all the variables in the **NAMELIST** have type PHYS_PROPERTY and both the value and the unit can be indicated. In this example although the value of the variable *Gas_n*, representing the gas density, is not indicated, the unit is needed to force the writing of results in cm^{-3}.

```
&PLASMAKIN_DATA Pressure = 0.15,'mbar', Gas_n = ,'cm-3', Gas_T = 300,'K' /
```

5. Procedures

pkCleanData

Name

pkCleanData — Deallocates arrays destroying all the information on chemical data

Description

The subroutine pkCleanData destroys all private arrays of *PLASMAKIN*. These arrays are used to hold the chemical data and are created when the data file is read.

Syntax

```
pkCleanData( [STAT] )
```

Name	Description	Data type and attributes
STAT	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

STAT = **0**: no errors; STAT = **n**: error number.

Comments

Why is this subroutine needed?

- Although most modern compilers are able to manage the memory efficiently and avoid memory leaks, it's a good programming practice to clean all arrays before leaving a routine or program.
- If you wish to build a program that uses more than one input data file (i.e. for batch processing) you must call this routine to clean the internal arrays before reading the next data file.

Examples

Example E-2. `pkCleanData`: Deallocating *PLASMAKIN* data.

```
INTEGER              :: ErrorStatus
CHARACTER, PARAMETER  :: StopCode = 'pkError: Memory deallocation error'
...
CALL pkCleanData( ErrorStatus )
IF( ErrorStatus /= 0 ) STOP StopCode
```

`pkGetParticle`

Name

`pkGetParticle` — Get the values of chemical species properties

Description

The subroutine `pkGetParticle` returns the value of a given property for one species or the values of that property for a sequence of species.

Syntax

```
pkGetParticle( Property, Value, Species[, STAT] )
```

Name	Description	Data type and attributes
`Property`	Case sensitive name of property	CHARACTER(len=*), INTENT(IN)
{*`Value`* | *`Value(:)`*}	Values(s) of property	{CHARACTER(20) | INTEGER | REAL(double)} , INTENT(OUT)

Name	Description	Data type and attributes
`SpeciesN`	{Index of *`Species`* | starting index of *`Species`*}	INTEGER, INTENT(IN)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The `Value` argument can be a scalar or an array. In the first case the value returned is the value of `Property` of `Species(SpeciesN)`. In the last case the values returned are the `Property` values from consecutive species from `Species(SpeciesN)` to `Species(SpeciesN+SIZE(Value)-1)`.

When the properties are of a derived-type, the result depends on the intrinsic type of the dummy argument `Value` and is the derived-type component with the same type as `Value`.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-3. `pkGetParticle`: Reading the names of all species into an array.

```
CHARACTER(20), ALLOCATABLE :: SpcName(:)                    ❶
...
ALLOCATE( SpcName(NSpecies) )
...
CALL pkGetParticle( 'name', SpcName, 1 )                    ❷
```

❶ The array `SpcName` will hold the names of all species

❷ The names of all species are assigned to the array `SpcName` following the order used by *PLASMAKIN*

Example E-4. `pkGetParticle`: Example showing how the returned values depend on the data type of the `Value` argument.

```
  CHARACTER(20) :: SpcFileName                                          ❶
  INTEGER       :: SpcFileIndex                                         ❷
  ...
  NAMELIST /TransportData/ Vd, D
  ...
  ! Reads transport properties in an external file
  DO i = NnC+1, NnC+NnTV
    CALL pkGetParticle( 'data_file', SpcFileName, i )                   ❸
    CALL pkGetParticle( 'data_file', SpcFileIndex, i )                  ❹
    OPEN( IOUnit, SpcFileName, ACTION='READ' )
    DO j = 1, SpcFileIndex      ! Reads file until finding the correct index
      READ( IOUnit, NML=TransportData, IOSTAT )
    END DO
    CLOSE( IOUnit )
END DO
```

❶ The variable `SpcFileName` will hold the name of the external file listing the transport parameters for the inquired `Species`.

❷ The variable `SpcFileIndex` indexes the transport data for each `Species` in the external file.

❸ In this case the value returned is a character string.

❹ In this case the value returned is an integer.

`pkGetPhotonEmission`

Name

`pkGetPhotonEmission` — Evaluates photon emission rates

Description

The subroutine `pkGetPhotonEmission` returns the photon energy and the photon emission rate for each photon.

Syntax

```
pkGetPhotonEmission( LDensity, PhRate[, STAT] )
```

Name	Description	Data type and attributes
`LDensity(NnTV)`	Local density of species with variable concentration	REAL(double), INTENT(IN)
`PhRate(2,NPhot)`	Photon energy and local photon emission rate for all photons	REAL(double), INTENT(OUT)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The array section `PhRate(1,:)` holds the photon energies and the array section `PhRate(2,:)` the photon emission rates.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-5. `pkGetPhotonEmission`: Getting the photon emission distribution in a 1D discharge

```
REAL(double), ALLOCATABLE :: Dens(:,:), PhDist(:,:), PhRate(:,:)
...
ALLOCATE( Dens(NnTV,Nx), PhRDist(NPhot,Nx), PhRate(2,NPhot) )
...
DO i = 1, Nx
  CALL pkGetPhotonEmission( Dens(:,1), PhRate )
  PhRDist(:,i) = PhRate(2,:)
END DO
```

pkGetPowerLosses

Name

pkGetPowerLosses — Evaluates the power losses and the relative contribution of each process

Description

The subroutine pkGetPowerLosses computes the electron collisional power losses, the power converted into heat in other collision processes, the power radiated and the relative contribution of each reaction for these losses.

Syntax

```
pkGetPowerLosses( LDensity[, eP][, eR][, HeatP][, HeatR][, RadP][, RadR]
  [, STAT] )
```

Name	Description	Data type and attributes
LDensity(NnTV)	Local density of species with variable concentration	REAL(double), INTENT(IN)
eP	Power lost by the electrons by unit volume	REAL(double), OPTIONAL, INTENT(OUT)
eR(NKea)	Relative contribution of each electron collision reaction to the electron power losses	REAL(double), OPTIONAL, INTENT(OUT)
HeatP	Power converted into heat in other collison processes by unit volume	REAL(double), OPTIONAL, INTENT(OUT)
HeatR(NKgas)	Relative contribution of each reaction to gas heating	REAL(double), OPTIONAL, INTENT(OUT)
RadP	Radiated power by unit volume	REAL(double), OPTIONAL, INTENT(OUT)

Name	Description	Data type and attributes
`RadR(NPhot)`	Relative contribution of each photon emission process to the radiated power	REAL(double), OPTIONAL, INTENT(OUT)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The power/volume losses are in Watt/L^3, where L^{-3} is the unit used to indicate the gas density in the input file.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-6. `pkGetPowerLosses`: Inquiring the power losses using the keyword form for dummy arguments.

```
REAL(double) :: ePLosses, HeatPLosses, RadPLosses
...
CALL pkGetPowerLosses( eP=ePLosses, HeatP=HeatPLosses, RadP=RadPLosses )
```

`pkGetReacCoef`

Name

`pkGetReacCoef` — Get the values of reaction coefficients properties

Description

The subroutine `pkGetReacCoef` returns the value of a given reaction coefficient property. If `Value` is an array, a list of values is returned.

Syntax

```
pkGetReacCoef( Property, Value, KReac[, STAT] )
```

Name	Description	Data type and attributes
`Property`	Case sensitive name of property	CHARACTER(len=*), INTENT(IN)
{*`Value`* | *`Value(:)`* | *`Value(:,:)`*}	Value(s) of property	{CHARACTER(20) | INTEGER | LOGICAL | REAL(double)} , INTENT(OUT)
`KReacN`	Index of reaction inquired if `Value` is a scalar or starting index of reaction if `Value` is an array	INTEGER, INTENT(IN)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The dummy argument `Value` returns the value of the property inquired and can be a scalar, a one or a two-dimension array. A two-dimension array is only used to inquire the name of the species involved in a reaction as *`reactants`* or *`products`*.

If the dummy argument `Value` is a scalar, the result is the value of the property for species `KReacN`. If `Value` is a one dimension array, the returned values are the property values from consecutive `SIZE(Value, DIM=1)` species, starting in species `KReacN`. If `Value` is a two-dimensional array, for each reaction inquired, the returned values are the `SIZE(Value, DIM=2)` first names of the species involved in the reaction as *`reactants`* or *`products`*. If `SIZE(Value, DIM=2)` > `NMaxReacSpc`, the values with index higher than `NMaxReacSpc` are returned empty.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-7. `pkGetReacCoef`: Inquiring the electron collision rates.

```
REAL(double), ALLOCATABLE :: eKRates(:)
...
ALLOCATE( eKRates(NKea) )
...
! The electron collision rates are the first NKea rates
CALL pkGetReacCoef( 'Value', eKRates, 1 )
```

`pkGetReverseCoef`

Name

`pkGetReverseCoef` — Get the values of reverse reactions properties

Description

The subroutine `pkGetReverseCoef` returns a list of values of a given property for reverse reactions. Two properties are available:

- The ratio, in the forward reaction, between the product of the degeneracy for the *`products`* species and the product of the degeneracy for the *`reactants`* species, - `'g_ratio'`
- The energy difference between the products and the reactants in the forward reaction - `'dE'`

Syntax

```
pkGetReverseCoef( Property, Value, KReacN[, STAT] )
```

Name	Description	Data type and attributes
`Property`	Case sensitive name of property	CHARACTER(len=*), INTENT(IN)
`Value(:)`	Values of property	REAL(double), INTENT(OUT)
`KReacN`	Starting index of reactions inquired	INTEGER, INTENT(IN)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The values returned are the values of the property inquired from consecutive `SIZE(Value)` reverse reactions starting in species `KReacN`.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-8. `pkGetReverseCoef`: Evaluating the electron superelastic cross sections from the corresponding forward process cross section.

```
INTEGER ::       NfKea,      &   ! N of forward e-atom reactions
                 NEbin,      &   ! N of energy bins
                 idE,        &   ! Index for the energy diff. between levels
                 jmax            ! Maximum index for superelastic CS
REAL(double) ::  Ebin            ! Energy bin
REAL(double),                &
ALLOCATABLE ::   g2_g1(:),   &   ! Ratio between degeneracy values
                 dE(:),      &   ! Energy difference between levels
 CrossSec(:,:)   ! Electron collision cross sections
...
NfKea = NKea - NSkea
ALLOCATE( CrossSec(NEbin,NKea) )
IF( NSKea > 0 ) THEN
  ! - Gets g_ratio and dE for superelastic processes
  CALL pkGetReverseCoef( 'g_ratio', g2_g1, NfKea+1 )
  CALL pkGetReverseCoef( 'dE', dE, NjKea+1 )
  ! - Computes superelastic cross sections
  DO i = NfKea + 1, NKea
    idE = NINT(dE(i)/Ebin); jmax = NEbin-idE
    FORALL( j = 1:jmax ) &
      CrossSec(j,i) = (1.d0+dE(i)/E(j)) / g2_g1(i) &
      * CrossSec(j+idE,Nindex(i))
```

```
  END DO
END IF
```

pkGetSources

Name

`pkGetSources` — Get the source terms for the continuity equations and the relative contribution of each process

Description

The subroutine `pkGetSources` evaluates the collisional source and loss terms of the continuity equations for each species. The relative contribution of each reaction can also be evaluated.

Syntax

```
pkGetSources( LDensity, SrcC[, SrcP][, Ratios][, locus][, STAT] )
```

Name	Description	Data type and attributes
`LDensity(NnTV)`	Local density of species with variable concentration	REAL(double), INTENT(IN)
`SrcC(NnTV)`	Collisional gain term of continuity equations	REAL(double), INTENT(OUT)
`SrcP(NnTV)`	Collisional loss term of continuity equations divided by `LDensity`	REAL(double), OPTIONAL, INTENT(OUT)
`Ratios(NKgas,NnTV)`	Percent contribution of each reaction to the collisional source and loss terms of each species	REAL(double), OPTIONAL, INTENT(OUT)
`locus`	Identifies whee the reactions take place	CHARACTER(10), OPTIONAL, INTENT(IN)

Name	Description	Data type and attributes
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

The gain term is in $L^{-3}s^{-1}$ where L^{-3} is the unit used to indicate the gas density in the data file; the loss term is in s^{-1}.

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Comments

If `locus` is not present or is equal to **`'gas'`** the reactions are in the gas. Otherwise the reactions are on a surface and the value of `locus` identifies the surface. This allows having different reactions in different surfaces.

Examples

Example E-9. `pkGetSources`: Computing the source terms in a 1D discharge taking account of gas temperature gradients.

```
INTEGER                  :: i, Nxp
REAL(double)             :: Tgas(Nxp)
REAL(double), ALLOCATABLE :: Dens(:,:), SrcC(:,:), SrcP(:,:)
...
ALLOCATE( Dens(NnTV,Nxp), SrcC(nnTV,Nxp), SrcP(NnTV,Nxp) )
...
DO i = 1, Nxp
  CALL pkSetValue( GasTemp=Tgas(i) )
  CALL pkGetSources( Dens(:,i), SrcC(:,i), SrcP(:,i) )
END DO
```

pkGetValue

Name

pkGetValue — Get the values of global properties

Description

The subroutine pkGetValue returns the gas temperature, electron temperature and total gas density.

Syntax

```
pkGetValue( [GasTemp][[,] eTemp][[,] GasN][[,] STAT] )
```

Name	Description	Data type and attributes
GasTemp	The gas temperature	REAL(double), OPTIONAL, INTENT(OUT)
eTemp	The electron temperature	REAL(double), OPTIONAL, INTENT(OUT)
GasN	The total gas density	REAL(double), OPTIONAL, INTENT(OUT)
STAT	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

Values are returned using the units indicated in the input file for these properties. If for a given property no units were explicitly indicated, the corresponding value is returned in S.I. units.

STAT = **0**: no errors; STAT = **n**: error number.

Examples

Example E-10. `pkGetValue`: Inquiring the gas temperature and density.

```
REAL(double) :: gasT, gasDens
...
CALL pkGetValue( gasT, GasN=gasDens )
```

`pkIsPhotoElec`

Name

`pkIsPhotoElec` — Tests if a photon produces photoelectric emission on a given surface

Description

The function `pkIsPhotoElec` is used to test if a given photon induces photoelectron emission in a given surface. Additionally it also returns the index of the parent species. The main use of this function is in computing the photoelectron emission from the volume distribution of excited species.

Syntax

```
pkIsPhotoElec( PhotN, Surf, NParent, KParent )
```

Name	Description	Data type and attributes
`PhotN`	Photon index	INTEGER(IN)
`Surf`	Surface name	CHARACTER(len=*), INTENT(IN)
`NParent`	Index of parent species	INTEGER, INTENT(OUT)
`KParent`	Index of photoelectric emission reaction	INTEGER, INTENT(OUT)

Results

The function returns a LOGICAL value.

Using a name to identify the surface allows to compute the photoelectron emission only in selected surfaces and to use different values of the photoelectron emission coefficient in different surfaces.

Examples

Example E-11. `pkIsPhotoElec`: Computing the photoelectron emission in a 1D discharge model.

In this example the matrix `MPhot` represents the probability of transmission of radiation from a plane `i`, perpendicular to the surface, to a surface "slice" `j`:

```
INTEGER      :: dx, NParent, KParent, iPh, i
REAL(double) :: MPhot(Nxp,Nxp), PhotElectrons(Nxp), KYield
...
PhotElectrons(:) = 0.d0
DO iPh = 1, NPhot
  IF( pkIsPhotoElec( iPh, 'wall', NParent, KParent ) ) THEN
    CALL pkGetReacCoef( 'Value', KYield, KParent )
    DO i = 1, Nxp
      DO j = 1, Nxp
        dx = ABS(j-i)+1
        PhotElectrons(i) = PhotElectrons(i) + &
          KYield * Dens(j,KParent) * MPhot(i,dx)
      END DO
    END DO
  END IF
END DO
```

`pkReadBaseData`

Name

`pkReadBaseData` — Reads the basic data in the input file

Description

The subroutine `pkReadBaseData` reads in the input file the **NAMELIST** instructions characterizing base properties as gas temperature, density or pressure and electron temperatures.

Syntax

```
pkReadBaseData( UNIT[, IOSTAT] )
```

Name	Description	Data type and attributes
`UNIT`	Unit number for the input file	INTEGER, INTENT(IN)
`IOSTAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

`IOSTAT` = **`0`**: no errors; `IOSTAT` = **`n`**: error number.

Comments

The calling program is responsible by opening and closing the input file.

Examples

Example E-12. `pkReadBaseData`: Reading the basic data in the input file.

```
OPEN( UNIT=10, FILE='TestData.txt', ACTION='READ' )
CALL pkReadBaseData( 10 )
```

pkReadChemReactions

Name

`pkReadChemReactions` — Reads only the chemical reactions in the input file

Description

The subroutine `pkReadChemReactions` reads in the input file the **NAMELIST** instructions characterizing the chemical reactions and process this information. The reactions are counted, classified, reactions involving vibrational series are split into individual reactions; the reactions are sorted according to the classification of Figure F-3; temperature dependent and reverse reactions rates are evaluated; branching ratios for cascade reactions are computed; etc.

Syntax

```
pkReadChemReactions( UNIT[, IOSTAT] )
```

Name	Description	Data type and attributes
`UNIT`	Unit number for the input file	INTEGER, INTENT(IN)
`IOSTAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

`IOSTAT` = **`0`**: no errors; `IOSTAT` = **`n`**: error number.

Comments

The calling program is responsible by opening and closing the input file.

Examples

Example E-13. `pkReadChemReactions`: Reading the chemical reactions in the input file.

```
OPEN( UNIT=10, FILE='TestData.txt', ACTION='READ' )
CALL pkReadChemReactions( 10 )
```

`pkReadData`

Name

`pkReadData` — Reads all data in the input file

Description

The subroutine `pkReadData` reads in the input file all the **NAMELIST** instructions characterizing the data handled by *PLASMAKIN* and process these informations. `pkReadData` is a wrapper routine calling the other data reading routines.

Syntax

```
pkReadData( UNIT[, IOSTAT] )
```

Name	Description	Data type and attributes
`UNIT`	Unit number for the input file	INTEGER, INTENT(IN)
`IOSTAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

`IOSTAT` = **`0`**: no errors; `IOSTAT` = **`n`**: error number.

Comments

The calling program is responsible by opening and closing the input file.

Examples

Example E-14. `pkReadData`: Reading the input file.

```
OPEN( UNIT=10, FILE='TestData.txt', ACTION='READ' )
CALL pkReadData( 10 )
```

`pkReadSpecies`

Name

`pkReadSpecies` — Reads only the species in the input file

Description

The subroutine `pkReadSpecies` reads in the input file the **NAMELIST** instructions characterizing the chemical species and process this information. The species are counted; classified; vibrational series are split into individual species and species concentration calculated; the species are sorted according to the classification of Figure F-2; etc.

Syntax

```
pkReadSpecies( UNIT[, IOSTAT] )
```

Name	Description	Data type and attributes
`UNIT`	Unit number for the input file	INTEGER, INTENT(IN)
`IOSTAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

`IOSTAT` = **`0`**: no errors; `IOSTAT` = **`n`**: error number.

Comments

The calling program is responsible by opening and closing the input file.

Examples

Example E-15. `pkReadSpecies`: Reading the basic data in the input file.

```
OPEN( UNIT=10, FILE='TestData.txt', ACTION='READ' )
CALL pkReadSpecies( 10 )
```

`pkSetPhoton`

Name

`pkSetPhoton` — Sets local photon density

Description

PLASMAKIN does not evaluate photon transport. However to compute reactions involving photons such as the photoelectric emission from surfaces of photoabsorption, the local photon density must be known. This routine allows setting the photon density values.

Syntax

```
pkSetPhoton( PhotDensity, NPhSrc[, STAT] )
```

Name	Description	Data type and attributes
`PhotDensity(NPhSrc)`	Photon density values	REAL(double), INTENT(IN)
`NPhSrc`	Size of `PhotDensity`	INTEGER, INTENT(IN)
`STAT`	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

`STAT` = **`0`**: no errors; `STAT` = **`n`**: error number.

Examples

Example E-16. `pkSetPhoton`: Setting the photon density prior to computing the source terms for other species.

```
INTEGER :: Nxp, i
REAL(double), ALLOCATABLE :: Dens(:,:), SrcC(:,:), SrcP(:,:), PhDens(:,:)
...
ALLOCATE( Dens(NnTV,Nxp), SrcC(NnTV,Nxp), &
  SrcP(NnTV, Nxp), PhDens(NPhSrc,Nxp) )
...
CALL COMPUTE_PH_TRANSPORT( PhDens)    ! User supplied routine
DO i = 1, Nxp
  CALL pkSetPhoton( PhDens(:,i), NPhSrc )
  CALL pkGetSources( Dens(:,i), SrcC(:,i), SrcP(:,i) )
END DO
```

pkSetReacCoef

Name

pkSetReacCoef — Sets the values of reaction properties

Description

The subroutine pkSetReacCoef allows setting the values of selected reaction properties. The properties that can be set are **'reverse'** and **'value'**. This allows to use user defined expressions for the reaction coefficients and is particularly useful for reactions involving vibrational levels.

Syntax

```
pkSetReacCoef( Property, Value, KReacN[, STAT] )
```

Name	Description	Data type and attributes
Property	Case sensitive name of property	CHARACTER(len=*), INTENT(IN)
{*Value* | *Value(:)*}	Value(s) of property	{LOGICAL | REAL(double)} , INTENT(IN)
KReacN	Starting index of reaction list	INTEGER, INTENT(IN)
STAT	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

STAT = **0**: no errors; STAT = **n**: error number.

Comments

The `Value` argument sets the value of the property inquired and can be a scalar or an array. In the last case the values set are the `SIZE(Value)` consecutive values of property.

Examples

Example E-17. `pkSetReacCoef`: Modifying the value of the 'reverse' property.

The following example is a continuation of Example E-8. In that example the superelastic electron collision cross sections were evaluated from the corresponding cross sections for the forward processes. Consequently we no longer need to consider those reactions as a reverse process and the `reverse` property is set to **FALSE**:

```
IF( NSKea > 0 ) THEN
  ... ! See code in pkGetReacCoef routine
  CALL pkSetReacCoef( 'reverse', (/(.FALSE.,i=1,NSKea)/), NdKea+1 )
END IF
```

Example E-18. `pkSetReacCoef`: Updating temperature dependent rate coefficients or coefficients defined through an external routine:

```
INTEGER :: NKaa
REAL(double), ALLOCATABLE :: Kvalue(:)
...
NKaa = NKgas - NKea
ALLOCATE( Kvalue(NKaa) )
...
CALL pkGetReacCoef( 'value', Kvalue, NKaa )
CALL pkSetValue( GasTemp=GasT )
CALL pkSetReacCoef( 'value' Kvalue, NKea+1 )
```

pkSetValue

Name

pkSetValue — Sets the values of global properties

Description

The subroutine pkSetValue sets the values of several properties: gas or electron temperature, total gas density and vibrational temperature for molecular species.

Syntax

```
pkSetValue( [GasTemp][[,] eTemp][[,] GasN][[,] N1byN0]
[[,] SpeciesN][[,] STAT] )
```

Name	Description	Data type and attributes
GasTemp	The gas temperature	REAL(double), OPTIONAL, INTENT(IN)
eTemp	The electron temperature	REAL(double), OPTIONAL, INTENT(IN)
GasN	The total gas density	REAL(double), OPTIONAL, INTENT(IN)
N1byN0(:)	The density ratio for the first two vibrational levels of molecular species	REAL(double), OPTIONAL, INTENT(IN)
SpeciesN	Starting index of species sequence for setting the vibrational temperature	INTEGER, OPTIONAL, INTENT(IN)
STAT	Execution status indicator	INTEGER, OPTIONAL, INTENT(OUT)

Results

STAT = **0**: no errors; STAT = **n**: error number.

Examples

Example E-19. `pkSetValue`: Setting the ratio of the number of molecules in the two first vibrational levels.

```
INTEGER :: Minvib, Nvib, dummy(NSpecies)
CHARACTER(20) :: SpcNames(NSpecies)
...
CALL pkGetParticle( 'name', SpcNames, 1 )
dummy(:) = INDEX( SpcNames(:), 'N2,v' )  ! Indexes of N2,v species
Minvib = MINLOC( dummy, dummy>0 ) - Nnc
Nvib = COUNT( dummy /= 0 )
CALL pkSetValue( N1byN0=SPREAD( Dens(Minvib+1)/Dens(Minvib), 1, Nvib), &
  Species=Minvib )
```

`SetRate`

Name

`SetRate` — User supplied routine to set the values of rate coefficients

Description

The subroutine `SetRate` sets the values of rate coefficients when the rate coefficient equation included in *PLASMAKIN* and indicated in chapter one is not appropriate. The user is free to write this routine but the routine must have the interface described bellow.

Syntax

```
SetRate( Idx, rIdx, pIdx, Temp, RateValue )
```

Name	Description	Data type and attributes

Name	Description	Data type and attributes
`Idx`	Index of desired rate coefficient equation in the routine	INTEGER, INTENT(IN)
`rIdx(NMaxReacSpc)`	Array with the vibrational quantum numbers of reactant species	INTEGER, INTENT(IN)
`pIdx(NMaxReacSpc)`	Array with the vibrational quantum numbers of product species	INTEGER, INTENT(IN)
`Temp`	The electron or gas temperature used in the rate coefficient equation	REAL(double), INTENT(IN)
`RateValue`	The rate coefficient value	REAL(double), INTENT(OUT)

6. Data Input Format

The data organization and the input file syntax were designed to provide as much freedom as possible to the user, to reduce to a minimum the amount of information needed in particular the information not directly related to the physical or chemical nature of the problem. At the same time the input file syntax was designed to enforce correction.

The following strategies were used:

- Exclusive use of **NAMELIST** groups for data input.
- Species and reactions are always initialized with default values. This means that only when the value of a variable is different from the default it must be explicitly included on the input file.
- Values as the total number of species, number of species per type, number of chemical reactions etc, are automatically evaluated by *PLASMAKIN*
- The syntax and data input routines allow the reading and processing of vibrational series and reactions involving vibrational series.
- All CHARACTER values are case sensitive. This facilitates the detection of spelling errors in chemical elements and units.

PLASMAKIN_DATA

Name

PLASMAKIN_DATA — Holds initial values of data applying to the entire system

Syntax

&PLASMAKIN_DATA [Cte_p = *Lvalue* [,]] { pressure = *Rvalue* [, *'Cvalue'*] [,]
gas_T = *Rvalue* [,*'Cvalue'*] | [,] gas_n = *Rvalue* [, *'Cvalue'*] }
[[,] electron_T = *Rvalue* [, *'Cvalue'*]] /

Name	Description	Data type and attributes	Default value	Units or values supported
Cte_p	Constant pressure system	LOGICAL	**.TRUE.**	**.T**, **T**, **.t**, **t**, **.F**, **F**, **.f**, **f**
Pressure	Total pressure	PHYS_PROPERTY	**0,'Pa'**	**'Pa'**, **'Nm-2'**, **'bar'**, **'mbar'**, **'torr'**, **'atm'**
Gas_T	Gas temperature	PHYS_PROPERTY	**300,'K'**	**'K'**, **'C'**
Gas_n	Total gas density	PHYS_PROPERTY	**0,'m-3'**	**'m-3'**, **'cm-3'**
electron_T	Electron temperature	PHYS_PROPERTY	**0,'eV'**	**'eV'**, **'K'**, **'C'**

Comments

This **NAMELIST** is usually used only once in the data input file. The Cte_p value is used to distinguish between constant pressure and constant volume conditions. The initial gas density can be indicated explicitly - specifying its value - or implicitly indicating the pressure and gas temperature. However even in the later case the user can be interested in indicate the gas density units. The electron temperature is only necessary if electron temperature dependent rate coefficients are used. Note that °C is just written C.

Examples

Example E-20. PLASMAKIN_DATA: Examples of PLASMAKIN_DATA NAMELIST syntax.

```
&PLASMAKIN_DATA Gas_n = 1.e+22 /                                    ❶

&PLASMAKIN_DATA                                                     ❷
  Gas_n    =   ,'cm-3'
  Pressure =  1,'mbar'
  Gas_T    = 25,'C'
/

&PLASMAKIN_DATA Cte_p = F, Pressure = 1,'torr', electron_T = 3,'eV' /   ❸
```

❶ Explicit indication of gas density using the default units (m^{-3}). Gas temperature has also the default value (300 K).

❷ The density is calculated by *PLASMAKIN* from the pressure and temperature values but the value of the total density passed to the calling program is in cm^{-3}.

❸ Constant volume conditions. Electron rate coefficients are evaluated with an initial electron temperature of 3 eV.

CHEM_SPECIES

Name

`CHEM_SPECIES` — Read chemical species and species properties

Syntax

&CHEM_SPECIES name = *'Cvalue'* [,] [constant = *Lvalue*] [,] [charge = *Rvalue*] [,] [mass = *Rvalue*] [,] [g = *Ivalue*] [,] [energy = *Rvalue* [,*'Cvalue'*]] [,] [heatC = *Rvalue*] [,] [K_T = *Rvalue*] [,] [v = *Ivalue(2)*] [,] [omega = *Rvalue* [,*'Cvalue'*]] [,] [vib_T = *Rvalue* [,*'Cvalue'*]] [,] [anharmonicity = *Rvalue*] [,] [rotational_cte = *Rvalue*] [,] [mpolar = *Rvalue* [,*'Cvalue'*]] [,] [initial_conc = *Rvalue* [,*'Cvalue'*]] [,] [num_scheme = *Ivalue*] [,] [cascade = *Lvalue*] [,] [data_file = *'Cvalue'*,*Ivalue*] /

Name	Description	Data type and attributes	Default value	Units or values supported
`name`	Name of species	CHARACTER(20)	**''**	
`constant`	Is the species concentration constant?	LOGICAL	**.FALSE.**	**.F.**, **F**, **.f.**, **f**, **.T.**, **T**, **.t.**, **t**
`charge`	Electrical charge in elementary charge units	INTEGER	**0**	
`mass`	Mass in a.m.u.	REAL(8)	**0**	
`g`	Level multiplicity	INTEGER	**1**	
`energy`	Level energy or standard molar enthalpy	PHYS_PROPERTY	**0,'eV'**	**''**, **'eV'**, **'kJ/mol'**, **'kJmol-1'**, **'kcal/mol'**, **'kcalmol-1'**
`heatC`	Heat capacity (c_p or c_v) in kJ/mol/k	REAL(8)	**0**	
`K_T`	Thermal conductivity	REAL(8)	**0**	
`v`	Range of quantum vibrational numbers	INTEGER	**0,0**	
`omega`	Vibrational frequency	PHYS_PROPERTY	**0,'cm-1'**	**''**, **'cm-1'**, **'s-1'**
`vib_t`	Vibrational temperature	PHYS_PROPERTY	**0,'K'**	**'K'**, **'C'**
`anharmonicity`	Anharmonicity parameter	REAL(8)	**0**	

Name	Description	Data type and attributes	Default value	Units or values supported
`rotational_cte`	Rotational constant	REAL(8)	**`0`**	
`mpolar`	Dipolar or quadrupolar momentum	PHYS_PROPERTY	**`0,"`**	**`"`**, **`'ea_o'`**, **`'ea_o2'`**
`initial_conc`	Species initial concentration	PHYS_PROPERTY	**`0,"`**	**`"`**, **`'%'`**
`num_scheme`	Index of an users' defined algorithm to evaluate species properties	INTEGER	0	
`cascade`	Is species a cascade level?	LOGICAL	**`.FALSE.`**	**`.F`**, **`F`**, **`.f`**, **`f`**, **`.T`**, **`T`**, **`.t`**, **`t`**
`data_file`	Data file with further informations	DATA_COLUMN	**`",0`**	

Comments

The following rules must be obeyed:

- All species must have a name. This name is used to identify the species in chemical reactions. Although nothing hinders the use of custom names, it is advisable to follow the chemical nomenclature with the limitation that under- or superscript numbers must be written inline. Names are case sensitive.
- The name for electrons *must* be the letter **`'e'`**.
- The names of photons must contain the word **`'photon'`**.
- A vibrational series is identified by a value of $v(2) > 0$. The name of each species in the series is formed by replacing the last closing brace in the series name by suffix **`'=n)'`** where n is the corresponding vibrational number. The range of vibrational levels in the series is defined by the values of the array v. The values of v must follow the order $v(1) \leqslant v(2)$. For a single vibrational level only the first value of v must be indicated.

Examples

Example E-21. CHEM_SPECIES: Example of CHEM_SPECIES NAMELIST syntax.

```
&CHEM_SPECIES                                                          ❶
name = 'Ne', constant = T, mass = 20.18, initial_conc = 100,'%' /

&CHEM_SPECIES                                                          ❷
  name = 'Ne(3P2)', energy = 16.61, g = 5,
  data_file = 'NeTransp.txt',1
/

&CHEM_SPECIES name = 'Ar(3p)', cascade = T /                           ❸

&CHEM_SPECIES name = 'Ne+', charge = 1, num_scheme = 2 /               ❹

&CHEM_SPECIES name = 'photon 1' /                                      ❺

&CHEM_SPECIES name = 'e', data_file = 'eTransp.txt' /                  ❻

&CHEM_SPECIES                                                          ❼
  name          = 'O2(X,v)'
  mass          = 32.
  v             = 0,15
  omega         = 1580.19,'cm-1'
  vib_T         = 2000
  anharmonicity = 7.58e-3
  initial_conc  = 5,'%'
/
```

❶ Dominant species. The value of *`constant`* must be indicated; *`mass`* must be indicated if the chemical kinetics model includes reverse reactions.

❷ Neon excited level; the *`energy`* (in eV) and *`g`* values will be used in evaluating superelastic rates. The external *`data_file`* is used to hold the value of the diffusion coefficient.

❸ This level will be handled as a cascade level.

❹ As species is an ion the *`charge`* value must be indicated. The index of an user defined algorithm is also indicated.

❺ Photons produced on radiative transitions and involved on radiation trapping or any other process must be listed and identified by a name containing the word **`'photon'`**.

❻ Electrons have the mandatory name **'e'**. The *data_file* can contain the values for the drift velocity and diffusion coefficient.

❼ A vibrational series (from *v*=**0** to *v*=**15**). The names of the corresponding species are **'O2(X,v=0)'**,..., **'O2(X,v=15)'**. The initial populations are computed from the values of the vibrational frequency, vibrational temperature and anharmonicity.

CHEM_REACTION

Name

CHEM_REACTION — Read a chemical reaction or reactions involving vibrational series

Syntax

&CHEM_REACTION [locus = *'Cvalue'*] [,] [reactants = *'Cvalue'* [[,]...]] [,] [products = *'Cvalue'* [[,]...]] [,] [comment = *'Cvalue'*] [,] [opposing = *Lvalue*] [,] [ee_loss = *Rvalue*] [,] {value = *Rvalue* [[,]...] | data_file = *'Cvalue',Ivalue*} [,] [units = *'Cvalue'*] /

Name	Description	Data type and attributes	Default value	Units or values supported
locus	Where the reaction takes place	CHARACTER(10)	**'gas'**	**'gas', 'user_string'**
reactants	Reactant species	CHARACTER(20)	**NMaxReacSpc*"**	
products	Product species	CHARACTER(20)	**NMaxReacSpc*"**	
comment	Relation between vibrational quantum numbers	CHARACTER(20)	**"**	

Name	Description	Data type and attributes	Default value	Units or values supported
`opposing`	The **NAMELIST** describes both forward and reverse reactions	LOGICAL	**`.FALSE.`**	**`.F`**, **`F`**, **`.f`**, **`f`**, **`.T`**, **`T`**, **`.t`**, **`t`**
`ee_loss`	Energy lost in the reaction by the electron	REAL(8)	**`0`**	
`units`	Units used for reaction coefficient	CHARACTER(10)	**`”`**	**`”`**,**`’s-1’`**,**`’m3s-1’`**, **`’cm3s-1’’m6s-1’`**,**`’cm6s-1’`**
`value`	Reaction rate equation coefficients	REAL(8)(12)	**`(0, i=1,12)`**	
`data_file`	Index of function to compute the reaction rate in the user supplied `SetRate` routine.	DATA_COLUMN	**`”,0`**	

Comments

The following rules must be followed:

- If the *`locus`* of a reaction in the gas is indicated, this value *must* be **`’gas’`**. For surface reactions any name can be used and reactions in different surfaces can be identified by different *`locus`* values.
- All species indicated in the *`reactants`* and *`products`* lists must be described in a CHEM_SPECIES **NAMELIST**.
- Reactions involving vibrational species can be collectively represented indicating the generic name representing the vibrational series. The vibrational species involved in the reaction can be selected through relational expressions involving the vibrational quantum numbers. These expressions can be included in the species name in the *`reactants`* or *`products`* fields or, if they involve two different vibrational quantum numbers, in the *`comment`* field. The vibrational quantum numbers in these expressions can only be represented by the letters **`v`** or **`w`**.

- The following classes of relational expressions can be used:

  a. If the reaction corresponds to a fixed variation of the vibrational quantum number, the relation between the initial and the final values can be translated into the names of the vibrational species in the *reactants* and *products*. In this case the arithmetic operators + and - can be used in the species names.

  b. If the range of vibrational levels is defined by a relation involving only one vibrational number, this relation can be included in the vibrational series name.

  c. If the expression used to select the vibrational levels involves two vibrational numbers, this relation must be written in the *comment* filed.

  In case these two last cases the relations can be written using the logical operators $<$, $\leq$, =, $\geq$ and $>$. In the last case the modulo operator can also be used.

- *value* is an array of 12 elements allowing representing the 6 pair of α^j and β^j values needed to represent the rate coefficient using the equation indicated in chapter one.

- Finally, if the vibrational rate coefficients must be computed from a table or analytically but the equation in chapter one is not adequate, the user can supply the external routine `SetRate` with user defined rate equations. The **index** value in the *data_file* field is used to selection of the appropriate rate equation. In this case the **name** value is not used.

> **Caution: Indistinguishable reactions**
>
> For a V-V reaction involving two vibrational levels of the same molecule the direct and reverse reactions are indistinguishable. In this case the *opposing* property should not be set to **T**. Otherwise the number of reactions is counted twice.

Examples

Example E-22. CHEM_REACTION: Example of CHEM_REACTION NAMELIST syntax.

```
&CHEM_REACTION                                                    ❶
  reactants = 'Ne(3P1)', products = 'Ne','photon 1',
  value = 0.486e8 /

&CHEM_REACTION                                                    ❷
  reactants = 'Ne(3P2)','Ne(3P1)', products = 'Ne','Ne+','e',
  value = 3.2e-10, units = 'cm3s-1' /
```

```
&CHEM_REACTION                                                          ❸
  reactants = 'e','Ne', products = 'e','Ne(3P2)',
  units = 'cm3s-1', data_file = 'Ne_data.txt',3 /

&CHEM_REACTION                                                          ❹
  reactants = 'e','Ne(3P2)', products = 'e','Ne(3P1)',
  value = 1.603e-6, -0.3, -6.e2, 1, units = 'cm3s-1',
  opposing = T /

&CHEM_REACTION                                                          ❺
  reactants = 'N2(X,v+1)','O2(X,w)', products = 'N2(X,v)','O2(X,w+1)'
  data_file = "",1 /

&CHEM_REACTION                                                          ❻
  reactants = 'Ne','N2(X,v<=10)', products = 'Ne,2*'N'
  data_file = "",2 /

&CHEM_REACTION                                                          ❼
  reactants = 'N2(X,v)','M', products = 'N2(X,w)','M'
  comment = '|v-w|>=1', data_file = "",3 /
```

❶ Radiation emission; the value of *value* is the transition probability in s^{-1}.

❷ Penning ionisation with constant rate coefficient.

❸ Electron excitation. The values of the rate coefficient are read in the external file **Ne_data.txt** and identified by the index **3**.

❹ Both the direct and reverse reactions are taken into account. The rate coefficient for the direct reaction has an Arrhenius temperature dependence with the above coefficients. The rate coefficient for the reverse reaction are automatically computed.

❺ Group of reactions on two vibrational series. The syntax for the relations between the vibrational quantum numbers corresponds to case a. above.

❻ Group of reactions involving a vibrational series. The syntax corresponds to case b. above.

❼ Group of reactions involving a vibrational series. the syntax corresponds to case c. above.

References

[Carver97] “ASAD”, G. D. Carver, P. D. Brown, and O. Wild *Comp. Phys. Commun.* 105 (1997) 197

[Gordiets72] B. F. Gordiets, S. S. Mamedov, and L. A. Shelepin *JEPT* 40 (1972) 640-646

[Kee96] “CHEMKIN-III”, R. J. Kee, F. M. Rupley, E. Meeks, and J. A. Miller Sandia National Laboratories (1996)

[Lindemann22] F. Lindemann *Trans. Faraday Soc.* 17 (1922) 598

[Molisch98] A. F. Molisch and B. P. Oehry *Radiation Trapping in Atomic Vapours* Oxford Univ. Press (1998)

[Pinhão01] N. R. Pinhão *Comp. Phys. Commun.* 135(1) (2001) 105-131

[Troe83] P. H. Gilbert, K. Luther, and J. Troe *Ber. Buns. Phys. Chem.* 87 (1983) 169

Appendix A. Error Messages

The number of error messages issued by *PLASMAKIN* is relatively small - twenty-seven - and in the present version no attempt has been made to cover all possible error situations. Most of the messages are related with the reading and processing of the input file where it is easy to make typographical mistakes. This is an area where improvement can be expected in next versions of the library.

The error messages have the structure *PLASMAKIN*:<classification and number>: <description>

Error messages and description

PLASMAKIN: severe(1): NAMELIST "PLASMAKIN_DATA" not found

PLASMAKIN: severe(2): NAMELIST "CHEM_SPECIES" not found

PLASMAKIN: severe(3): NAMELIST "CHEM_REACTION" not found

PLASMAKIN: severe(4): Syntax error in "CHEM_SPECIES" NAMELIST #

PLASMAKIN: severe(5): Syntax error in "CHEM_REACTION" NAMELIST #

PLASMAKIN: severe(6): Too many values for NAMELIST variable

PLASMAKIN: severe(7): Invalid reference to variable in NAMELIST input

PLASMAKIN: severe(8): NAMELIST I/O not consistent with OPEN options

PLASMAKIN: severe(9): Invalid NAMELIST input format

PLASMAKIN: severe(10): REWIND error

PLASMAKIN: error(11): Undetermined I/O error

PLASMAKIN: severe(12): Error in physical units or units not supported

PLASMAKIN: error(13): Total sum of species initial concentration is not 1

PLASMAKIN: severe(14): Attempt to re-read chemical species list

PLASMAKIN: severe(15): Attempt to re-read reaction list

PLASMAKIN: severe(16): Unknown species in reaction #

PLASMAKIN: severe(17): Violation of charge conservation in reaction #

PLASMAKIN: severe(151): Allocatable array is already allocated

PLASMAKIN: severe(727): Cannot ALLOCATE allocatable array - out of memory

PLASMAKIN: error(20): Undetermined memory error

PLASMAKIN: error(21): Syntax error in call to routine "pkSetValue"

PLASMAKIN: error(22): Syntax error in call to routine "pkSetPhoton"

PLASMAKIN: error(23): Syntax error in call to routine "pkGetSources"

PLASMAKIN: error(24): Syntax error in call to routine "pkGetReacCoef"

PLASMAKIN: warning(25): Interrogation on unknown property

PLASMAKIN: warning(26): Unable to set property of rate coefficient

PLASMAKIN: warning(27): Attempt to set the value of T-dependent rate coefficient

Appendix B. GNU Free Documentation License

Copyright © 2000 by Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher

of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

L. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

M.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the

Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.