FAC 1.1.1 Manual

M. F. Gu*

---

* gu2@llnl.gov

Contents

1 Overview **12**

1.1 What Is FAC 12

1.2 Obtain and Install FAC 12

1.3 Quick Start 13

1.3.1 SFAC Interface 13

1.3.2 PFAC Interface 13

2 Description of Output Files **17**

2.1 Binary Format 17

2.1.1 `F_HEADER` 17

2.1.2 `EN_HEADER` 18

2.1.3 `EN_RECORD` 18

2.1.4 `TR_HEADER` 20

2.1.5 `TR_RECORD` 20

2.1.6 `TR_EXTRA` 21

2.1.7 `CE_HEADER` 21

2.1.8 `CE_RECORD` 22

2.1.9 `RR_HEADER` 24

2.1.10 `RR_RECORD` 25

2.1.11 `AI_HEADER` 26

2.1.12 `AI_RECORD` 26

2.1.13 `CI_HEADER` 27

2.1.14 `CI_RECORD` 28

2.1.15 `SP_HEADER` 29

2.1.16 `SP_RECORD` 30

2.1.17 `SP_EXTRA` 30

2.1.18 `RT_HEADER` 30

2.1.19 `RT_RECORD` 32

2.1.20 `DR_HEADER` 32

2.1.21 `DR_RECORD` 33

2.1.22 `AIM_HEADER` 34

2.1.23 `AIM_RECORD` 34

2.1.24 `CIM_HEADER` 35

2.1.25 `CIM_RECORD` 36

2.2 ASCII Format 36

2.2.1 `DB_EN` 37

2.2.2 `DB_TR` 37

2.2.3 `DB_CE` 38

2.2.4 `DB_RR` 40

2.2.5 `DB_AI` 41

2.2.6 `DB_CI` 41

2.2.7 `DB_SP` 43

2.2.8 `DB_RT` 44

2.2.9 `DB_DR` 46

2.2.10 `DB_AIM` 46

2.2.11 `DB_CIM` 47

3 FAC Function Reference **48**

3.1 `fac`–Core FAC Module 48

3.1.1 Variables 48

ATOMICMASS 48

ATOMICSYMBOL 48

QKMODE 48

VERSION 48

3.1.2 Functions 49

AIBranch 49

AITable 49

AITableMSub 49

AppendTable 49

Asymmetry 49

AvgConfig 49

BasisTable 49

CECross 50

CERate 50

CETable 50

CETableEB 50
CETableMSub 50
CITable 50
CITableMSub 50
CheckEndian 50
ClearLevelTable 50
ClearOrbitalTable 50
CloseSFAC 50
Closed 50
Config 51
ConfigEnergy 51
ConvertToSFAC 51
CorrectEnergy 51
CutMixing 51
DROpen 51
GetCFPOld 52
GetCG 52
GetConfigNR 52
GetPotential 52
GetW3j 52
GetW6j 52
GetW9j 52
Info 52
InterpCross 52
JoinTable 52
LevelInfor 53
ListConfig 53
MaxwellRate 53
MemENTable 53
ModifyPotential 53
OptimizeRadial 53
PICrossH 53
PrepAngular 53
Print 53
PrintTable 54

PropogateDirection 54
RecStates 54
RefineRadial 54
Reinit 54
ReinitConfig 54
ReinitDBase 54
ReinitExcitation 55
ReinitIonization 55
ReinitRadial 55
ReinitRecombination 55
ReinitRecouple 55
ReinitStructure 55
RestorePotential 55
RMatrixBasis 55
RMatrixBoundary 55
RMatrixCE 56
RMatrixConvert 56
RMatrixExpansion 56
RMatrixFMode 56
RMatrixNBatch 56
RMatrixNMultipoles 56
RMatrixSurface 56
RMatrixTargets 56
RRCrossH 57
RRMultipole 57
RRTable 57
SavePotential 57
SetAICut 57
SetAngZCut 57
SetAtom 57
SetBoundary 57
SetBreit 57
SetCEBorn 58
SetCEGrid 58
SetCEGridLimits 58

SetCELCB 58
SetCELMax 58
SetCELQR 58
SetCEQkMode 58
SetCIEGrid 58
SetCIEGridLimits 58
SetCILCB 58
SetCILevel 59
SetCILMax 59
SetCILMaxEject 59
SetCILQR 59
SetCITol 59
SetCIQkMode 59
SetFields 59
SetHydrogenicNL 59
SetIEGrid 59
SetMaxRank 59
SetMixCut 59
SetMS 59
SetNStatesPartition 59
SetOptimizeControl 60
SetOptimizeMaxIter 60
SetOptimizePrint 60
SetOptimizeStabilizer 60
SetOptimizeTolerance 60
SetPEGrid 60
SetPEGridLimits 60
SetRadialGrid 60
SetRRTEGrid 60
SetRecPWLimits 60
SetRecPWOptions 60
SetRecQkMode 60
SetRecSpectator 61
SetScreening 61
SetSE 61

SetSlaterCut 61
SetTEGrid 61
SetTransitionCut 61
SetTransitionGauge 61
SetTransitionMaxE 61
SetTransitionMaxM 61
SetTransitionMode 61
SetTransitionOptions 61
SetUsrCEGrid 61
SetUsrCIEGrid 62
SetUsrPEGrid 62
SetUTA 62
SetVP 62
SlaterCoeff 62
Structure 62
Structure 62
StructureEB 63
StructureMBPT 63
StructureMBPT 63
StructureMBPT 63
StructureMBPT 63
StructureMBPT 63
TotalCICross 63
TotalPICross 64
TotalRRCross 64
TransitionMBPT 64
TransitionMBPT 64
TransitionTable 64
TRBranch 64
TRTable 64
TRTableEB 64
TRRateH 64
WaveFuncTable 64
Y5N 65
3.2 `crm`–Collisional Radiative Model 65

3.2.1 Functions 65
AddIon 65
Cascade 65
CBeli 65
CFit 65
CheckEndian 65
CloseSCRM 65
ColFit 66
ConvertToSCRM 66
DRBranch 66
DRFit 66
DRStrength 66
DumpRates 67
EBeli 67
EColFit 67
EleDist 67
EPhFit 67
FracAbund 67
InitBlocks 67
IonDensity 68
Ionis 68
LevelPopulation 68
MaxAbund 68
NormalizeMode 68
NDRFit 68
NRRFit 68
PhFit 68
PhoDist 68
PlotSpec 68
Print 69
PrintTable 69
RateTable 69
RBeli 69
Recomb 69
ReinitCRM 69

RRFit . . . 69
RRRateH . . . 69
SelectLines . . . 70
SetAIRates . . . 70
SetAIRatesInner . . . 70
SetAbund . . . 70
SetBlocks . . . 70
SetCERates . . . 70
SetCIRates . . . 70
SetCascade . . . 70
SetEleDensity . . . 70
SetEleDist . . . 70
SetExtrapolate . . . 71
SetInnerAuger . . . 71
SetIteration . . . 71
SetNumSingleBlocks . . . 71
SetPhoDensity . . . 71
SetPhoDist . . . 71
SetRRRates . . . 71
SetRateAccuracy . . . 71
SetTRRates . . . 71
SetUTA . . . 71
SpecTable . . . 71
TwoPhoton . . . 72
3.3 `pol`–Line Polarizations . . . 72
3.3.1 Functions . . . 72
CloseSPOL . . . 72
ConvertToSPOL . . . 72
Orientation . . . 72
PolarizationTable . . . 72
PopulationTable . . . 72
Print . . . 72
SetDensity . . . 72
SetEnergy . . . 72
SetIDR . . . 73

SetMIteration 73
SetMaxLevels 73
SetMAIRates 73
SetMCERates 73
SetMLevels 73
3.4 `util`–Utility Functions 73
3.4.1 Functions 73
Spline 73
Splint 73
UVIP3P 73
3.5 `config`–Electronic Configuration Specification 73
3.6 `const`–Physical Constants 74
3.7 `table`–Text Tabulation 74
3.7.1 Format of Text Table 74
3.7.2 Class Attributes and Methods 75
fname 75
title 75
authors 75
date 75
separator0 75
separator 75
add_column 76
open 76
close 76
write_header 76
write_row 76
read_header 76
read_columns 76
convert2tex 76
rewind 76
3.7.3 Example 76
3.8 `atom`–Application of `fac` Module 78
atomic_data 78
3.9 `spm`–Application of `crm` Module 78

4 Frequently Asked Questions (FAQ) To FAC 79

4.1 General 79

4.2 Atomic Structure 80

4.3 Collisional Excitation 81

4.4 Photoionization and Radiative Recombination 81

4.5 Autoionization and Dielectronic Recombination 82

4.6 Collisional Ionization 82

4.7 Collisional Radiative Model 82

1 Overview

1.1 What Is FAC

FAC stands for The Flexible Atomic Code. It is an integrated software package to calculate various atomic radiative and collisional processes, including energy levels, radiative transition rates, collisional excitation and ionization by electron impact, photoionization, autoionization, radiative recombination and dielectronic capture. The package also includes a collisional radiative model to construct synthetic spectra for plasmas under different physical conditions.

The atomic structure calculation in FAC is based on the relativistic configuration interaction with independent particle basis wavefunctions. These basis wavefunctions are derived from a local central potential, which is self-consistently determined to represent electronic screening of the nuclear potential. Relativistic effects are fully taken into account using the Dirac Coulomb Hamiltonian. Higher order QED effects are included with Breit interaction in the zero energy limit for the exchanged photon, and hydrogenic approximations for self-energy and vacuum polarization effects. Continuum processes are treated in the distorted-wave (DW) approximation. Systematic application of the factorization-interpolation method of **?** makes the present code highly efficient for large scale calculations. The details of theoretical background and computational methods are not discussed in this manual, instead, they are described in a series of papers which are distributed along with this package and this manual.

FAC is a step forward to bring detailed atomic model accessable to a wide community of laboratory and astrophysical plasma diagnostics. Its flexible interface is designed to be useful even for people without a deep understanding of the underlying atomic theories. It is also powerful enough for experienced users to explore the effects of algorithmic choices and different physical approximations.

FAC is freely distributed in the hope that it will be useful. The author makes every effort to ensure its correctness. However, he does not guarantee its fitness to any specific purpose. The author is not responsible for any damage resulting from the use of this program, including failure to obtain or loss of tenure.

1.2 Obtain and Install FAC

The latest version of FAC is 1.1.1. It can be obtained from **http://sprg.ssl.berkeley.edu/ mfgu/fac**. I can also send a copy to you through email. Please request to **mfgu@ssl.berkeley.edu**. It is being continously developed at present, so please check regularly to get the newest version.

Much of the FAC package is written in ANSI C and Fortran 77. It should therefore work on any platform with a C and Fortran 77 compilers. However, this is only true to the rather simple command parser that comes with FAC, referred to as SFAC. The flexibility of FAC is realized when the Python interface (PFAC) is used. The numerical subroutines implemented in FAC are exported through several Python modules. The computation task can therefore be completed by programming in the scripting language Python. These python modules are compiled as shared objects, and are dynamically loaded. This primarily works under ELF systems, such as almost all modern Unix and Linux systems. It has also been tested to work under Mac OS X and Windows (In the case of Windows, the Unix API emulation by Cygwin is required, which is available at **www.cygwin.com**). To fully utilize the strength of FAC, it is strongly recommended that Python be installed, which can be obtained from **www.python.org**.

Step-by-step instructions for installation can be found in the README file in the top directory of FAC distribution.

1.3 Quick Start

1.3.1 SFAC Interface

The SFAC interface is basically a stripped down command interpreter modeled after Python syntax, with the omission of flow control freatures, such as conditional execution and loops. Therefore simple Python scripts may be converted to SFAC input files without difficulty. The Python interface actually contains functions to do the convertion automatically. Through out this manual, we mainly focus on the more useful Python interface. Most of the Python fuctions implemented in the extension modules are also available in SFAC interface with identical calling sequences. To use SFAC, one passes the input files to the 3 executables `sfac`, `scrm` and `spol` on the command line such as

```
sfac input.sf
scrm input.sf
```

or, one may invoke `sfac` and `scrm` without arguments, in which case, they read from `stdin` for inputs, where commands are interpreted line by line. The program `sfac` handles atomic calculations, `scrm` is used to construct collisional radiative spectral models, and `spol` is used to calculate line polarizations due to collisional excitation.

1.3.2 PFAC Interface

To use the PFAC interface, one needs to be familiar with the basics of Python scripting language. Python has excellent documentations that come with the standard distribution. It is an extremely well designed language to learn, and to use.

Perhaps, the quickest way to get familiar with FAC is to inspect the simple demo scripts in the `demo/` directory in FAC distribution. There are individual scripts and their SFAC conterparts demonstrating the calculation of energy levels, radiative transition rates, collisional excitation and ionization cross sections, radiative recombination cross sections and autoionization rates. There is also a more advanced example for the calculation of iron L-shell atomic data, and their application in the collisional radiative model.

In this section, we look into the details of one of these scripts, `demo/structure/fe17_structure.py` for the calculation of Ne-like iron energy levels and radiative transition rates between $n = 2$ and $n = 3$ complexes. The following is a duplication of that script.

```
 1: from pfac import fac

 2: fac.SetAtom('Fe')
 3: # 1s shell is closed
 4: fac.Closed('1s')
 5: fac.Config('2*8', group = 'n2')
 6: fac.Config('2*7 3*1', group = 'n3')

 7: # Self-consistent iteration for optimized central potential
 8: fac.ConfigEnergy(0)
# the configurations passed to OptimizeRadial should always
# be one or two of the lowest lying ones. If you need more highly
# excited levels, such as n=4, 5, 6, ..., do not put them into
# OptimizeRadial.
 9: fac.OptimizeRadial(['n2'])
10: fac.ConfigEnergy(1)
11: fac.Structure('ne.lev.b', ['n2', 'n3'])
12: fac.MemENTable('ne.lev.b')
13: fac.PrintTable('ne.lev.b', 'ne.lev', 1)
```

```
14: fac.TransitionTable('ne.tr.b', ['n2'], ['n3'])
15: fac.PrintTable('ne.tr.b', 'ne.tr', 1)
```

Line numbers are added for easy reference, they are not part of the script. As is evident from the above list, all functions implemented in the FAC extension modules have a naming convention of concatenated capitalized words. Line 1 imports the extension module `fac` from the package `pfac`. Alternatively, one could have used

```
from pfac.fac import *
```

then, all module qualifiers `fac.` in the following lines can be omitted. Line 2 set the atomic element to be iron. Line 3 is a comment, which starts with a `#`. Line 4–6 specifies the electronic configurations to be included in the calculation. The closed shells specified by the function `Closed` must be inactive in this calculation. In the `Config` functions, `2*8` stands for an $n = 2$ complexes with 8 electrons, while `2*7 3*1` stands for all configurations resulting from excitation of one electron from $n = 2$ to $n = 3$. For more possibilities in the specification of electronic configurations, one is referred to Chapter 3. Line 8–10 carries out a Dirac-Fock-Slater self-consistent calculation to derive a local central potential which represents the electronic screening of the nuclear potential. In this calculation, the potential is optimized to the average electron clouds of configurations `n2` and `n3`, since in FAC, all atomic processes are treated with basis wavefunctions generated from a single potential. This results in the potential to be less optimized for `n2` and `n3` individually. Lines 8 and 10 are used to make a crude correction to the resulting energy levels due to this effect. The first call to `ConfigEnergy(0)` will make individual optimization to all configuration groups. The average energy of each configuration group with these indivudually optimized potential is then calculated and stored. The second call to `ConfigEnergy(1)` will then recalculate the average energy of configuration groups under the potential taking into account all configuration groups. The difference between the two represents the effect of a less optimized potential, and are used to adjust the final energy levels. If this procedure is not needed, one can omit line 8 and 10 in this script. Line 11 sets up the Hamiltonian matrix for levels in $n = 2$ and $n = 3$ complexes, diagonalize it, and saves to the energy level information in the binary file `ne.lev.b`. Line 12 builds an in-memory table of energy levels, which is used to convert the binary files to their ASCII counterparts in verbose mode, such as done in Line 13, which converts `ne.lev.b` to `ne.lev` (the last argument to `PrintTable` indicates it be done in verbose mode). For the conversion in simple mode (the last argument is 0), the in-memory table is not needed, and Line 12 may be omitted. For the difference between the verbose and simple ASCII files, see Chapter 2. Line 14 calculates the E1 oscillator strength and transition rates between cofiguration groups `n2` and `n3`, and saves the results in the binary file `ne.tr.b`. The function `TransitionTable` accepts an optional 4th integer argument specifying the transition type. A negative integer means electric multipol and a positive integer for magnetic multipole. The absolute value of the integer indicates the rank of the multipole. Therefore, -1 would be E1, $+1$ would be M1, etc. Without this argument, the default is E1, as is done here. Line 15 converts the binary output to an ASCII file in verbose mode. The exact formats of binary and ASCII files are explained in Chapter 2. Here we list the two ASCII files `ne.lev` and `ne.tr` resulted from this calculation.

```
File ne.lev:

FAC 1.0.7
Endian = 1
TSess = 1103048155
Type = 1
Verbose = 1
Fe Z =  26.0
NBlocks = 1
E0 = 0, -3.12289011E+04

NELE = 10
```

```
NLEV = 37
  ILEV  IBASE    ENERGY       P   VNL 2J
     0     -1 0.00000000E+00 0   201  0 1*2 2*8       2p6          2p+4(0)0
     1     -1 7.23817529E+02 1   300  4 1*2 2*7 3*1   2p5 3s1      2p+3(3)3 3s+1(1)4
     2     -1 7.25866864E+02 1   300  2 1*2 2*7 3*1   2p5 3s1      2p+3(3)3 3s+1(1)2
     3     -1 7.36421878E+02 1   300  0 1*2 2*7 3*1   2p5 3s1      2p-1(1)1 3s+1(1)0
     4     -1 7.37744083E+02 1   300  2 1*2 2*7 3*1   2p5 3s1      2p-1(1)1 3s+1(1)2
     5     -1 7.54155726E+02 0   301  2 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p-1(1)2
     6     -1 7.57795103E+02 0   301  4 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p-1(1)4
     7     -1 7.59348028E+02 0   301  6 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p+1(3)6
     8     -1 7.60559034E+02 0   301  2 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p+1(3)2
     9     -1 7.62368042E+02 0   301  4 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p+1(3)4
    10     -1 7.68005929E+02 0   301  0 1*2 2*7 3*1   2p5 3p1      2p+3(3)3 3p+1(3)0
    11     -1 7.69846810E+02 0   301  2 1*2 2*7 3*1   2p5 3p1      2p-1(1)1 3p-1(1)2
    12     -1 7.73062840E+02 0   301  2 1*2 2*7 3*1   2p5 3p1      2p-1(1)1 3p+1(3)2
    13     -1 7.73470206E+02 0   301  4 1*2 2*7 3*1   2p5 3p1      2p-1(1)1 3p+1(3)4
    14     -1 7.90365155E+02 0   301  0 1*2 2*7 3*1   2p5 3p1      2p-1(1)1 3p-1(1)0
    15     -1 8.00169329E+02 1   302  0 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d-1(3)0
    16     -1 8.01137358E+02 1   302  2 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d-1(3)2
    17     -1 8.02966076E+02 1   302  4 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d+1(5)4
    18     -1 8.03096178E+02 1   302  8 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d+1(5)8
    19     -1 8.03812355E+02 1   302  6 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d-1(3)6
    20     -1 8.05502105E+02 1   302  4 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d-1(3)4
    21     -1 8.06580377E+02 1   302  6 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d+1(5)6
    22     -1 8.11326891E+02 1   302  2 1*2 2*7 3*1   2p5 3d1      2p+3(3)3 3d+1(5)2
    23     -1 8.16455702E+02 1   302  4 1*2 2*7 3*1   2p5 3d1      2p-1(1)1 3d-1(3)4
    24     -1 8.17103423E+02 1   302  4 1*2 2*7 3*1   2p5 3d1      2p-1(1)1 3d+1(5)4
    25     -1 8.17679088E+02 1   302  6 1*2 2*7 3*1   2p5 3d1      2p-1(1)1 3d+1(5)6
    26     -1 8.25272086E+02 1   302  2 1*2 2*7 3*1   2p5 3d1      2p-1(1)1 3d-1(3)2
    27     -1 8.60711742E+02 0   300  2 1*2 2*7 3*1   2s1 3s1      2s+1(1)1 3s+1(1)2
    28     -1 8.67823477E+02 0   300  0 1*2 2*7 3*1   2s1 3s1      2s+1(1)1 3s+1(1)0
    29     -1 8.93685214E+02 1   301  0 1*2 2*7 3*1   2s1 3p1      2s+1(1)1 3p-1(1)0
    30     -1 8.94146680E+02 1   301  2 1*2 2*7 3*1   2s1 3p1      2s+1(1)1 3p-1(1)2
    31     -1 8.96450212E+02 1   301  4 1*2 2*7 3*1   2s1 3p1      2s+1(1)1 3p+1(3)4
    32     -1 8.98437005E+02 1   301  2 1*2 2*7 3*1   2s1 3p1      2s+1(1)1 3p+1(3)2
    33     -1 9.38592464E+02 0   302  2 1*2 2*7 3*1   2s1 3d1      2s+1(1)1 3d-1(3)2
    34     -1 9.38724254E+02 0   302  4 1*2 2*7 3*1   2s1 3d1      2s+1(1)1 3d-1(3)4
    35     -1 9.38975219E+02 0   302  6 1*2 2*7 3*1   2s1 3d1      2s+1(1)1 3d+1(5)6
    36     -1 9.43734651E+02 0   302  4 1*2 2*7 3*1   2s1 3d1      2s+1(1)1 3d+1(5)4
```

File ne.tr:

```
FAC 1.0.7
Endian  = 1
TSess   = 1103048155
Type    = 2
Verbose = 1
Fe Z    =  26.0
NBlocks = 1

NELE    = 10
```

```
NTRANS  = 7
MULTIP  = -1
GAUGE   = 2
MODE    = 1
     2  2     0  0  7.2587E+02  1.130597E-01  8.616084E+11  1.127617E-01
     4  2     0  0  7.3774E+02  9.944485E-02  7.828559E+11  1.048997E-01
    16  2     0  0  8.0114E+02  9.438239E-03  8.761793E+10 -3.101188E-02
    22  2     0  0  8.1133E+02  6.221187E-01  5.923155E+12 -2.501928E-01
    26  2     0  0  8.2527E+02  2.493449E+00  2.456309E+13  4.966355E-01
    30  2     0  0  8.9415E+02  3.203146E-02  3.704097E+11  5.407792E-02
    32  2     0  0  8.9844E+02  2.652003E-01  3.096259E+12 -1.552313E-01
```

In file `ne.lev`, the energy, parity, $2J$ (J is the total angular momentum of the level), and configuration coupling informations are listed. In file `ne.tr`, the upper and lower level indexes, the $2J$ values of these levels, the transition energy, gf-values, radiative decay rates, and the reduced multipole matrix elements are given.

Acknowledgments

Throughout the development of this work, the discussion with Ehud Behar, Masao Sako, Peter Beiersdorfer, Ali Kinkhabwala and Steven Kahn has been very useful. Many Fortran 77 subroutines were retrieved from Netlib repository (**www.netlib.org**) and used in this package, as well as several programs from Computer Physics Communications Program Library at **www.cpc.cs.qub.ac.uk**.

The original development of this code (during Dec 2000 – Aug 2003, or prior to version 1.0.2) was supported by NASA through Chandra Postdoctoral Fellowship Award Number PF01-10014 issued by the Chandra X-ray Observatory Center, which is operated by Smithsonian Astrophysical Observatory for and on behalf of NASA under contract NAS8-39073.

Any opinions, findings and conclusions or recommendations expressed in this manual are those of the author and do not necessrarily reflect the views of the National Aeronautics Space Administration and/or the Smithonian Astrophysical Observatory.

2 Description of Output Files

The primary output files of FAC are in binary format. The I/O functionality and the conversion from binary to ASCII format are implemented in the source files `faclib/dbase.h` and `faclib/dbase.c`. In this chapter, we describe the structure of these files in detail.

2.1 Binary Format

Presently, FAC produces different types of files. Each type is asigned a unique integer, which corresponds to a macro define in the file `faclib/dbase.h`. These types are

`DB_EN = 1` :	Energy levels produced by the function `fac.Structure`.
`DB_TR = 2` :	Radiative transition rates produced by `fac.TransitionTable`.
`DB_CE = 3` :	Collisional excitation cross sections produced by `fac.CETable`.
`DB_RR = 4` :	Radiative recombination and photoionization cross sections produced by `fac.RRTable`.
`DB_AI = 5` :	Autoionization rates produced by `fac.AITable`.
`DB_CI = 6` :	Collisional ionization cross sections produced by `fac.CITable`.
`DB_SP = 7` :	Spectral line strengths produced by `crm.SpecTable`.
`DB_RT = 8` :	Various population rates produced by `crm.RateTable`.
`DB_DR = 9` :	Various population rates produced by `crm.DRStrength`.
`DB_AIM = 10` :	Magnetic sublevel autoionization rates and DR capture strengths produced by `fac.AITableMSub`.
`DB_CIM = 11` :	Magnetic sublevel collisional ionization cross sections produced by `fac.CITableMSub`.

All files have a common structure. It consists of a file header and one or more data blocks. Each data block is comprised of a data header and one or more data records. In the following, we show the C definition of all structs and describe each field in detail. When one field is a pointer, it means that an array is saved in the database. The pointer points to the memory location where the data is stored. In versions 1.0.8 or earlier, the value of the pointer itself is also saved in the file followed by the data stored in the array. Obviously, the saved pointer itself has no meaning once the program exits (since it is a memory location). When reading out the data from the database file, these pointer values should be ignored. In version 1.0.9 or later, the pointers are no longer saved, and the file IO are rewritten in a platform independent way, i.e., the structure fields are written and read one by one, instead of dealing with the structure as an integrated object. This avoids the different memory padding added by the compilers which may be different on different machines.

2.1.1 `F_HEADER`

`F_HEADER` is the file header common to all data files.

```
typedef struct _F_HEADER_ {
  long  tsession;
  int   version;
  int   sversion;
  int   ssversion;
  int   type;
  float atom;
  char  symbol[4];
  int   nblocks;
} F_HEADER;
```

Field Description:

`long tsession`:	Time stamp when the file is created. This is the value returned by the C lib function time(0). It is platform dependent.
`int version`:	Major version number of FAC.
`int sversion`:	Minor version number of FAC.
`int ssversion`:	Release number of FAC.
`int type`:	Type of the data file.
`float atom`:	Atomic number.
`char symbol[4]`:	The first 3 bytes contains a NULL terminated C string representing the 2-charactor abbreviation of the atomic symbol. The 4th byte is either 0 or 1, indicating whether the platform stores data in litle or big endian.
`int nblocks`:	Number of data blocks in this file.

2.1.2 `EN_HEADER`

`EN_HEADER` is the data header for energy level data blocks.

```
typedef struct _EN_HEADER_ {
  long position;
  long length;
  int nele;
  int nlevels;
} EN_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int nlevels`:	Number of levels in this block.

2.1.3 `EN_RECORD`

`EN_RECORD` represents an energy level.

```
#define LNCOMPLEX   32
#define LSNAME      24
#define LNAME       56

typedef struct _EN_RECORD_ {
  short p;
  short j;
  int ilev;
  int ibase;
  double energy;
  char ncomplex[LNCOMPLEX];
  char sname[LSNAME];
  char name[LNAME];
} EN_RECORD;
```

Field Description:

`LNCOMPLEX`:	The length of array holding the complex name.
`LSNAME`:	The length of array holding the non-relativistic configuration name.
`LNAME`:	The length of array holding the relativistic configuration array.
`short p`:	The parity of the level. This parameter was changed in version 0.7.6, and it becomes $\pm(100 \times n + l)$, where n and l are the principle quantum number and orbital angular number of the valence electron, and the $\pm$ sign indicates an even (+) or odd (−) parity state.
`short j`:	2 × the total angular momentum of the level. In UTA mode, `j` is supposed to be the statistical weight minus 1 of the UTA level. However, because a `short` variable is sometimes insufficient to store that value, the code stores it in `ibase` instead. In this case, `j` is always -1.
`int ilev`:	The index of the level.
`int ibase`:	The index of the base level. The base level obtained by peeling off the valence electron, and the resulting levels must be present in the same level file. Other wise, its value is -1. It is not always possible to determine the base level, e.g., when the valence orbital is occupied by more than one electrons. In such cases, `ibase` is also -1. The value of `ibase` is primarily used in dealing with DR and RE rates, where it may help the distinguish different resonance channels and facilitate easy extrapolation. This variable is only added in version FAC 1.0.4. So these later versions are not compartible with the binary output of the ealier versions.
`energy`:	The energy of the level in Hartree.
`char ncomplex[LNCOMPLEX]`:	The complex name. It is in the format of `n1*nq1 n2*nq2`$\cdots$, where `n1` and `n2` are the principle quantum numbers of the shell, `nq1` and `nq2` are the occupation number of these shells.
`char sname[LSNAME]`:	The non-relativstic configuration name of the level. Each non-relativistic shell is denoted by the standard spectroscopic notation, e.g., `2p2` for 2 electrons in $2p$ shell. Only open and non-empty shells are given. No coupling information is available in this name.
`char name[LNAME]`:	The relativstic configuration name of the level. Each shell is denoted such that `2p+2(2)` represents 2 electrons in $2p_{3/2}(J=1)$ and `2p-2(2)` represents 2 electrons in $2p_{1/2}(J=1)$. The number in the parenthesis is 2 times the total angular momentum of the coupled shell. Immediately after the parenthesis, there is a number indicate the $2J$ value when all preceding shells are coupled. Therefore, `2p+2(2)2 2p-2(2)0` represents a state $[2p^2_{3/2}(J=1)2p^2_{1/2}(J=1)]J=0$.

2.1.4 `TR_HEADER`

`TR_HEADER` is the data header for the radiative transition data blocks.

```
typedef struct _TR_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int gauge;
  int mode;
  int multipole;
} TR_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	Number of transitions in this block.
`int gauge`:	Gauge used in the calculation. 1 is Coulomb gauge, or the velocity form in non-relativistic limit. 2 is Babushkin gauge or the length form.
`int mode`:	Mode used in the calculation. 0 is fully relativistic. 1 is non-relativistic approximation for multipole operators.
`int multipole`:	Multipole type of the transition. Its absolute value is the rank of the multipole, 1 for dipole, 2 for quadrupole, etc. The positive sign represents magnetic type and negative sign represents electric type.

2.1.5 `TR_RECORD`

`TR_RECORD` is the for radiative transition data.

```
typedef struct _TR_RECORD_ {
  int lower;
  int upper;
  float strength;
} TR_RECORD;
```

Field Description:

`int lower`: The lower level index of the transition.

`int upper`: The upper level index of the transition.

`float strength`: In version 1.0.6 or older, This is the weighted oscillator strength gf of the transition. The weighted radiative transition rate is related to gf as (in atomic units):

$$gA = 2\alpha^3\omega^2 gf, \tag{2.1}$$

where α is the fine structure constant, and ω is transition energy in Hartree atomic units.

In version 1.0.7 or newer, this stores the multipole matrix elements M instead. It is related to the gf value as

$$gf = (2L+1)^{-1}\,\omega\,(\alpha\omega)^{2L-2}\,|M|^2\,, \tag{2.2}$$

where L is the multipole rank.

2.1.6 `TR_EXTRA`

`TR_EXTRA` contains the UTA related transition data, namely, the transition energy including the UTA shift, the UTA Gaussian width, and the configuration interaction multipler. This structure is written to the `DB_TR` file only in UTA mode, which is set by `SetUTA()` function.

```
typedef struct _TR_EXTRA_ {
  float energy;
  float sdev;
  float sci;
} TR_EXTRA;
```

Field Description:

`float energy`:	The transition energy including UTA shift.
`float sdev`:	The Gaussian standard deviation of the UTA.
`float sci`:	The configuration interaction multipler, which accounts for the CI within the same non-relativistic configurations.

2.1.7 `CE_HEADER`

`CE_HEADER` is the data header for collisional excitation data blocks.

```
typedef struct _CE_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int qk_mode;
  int n_tegrid;
  int n_egrid;
  int egrid_type;
  int n_usr;
  int usr_egrid_type;
  int nparams;
  int pw_type;
  int msub;
  float te0;
  double *tegrid;
  double *egrid;
  double *usr_egrid;
} CE_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.

`long length`: Number of bytes in this data block, excluding the length of the header.

`int nele`: Number of electrons in the ion for this block.

`int ntransitions`: Number of transitions in this block.

`int qk_mode`: The mode for the calculation of radial integrals. There are 3 choices for collisional excitation. 0 for EXACT, 1 for INTERPOLATE, and 2 for FIT. In the EXACT mode, the collison strengths are calculated at the energy grid specified as is, so the `egrid` and `usr_egrid` must be the same. In the INTERPOLATE mode, the collision strengths are calculated at `egrid`, and interpolated to `usr_egrid`. In the FIT mode, the collision strengths are fitted to an analytic formula and the parameters are output as well. For collision strengths of magnetic sublevels, the FIT mode is not implemented.

`int n_tegrid`: Number of points for the transition energy grid.

`int n_egrid`: Number of points for the collision energy grid.

`int egrid_type`: Type of the energy grid. 0 for the incident electron energy, 1 for scattered electron energy. In the present implementation, only scattered electron energy grid is supported.

`int n_usr`: Number of points for the user collision energy grid.

`int usr_egrid_type`: Type of the user energy grid. 0 for the incident electron energy, 1 for scattered electron energy. In the present implementation, only scattered electron energy grid is supported.

`int nparams`: Number of parameters in the fitting formula if the collision strengths are calculated in the FIT mode. At present, `nparams` is 4.

`int pw_type`: Partial wave type for the last summation. 0 for the incident electron, 1 for the scattered electron.

`int msub`: 0 for total collision strength, 1 for magnetic sublevel specific collision strength.

`float te0`: The characteristic transition energy of the transition array. This is used for the automatic construction of the collision energy grid. The grid has equal space in ln(`egrid+te0`) if `egrid_type = 1`, otherwise, this variable is not used.

`double *tegrid`: The transition energy grid, the number of elements is given by `n_tegrid`.

`double *egrid`: The energy grid, the number of elements is given by `n_egrid`.

`double *usr_egrid`: The user energy grid, the number of elements is given by `n_usr`.

2.1.8 CE_RECORD

CE_RECORD is for collisional excitation data.

```
typedef struct _CE_RECORD_ {
  int lower;
  int upper;
  int nsub;
  float bethe;
  float born[2];
  float *params;
  float *strength;
} CE_RECORD;
```

Field Description:

`int lower`: The lower level index.

`int upper`: The upper level index.

`int nsub`: Number of magnetic sublevel transitions. Because of time reversal symmetry, $\sigma_{m_1 \to m_2} = \sigma_{-m_1 \to -m_2}$, only cross sections with $m_1 <= 0$ are tabulated.

`float bethe`: The Bethe coefficients in the first-Born approximation. It is the logarithmic coefficients at high energies. If `bethe[0]`< 0, it is a spin forbidden transition. Otherwise, it is either a optical-allowed transition or other multipole-allowed transitions.

`float born[2]`: The Born limit of the collision strengths at high energies, which is

$$\begin{aligned} x &= \frac{E_0}{E_{th}} \\ \Omega &= b_0 \ln(x) + b_1, \end{aligned} \tag{2.3}$$

where b_0 is given by `bethe`, if it is an allowed transition. The parameter b_1 is calculated at an energy given by b_2, which is chosen to be very high, about $10^2 E_{th}$ or higher. For spin forbidden transitions, $b_0 = 0$. b_1, b_2 are stored in the array `born[2]`. These numbers are useful to extrapolate the collision strengths to high energies with correct aysmptotic behaviour.

`float *params`: Parameters for the fitting formula, if the fitting mode is used. The number of elements is given by `nparams` in `CE_HEADER`. In the present implementation, different fitting formulae are used for allowed and forbidden transitions. The number of parameters is 4 in all cases. The FIT mode is not robust, avoid using it.

For dipole and higher multipole allowed transitions, the collision strength Ω is given by

$$\begin{aligned} x &= \frac{E_0}{E_{th}} \\ \Omega &= p_0 \left(\frac{1}{x}\right)^{p_1} + p_2 \left(1 - \frac{1}{x}\right)^{p_3} + b \ln x, \end{aligned} \tag{2.4}$$

where E_0 is the energy of the incident electron, E_{th} is the transition threshold, p_0, p_1, p_2 and p_4 are four parameters, and b is the Bethe coefficient, which is 0 for non-dipole transitions.

For forbidden transitions, the collision strength is given by

$$\begin{aligned} \gamma &= -2.0 + p_1 \frac{1}{p_3 + x} + p_2 \left(\frac{1}{p_3 + x}\right)^2 \\ \Omega &= p_0 x^{\gamma}. \end{aligned} \tag{2.5}$$

The FIT mode only applies to the calculation of total cross sections. For magnetic sublevel cross sections, `params` has `nsub` elements, which are the ratios of magnetic sublevel collision strengths to the total collision strength at high energy limit for allowed transitions. For forbidden transitions, these numbers are all 0.

`float *stregnth`:	Collision stregnth on the user energy grid. The number of elements is given by `n_usr` in `CE_HEADER`. It is related to the excitation cross section as (in atomic units): $$\sigma = \frac{\pi}{k_0^2 g_0}\Omega, \qquad (2.6)$$ where g_0 is the statistical weight of the initial state, and k_0 is the kinetic momentum of the incident electron. The number of elements in this array is `nsub`×`n_usr`.

2.1.9 RR_HEADER

`RR_HEADER` is the data header for radiative recombination and photoionization data blocks.

```
typedef struct _RR_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int qk_mode;
  int multipole;
  int n_tegrid;
  int n_egrid;
  int egrid_type;
  int n_usr;
  int usr_egrid_type;
  int nparams;
  double *tegrid;
  double *egrid;
  double *usr_egrid;
} RR_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	Number of transitions in this block.
`int qk_mode`:	The mode for the calculation of radial integrals. There are 3 choices at present. 0 for EXACT, 1 for INTERPOLATE, and 2 for FIT, similar to collsional excitation. However, even if the FIT mode is used, the fitting formula is only valid in the high energy asymptotic regions. The low energy results should be obtained by interpolation.
`int multipole`:	Multipole type of the transition. Its absolute value is the rank of the multipole, 1 for dipole, 2 for quadrupole, etc. The positive sign for magnetic type and negative sign for electric type. Usually, only E1 type is relavent for radiative recombination and photoionization.

`int n_tegrid`: Number of points for the transition energy grid.

`int n_egrid`: Number of points for the collision energy grid.

`int egrid_type`: Type of the energy grid. 0 for the incident photon energy, 1 for photo-electron energy.

`int n_usr`: Number of points for the user collision energy grid.

`int usr_egrid_type`: Type of the user energy grid. 0 for the incident photon energy, 1 for photo-electron energy.

`int nparams`: Number of parameters in the fitting formula if the bound-free oscillator strengths are calculated in the FIT mode. In the present imprementation, `nparams` is 4.

`double *tegrid`: The transition energy grid, the number of elements is given by `n_tegrid`.

`double *egrid`: The energy grid, the number of elements is given by `n_egrid`.

`double *usr_egrid`: The user energy grid, the number of elements is given by `n_usr`.

2.1.10 `RR_RECORD`

`RR_RECORD` is for radiative recombination and photoionization data.

```
typedef struct _RR_RECORD_ {
  int b;
  int f;
  int kl;
  float *params;
  float *strength;
} RR_RECORD;
```

Field Description:

`int b`: The bound state index.

`int f`: The free state index.

`int kl`: The orbital angular momentum of the ionized shell for the dominant wavefunction component.

`float *params`: The parameters in the fitting formula for the bound-free oscillator strength, if the FIT mode is used. The fitting formula only provides a high energy asymptotic behavior. Low energy values should be interpolated from the tabulated strengths. The fitting formula is

$$\begin{aligned} x &= \frac{E_e + p_3}{p_3} \\ y &= \frac{1 + p_2}{\sqrt{x} + p_2} \\ \frac{d(gf)}{dE} &= \frac{E_\gamma}{E_e + p_3} p_0 x^{-3.5-l+\frac{1}{2}p_1} y^{p_1}, \end{aligned} \tag{2.7}$$

where E_e is the photo-electron energy, E_γ is the photon energy, E_{th} is the ionization threshold, p_0, p_1, p_2, and p_3 are the parameters, and l is the orbital angular momentum of the ionized shell. The asymptotic behavior represented by the power law only takes into account the ionization of the dominant basis in the wavefunction expansion. The result is in atomic unit Hartree^{-1}.

`float *strength`: The weighted bound-free oscillator strength in atomic units. It is related to photoionization and radiative recombination as (in atomic units):

$$\begin{aligned}
\sigma_{PI} &= 2\pi\alpha\frac{df}{dE} \\
&= \frac{2\pi\alpha}{g_i}\frac{1+\alpha^2\varepsilon}{1+\frac{1}{2}\alpha^2\varepsilon}\frac{d(gf)}{dE} \\
\sigma_{RR} &= \frac{\alpha^2}{2}\frac{g_i}{g_f}\frac{\omega^2}{\varepsilon\left(1+\frac{1}{2}\alpha^2\varepsilon\right)}\sigma_{PI}, \qquad (2.8)
\end{aligned}$$

where α is the fine structure constant, g_i and g_f are the statistical weight of the bound states before and after the photoionzation takes place respectively, ω is the photon energy, and ε is the energy of the ejected photo-electron. The tabulated values are $d(gf)/dE$.

2.1.11 `AI_HEADER`

`AI_HEADER` is the data header for autoionization data blocks.

```
typedef struct _AI_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int channel;
  int n_egrid;
  double *egrid;
} AI_HEADER;
```

Field Description:

`long position`: The number of bytes from the beginning of the file to the place where this data block starts.

`long length`: Number of bytes in this data block, excluding the length of the header.

`int nele`: Number of electrons in the ion for this block.

`int ntransitions`: Number of transitions in this block.

`int channel`: This an identifier to label the autoionization channel, which does not have specific physical meaning.

`int n_egrid`: The number of points for the Auger electron energy grid. The autoionzation radial integrals are calculated on this grid and interpolated to the actual discrete energies.

`double *egrid`: The energy grid. The number of elements is given by `n_egrid`.

2.1.12 `AI_RECORD`

`AI_RECORD` is for autoionization data.

```
typedef struct _AI_RECORD_ {
  int b;
  int f;
  float rate;
} AI_RECORD;
```

Field Description:

`int b`:	The bound state index.
`int f`:	The free state index.
`float rate`:	The autoionization rate.

2.1.13 `CI_HEADER`

`CI_HEADER` is the data header for collisional ionization data blocks.

```
typedef struct _CI_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int qk_mode;
  int n_tegrid;
  int n_egrid;
  int egrid_type;
  int n_usr;
  int usr_egrid_type;
  int nparams;
  int pw_type;
  double *tegrid;
  double *egrid;
  double *usr_egrid;
} CI_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	Number of transitions in this block.
`int qk_mode`:	The mode for the calculation of radial integrals. At present, there are 3 choices. 3 for CB mode (Coulomb-Born), 4 for DW mode (distorted-wave), and 5 for BED mode (binary-encounter-dipole). In CB mode, the radial integrals are obtained by looking up a table of Coulomb-Born-Exchange results from **??**, which is very fast. In DW mode, the integrals are calculated using the distorted-wave approximation, which is very slow. In BED mode, the binary-encounter-dipole theory of **?** is used which makes use of bound-free oscillator strength of the same transition. This method is also very fast.

`int n_tegrid`: Number of points for the transition energy grid.

`int n_egrid`: Number of points for the collision energy grid.

`int egrid_type`: Type of the energy grid. 0 for the incident electron energy, 1 for the total energy of scattered and ejected electron.

`int n_usr`: N umber of points for the user collision energy grid.

`int usr_egrid_type`: Type of the user energy grid. 0 for the incident electron energy, 1 for the total energy of scattered and ejected electrons .

`int nparams`: Number of parameters in the fitting formula. The final collision strength for total ionization cross sections are fitted with a 4 parameter formula.

`int pw_type`: Partial wave type for the last summation. 0 for the incident electron, 1 for the scattered electron. It is always 0 for distorted-wave calculation of ionization.

`double *tegrid`: The transition energy grid, the number of elements is given by `n_tegrid`.

`double *egrid`: The energy grid, the number of elements is given by `n_egrid`.

`double *usr_egrid`: The user energy grid, the number of elements is given by `n_usr`.

2.1.14 `CI_RECORD`

`CI_RECORD` is for collisional ionization data.

```
typedef struct _CI_RECORD_ {
  int b;
  int f;
  int kl;
  float *params;
  float *strength;
} CI_RECORD;
```

Field Description:

`int b`: The bound state index.

`int f`: The free state index.

`int kl`: The orbital angular momentum of the ionized shell for the dominant wavefunction component.

`float *params`: The parameters in the fitting formula for the collision strength. The number of elements is given by `nparams` in `CI_HEADER`, which is 4. The formula used is

$$\begin{aligned} x &= \frac{E_0}{E_{th}} \\ y &= 1 - \frac{1}{x} \\ \Omega &= p_0 \ln x + p_1 y^2 + p_2 \frac{1}{x} y + p_3 \frac{1}{x^2} y, \end{aligned} \tag{2.9}$$

where E_0 is the energy of the incident electron, E_{th} is the ionization threshold, p_0, p_1, p_2, and p_3 are the four parameters. The parameter p_0 is actually obtained from the bound-free oscillator strength, which is more reliable than one would get by fitting the calculated collision stregnths.

`float *strength`:	The collision strength for ionization. It is related to the ionization cross section as (in atomic units):

$$\sigma = \frac{1}{k_0^2 g_0}\Omega, \tag{2.10}$$

where k_0 is the kinetic momentum of the incident electron, and g_0 is the statistical weight of the initial state. The missing of the factor π as compared to the formula for collisional excitation is due to the different normalization for bound and free states.

2.1.15 `SP_HEADER`

`SP_HEADER` is the data header for spectral line data blocks. The spectral data are generated by CRM model. A `DB_SP` data file contains two parts. The first part is the detailed level population table. The second part is the spectral line emissivity table. Energy levels in a CRM model are divided into superlevel blocks. One superlevel block occupies one data block in the first part, and all transitions between two superlevel blocks make up one data block in the second part.

```
typedef struct _SP_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int iblock;
  int fblock;
  char icomplex[LNCOMPLEX];
  char fcomplex[LNCOMPLEX];
  int type;
} SP_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	For the first part, this the number of levels in the block. For the second part, this is the number of spectral lines.
`int iblock`:	For the first part, this the superlevel block index. For the second part, this is the superlevel block index for the initial states of transitions.
`int fblock`:	For the first part, this is always 0. For the second part, this is the superlevel block index for the final states of the transitions.
`char icomplex[LNCOMPLEX]`:	The configuration complex name for the initial states.
`char fcomplex[LNCOMPLEX]`:	The configuration complex name for the final states. For the first part, this is always an empty string.

`int type`: Type of the block. For the first part, the type is always 0. For the second part, the type is encoded as $10000 \times n_0 + 100 \times n_1 + n_2$, where n_1 is the initial principle quantum number of the electron making the transition, and n_2 is the final principle quantum number. If $n_0 \neq 0$, this transition is the so called dielectronic recombination satellite line, which have a spectator electron at the orbital with principle quantum number n_0. If $n_0 = n_1 = 0$, then the line is radiative recombination continuum onto the orbital with principal quantum number n_2.

2.1.16 SP_RECORD

```
typedef struct _SP_RECORD_ {
  int lower;
  int upper;
  float energy;
  float strength;
  float rrate;
  float trate;
} SP_RECORD;
```

Field Description:

`int lower`: For the level population table, this is level index within the same ion. For the line emissivity table, this is the level index for the lower state of the transition.

`int upper`: For the level population table, this is the level index within the same superlevel block. For the line emissivity, this is the index for the upper state of the transition.

`float energy`: For the level population table, this is the energy of the level in atomic unit. For the line emissivity, this is the transition energy in eV.

`float strength`: For the level population table, this is the concentration of the level. For the line emissivity, this is the line luminosity in photons/s.

`float rrate`: The radiative transition rate from `upper` to `lower`.

`float trate`: The total decay rate of `upper`.

2.1.17 SP_EXTRA

`SP_EXTRA` contains the UTA width of the transition. It is written to the `DB_SP` file only in the UTA mode.

```
typedef struct _SP_EXTRA_ {
  float sdev;
} SP_EXTRA;
```

Field Description:

`float sdev`: The Gaussian UTA standard deviation of the transition.

2.1.18 RT_HEADER

`RT_HEADER` is the header for the rate data file. The rates are generated by the CRM model. It tabulates the rates for individual processes from and to specific levels or superlevel blocks. In the `DB_RT` files, a data block consists the rates from all other states to one level or one superlevel depending on how the file is created.

```
typedef struct _RT_HEADER_ {
  long position;
  long length;
  int ntransitions;
  int iedist;
  int np_edist;
  double *p_edist;
  float eden;
  int ipdist;
  int np_pdist;
  double *p_pdist;
  float pden;
} RT_HEADER;
```

Field Description:

`long position`: The number of bytes from the beginning of the file to the place where this data block starts.

`long length`: Number of bytes in this data block, excluding the length of the header.

`int ntransitions`: The number of superlevel blocks from which the rates are tabulated in this block.

`int iedist`: The type of electron energy distribution for the CRM model. At present, two types are supported. 0 for Maxwellian, and 1 for an Gaussian monoenergetic beam.

`int np_edist`: Number of parameters for the electron energy distribution.

`double *p_edist`: Parameters for the electron energy distribution. The following table describes the parameters for exsisting distributions:

ID	Dist	Params	Unit
0	Maxwellian	0: T_e	eV
		1: E_{min}	eV
		2: E_{max}	eV
1	Gaussian	0: E_0	eV
		1: σ_E	eV
		2: E_{min}	eV
		3: E_{max}	eV
2	MaxPower	0: T_e	eV
		1: α	
		2: $\frac{E_p}{E_m}$	
		3: E^p_{min}	eV
		4: E^p_{max}	eV
		5: E^m_{min}	eV
		6: E^m_{max}	eV
3	PowerLaw	0: α	
		1: E_{min}	eV
		2: E_{max}	eV

`float eden`: The electron density in unit of 10^{10} cm^{-3}.

`int ipdist`: The type of photon energy distribution for the CRM model. At present, only one type is supported. 0 for power law.

`int np_pdist`:	Number of parameters for the photon energy distribution.
`double *p_pdist`:	Parameters for the photon energy distribution. The following table describes the parameters for exsisting distributions:

ID	Dist	Params	Unit
0	Black Body	0: T	eV
		1: E_{min}	eV
		2: E_{max}	eV
1	PowerLaw	0: α	
		1: E_{min}	eV
		2: E_{max}	eV

`float pden`:	The photon energy density of the radiation field in unit of erg cm^{-3}. If the distribution is Black Body, this is the dilution factor.

2.1.19 RT_RECORD

```
typedef struct _RT_RECORD_ {
  int dir;
  int iblock;
  float nb;
  float tr;
  float ce;
  float rr;
  float ai;
  float ci;
  char icomplex[LNCOMPLEX];
} RT_RECORD;
```

Field Description:

`int dir`:	The direction of the transition. If `dir` is 0, the rate is into this superlevel, if `dir` is 1, the rate is out of this superlevel.
`int iblock`:	The index of the superlevel block from which the rates originate.
`float nb`:	The concentration of the superlevel block from which the rates originate.
`float tr`:	The radiative transition rates.
`float ce`:	The collisional excitation rates.
`float rr`:	The radiative recombination and possible photoionization rates (if there is radiation field included in the model).
`float ai`:	The autoionization rates and dielectronic capture rates.
`float ci`:	The collisional ionization rates.
`char icomplex[LNCOMPLEX]`:	The configuration complex name of the superlevel block from which the rates originate.

2.1.20 DR_HEADER

```
typedef struct _DR_HEADER_ {
  long int position;
  long int length;
  int nele;
  int ilev;
  int ntransitions;
  int vn;
  int j;
  float energy;
} DR_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the recombining ion for this block.
`int ilev`:	The level index of the recombining ion for this block.
`int ntransitions`:	The number of superlevel blocks from which the rates are tabulated in this block.
`int vn`:	The principle quantum number of the captured electron.
`int j`:	The $2j$ value of the recombining state.
`float energy`:	The energy of the recombining state.

2.1.21 `DR_RECORD`

```
typedef struct _DR_RECORD_ {
  int ilev;
  int flev;
  int ibase;
  int fbase;
  int vl;
  int j;
  float energy;
  float etrans;
  float br;
  float ai;
  float total_rate;
} DR_RECORD;
```

Field Description:

`int ilev`:	The level index of the autoionizing state.
`int flev`:	The level index of the final state due to the decay (either autoionization or radiative) of the autoionizing state. For the total DR strength, this variable is always -1, since the result is the sum of all radiative stabilazation routes.
`int ibase`:	The level index of the core for the resonance state.
`int fbase`:	The level index of the core for the final state, if applicable.
`int vl`:	The orbital angular momentum of the captured electron.
`int j`:	The $2j$ value of the autoionizaing state.

`float energy`:	The energy of the autoionizing state relative to the recombining state, whose energy is given in the `DR_HEADER`.
`float etrans`:	The transition energy from `ilev` to `flev`.
`float br`:	The branching ratio of the autoionizing state to various final states. For total DR strength, this is the total radiative branching ratio. For resonance excitation, this is the branching ratio of autoionization to individual states. For satellite calculations, this is the braching ratio of radiative transition to individual states.
`float ai`:	The autoionization rate to the recombining level.
`float total_rate`:	The total decay rate of the autoionization state, i.e., including all autoionization and radiative decay channels.

2.1.22 AIM_HEADER

```
typedef struct _AIM_HEADER_ {
  long int position;
  long int length;
  int nele;
  int ntransitions;
  int channel;
  int n_egrid;
  double *egrid;
} AIM_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	Number of transitions in this block.
`int channel`:	This an identifier to label the autoionization channel, which does not have specific physical meaning.
`int n_egrid`:	The number of points for the Auger electron energy grid. The autoionzation radial integrals are calculated on this grid and interpolated to the actual discrete energies.
`double *egrid`:	The energy grid. The number of elements is given by `n_egrid`.

2.1.23 AIM_RECORD

```
typedef struct _AIM_RECORD_ {
  int b;
  int f;
  int nsub;
  float *rate;
} AIM_RECORD;
```

Field Description:

`int b`:	The bound state index.
`int f`:	The free state index.
`int nsub`:	The number of entries in array `rate`. It is twice the number of transitions tabulated, because both autoionziation and DR capture strength need to be stored.
`float *rate`:	An array containing magnetic sublevel autoionization rates and DR capture strength. Suppose $A(M_1, M_2)$ represents autoionization rate from M_1 sublevel of `b` to M_2 sublevel of `f`, and $R(M_1, M_2)$ is capture strength from M_2 sublevel of `f` to M_1 sublevel of `b`. Then the array `rate` is generated in the order consistent with the following code:

```
t = 0
for (M1 = -J1; M1 <= 0; M1++) {
  for (M2 = -J2; M2 <= J2; M2++) {
     rate[t++] = A(M1,M2);
     rate[t++] = R(M1,M2);
  }
}
```

Due to axial symmetry, values for $M_1 > 0$ can be obtained from those with $M_1 \leq 0$.

2.1.24 CIM_HEADER

`CIM_HEADER` is the data header for magnetic sublevel collisional ionization data blocks.

```
typedef struct _CIM_HEADER_ {
  long position;
  long length;
  int nele;
  int ntransitions;
  int n_egrid;
  int egrid_type;
  int n_usr;
  int usr_egrid_type;
  double *egrid;
  double *usr_egrid;
} CIM_HEADER;
```

Field Description:

`long position`:	The number of bytes from the beginning of the file to the place where this data block starts.
`long length`:	Number of bytes in this data block, excluding the length of the header.
`int nele`:	Number of electrons in the ion for this block.
`int ntransitions`:	Number of transitions in this block.
`int n_egrid`:	Number of points for the collision energy grid.
`int egrid_type`:	Type of the energy grid. 0 for the incident electron energy, 1 for the total energy of scattered and ejected electron.
`int n_usr`:	N umber of points for the user collision energy grid.

`int usr_egrid_type`:	Type of the user energy grid. 0 for the incident electron energy, 1 for the total energy of scattered and ejected electrons .
`double *tegrid`:	The transition energy grid, the number of elements is given by `n_tegrid`.
`double *egrid`:	The energy grid, the number of elements is given by `n_egrid`.
`double *usr_egrid`:	The user energy grid, the number of elements is given by `n_usr`.

2.1.25 `CIM_RECORD`

`CIM_RECORD` is for magnetic sublevel collisional ionization data.

```
typedef struct _CIM_RECORD_ {
  int b;
  int f;
  int nsub;
  float *strength;
} CIM_RECORD;
```

Field Description:

`int b`: The bound state index.

`int f`: The free state index.

`int nusb`: The number of sublevel transitions in the `strength` array.

`float *strength`: The magnetic sublevel collision strength for ionization. It is related to the ionization cross section as (in atomic units):

$$\sigma = \frac{1}{k_0^2}\Omega, \tag{2.11}$$

where k_0 is the kinetic momentum of the incident electron. The different magnetic sublevel transitions are arranged in the order similar to those for excitation, i.e., $-J_i \to -J_f$, $-J_i \to -J_f + 1$, $\cdots$, $-J_i \to J_f$, $-J_i + 1 \to -J_f$, $-J_i + 1 \to -J_f + 1$, $\cdots$, $-J_i + 1 \to J_f$, $\cdots$. Only the cross sections with $M_i \leq 0$ are included, since those with $M_i \geq 0$ can be obtained using time reversal symmetry.

2.2 ASCII Format

FAC provides functions to convert the binary output to ASCII files. There are two types of ASCII formats, a simple translation of binary files and a more verbose version that adds more derived information for the sake of convenience. If the ASCII files are created to be human-readable, the verbose form should be used.

In the simple form, the contents of binary files are converted to ASCII format as is. No additional information is added. All physical values are in atomic units as is in binary files. The different byte-order used by different platforms are taken into account automatically. Therefore, it is possible to create the binary files on a little endian machine (probably faster), then convert them to ASCII format on a slower big endian machine.

In the verbose form, the more common units of physical quanties are used. Specifically, s^{-1} for transition rates, 10^{-20} cm^2 for cross sections, and eV for energies. For data files other than `DB_EN` type, the energies and angular momenta of the levels involved in the processes are not included in the binary version. In the verbose form of corresponding ASCII files, these infomations are added by looking up in the energy level table. Also, for `DB_TR` files, not only matrix elements, but also gf values and radiative transition rates are tabulated. For `DB_CE` and `DB_CI`, cross sections are tabulated along with the collision strengths. For

`DB_RR`, radiative recombination and photoionization cross sections are tabulated along with the bound-free differential gf values. For `DB_AI`, the energy integrated dielectronic capture strengths (in unit of 10^{-20} eV cm^2) are tabulated in addition to the autoionization rates.

In the following sections, a portion of each type of database file in the verbose form is listed and significant fields explained. The lines start with a "`#`" are the added explanation, which are not part of the output file. These files are generated with the scripts in the `demo/` directory come with FAC.

2.2.1 `DB_EN`

```
# version numbers
FAC 1.0.4
# binary order used in the binary file
Endian = 0
# time stamp when the file was created.
TSess = 1020438482
# database type
Type = 1
# this file is in verbose form
Verbose = 1
# atomic symbol and atomic number
Fe Z = 26.0
# number of data blocks in this file
NBlocks = 1
# the index and the absolute energy of the ground state
E0 = 0, -3.12494784E+04

# data block begins
# number of electron for the states in this block
NELE = 10
# number of levels in this block
NLEV = 37
  ILEV  IBASE    ENERGY       P   VNL 2J
     0     -1  0.00000000E+00 0   201  0 1*2 2*8      2p6      2p+4(0)0
     1     -1  7.23810448E+02 1   300  4 1*2 2*7 3*1  2p5 3s1  2p+3(3)3 3s+1(1)4
     2     -1  7.25859655E+02 1   300  2 1*2 2*7 3*1  2p5 3s1  2p+3(3)3 3s+1(1)2
     3     -1  7.36414516E+02 1   300  0 1*2 2*7 3*1  2p5 3s1  2p-1(1)1 3s+1(1)0
     4     -1  7.37736604E+02 1   300  2 1*2 2*7 3*1  2p5 3s1  2p-1(1)1 3s+1(1)2
     5     -1  7.54149163E+02 0   301  2 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p-1(1)2
     6     -1  7.57788593E+02 0   301  4 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p-1(1)4
     7     -1  7.59341459E+02 0   301  6 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p+1(3)6
     8     -1  7.60552577E+02 0   301  2 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p+1(3)2
     9     -1  7.62361512E+02 0   301  4 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p+1(3)4
    10     -1  7.67999430E+02 0   301  0 1*2 2*7 3*1  2p5 3p1  2p+3(3)3 3p+1(3)0
   ......
```

The column labels by `VNL` is $100 \times n + l$, where n and l are the principle and orbital angular quantum numbers of the valence electron.

2.2.2 `DB_TR`

```
FAC 1.0.7
Endian = 0
```

```
TSess = 1021577025
Type = 2
Verbose = 1
Fe Z =  26.0
NBlocks = 1

# the data block begins
NELE = 10
# number of transitions in this block
NTRANS = 7
# multipole type of the transition
Multip = -1
# gauge used in the calculation
Gauge = 2
# mode used in the radial integral
Mode = 1
#upper 2J  lower 2J  Delta E     gf            TR rate(1/s)  multipole
     2  2      0  0  7.2587E+02  1.130597E-01  8.616084E+11  1.127617E-01
     4  2      0  0  7.3774E+02  9.944485E-02  7.828559E+11  1.048997E-01
    16  2      0  0  8.0114E+02  9.438239E-03  8.761793E+10 -3.101188E-02
    22  2      0  0  8.1133E+02  6.221187E-01  5.923155E+12 -2.501928E-01
    26  2      0  0  8.2527E+02  2.493449E+00  2.456309E+13  4.966355E-01
    30  2      0  0  8.9415E+02  3.203146E-02  3.704097E+11  5.407792E-02
    32  2      0  0  8.9844E+02  2.652003E-01  3.096259E+12 -1.552313E-01
```

After version 1.0.8, if the UTA mode is used, the output contains an additional column after the transition energy, which is the Gaussian standard deviation of the UTA transition. The $2J$ values in this case are also redefined to be the statistical weight of the configuraiton minus 1.

2.2.3 DB_CE

```
FAC 0.7.9
Endian = 0
TSess = 1021577097
Type = 3
Verbose = 1
Fe Z =  26.0
NBlocks = 1

# data blocks begin
NELE = 10
NTRANS = 36
# mode used in the radial integral
QKMODE = 0
# number of parameters in the fitting formula (only if QKMODE = 2)
NPARAMS = 0
# 0 for total collision strength. 1 for magnetic sublevel.
MSUB = 0
# partial wave summation mode. always 0.
PWTYPE = 0
# number of points in the transition energy grid, followed by the grid
NTEGRID = 2
```

```
  7.24352072E+02
  9.45773957E+02
# characteristic transition energy used in grid construction.
TE0 =  9.44829120E+02
# energy grid type.
ETYPE = 1
# energy grid
NEGRID = 6
  4.72414560E+01
  5.79761386E+02
  1.39812537E+03
  2.65576771E+03
  4.58848260E+03
  7.55863296E+03
# user energy grid type and the user grid.
UTYPE = 1
NUSR = 6
  4.72414560E+01
  5.79761386E+02
  1.39812537E+03
  2.65576771E+03
  4.58848260E+03
  7.55863296E+03
#lower 2J upper 2J Delta E  nsub
    0  0     1  4  7.2435E+02 1
#The Bethe coefficient and 2 Born coefficients in the Born approximation.
-1.0000E+00  0.0000E+00  0.0000E+00
# if QKMODE = 2, the parameter line is present here.
#user egrid  coll. str.  cross sec.
 4.7241E+01  1.5347E-03  2.3789E-01
 5.7976E+02  9.5137E-04  8.7207E-02
 1.3981E+03  5.1906E-04  2.9211E-02
 2.6558E+03  2.5016E-04  8.8294E-03
 4.5885E+03  1.1322E-04  2.5376E-03
 7.5586E+03  5.1291E-05  7.3523E-04
    0  0     2  2  7.2639E+02 1
 9.1750E-03 -7.0392E-03  7.2655E-03
 4.7241E+01  1.8857E-03  2.9153E-01
 5.7976E+02  3.7280E-03  3.4119E-01
 1.3981E+03  6.3378E-03  3.5633E-01
 2.6558E+03  9.4893E-03  3.3472E-01
 4.5885E+03  1.2996E-02  2.9116E-01
 7.5586E+03  1.6729E-02  2.3974E-01
 ......
```

If `MSUB` $= 1$, then the data for each transition contains `nsub` blocks, representing several $m_i \to m_f$ transitions. Before each block, the ratio of the magnetic sublevel collision strengths to the total collision strength at high energy limit is given. Due to the time reversal symmetry, the cross section for $-m_i \to -m_f$ is the same as that for $m_i \to m_j$, only the cross sections with $m_i \leq 0$ are tabulated in the order $-J_i \to -J_f$, $-J_i \to -J_f+1$, $\cdots$, $-J_i \to J_f$, $-J_i+1 \to -J_f$, $-J_i+1 \to -J_f+1$, $\cdots$, $-J_i+1 \to J_f$, $\cdots$.

2.2.4 DB_RR

```
FAC 0.7.3
Endian = 0
TSess = 1021577047
Type = 4
Verbose = 1
Fe Z =  26.0
NBlocks = 1

# the data blocks begin
NELE = 3
NTRANS = 3
QKMODE = 2
# multipole type
MULTIP = -1
# number of parameters in the fitting formula
NPARAMS = 4
NTEGRID = 1
  2.01377924E+03
ETYPE = 1
NEGRID = 6
  1.00688962E+02
  1.23568529E+03
  2.97992069E+03
  5.66042025E+03
  9.77974833E+03
  1.61102339E+04
UTYPE = 1
NUSR = 6
  1.00688962E+02
  1.23568529E+03
  2.97992069E+03
  5.66042025E+03
  9.77974833E+03
  1.61102339E+04
#bound 2J  free 2J  Delta E    L
    7  1     0  0  2.0465E+03  0
# the parameters in the fitting formula
 3.8124E-02  4.9724E+00  1.2195E+00  2.1768E+03
#user egrid RR cross sec. PI cross sec. gf
 1.0069E+02  1.7567E-01  1.9604E+00  3.0537E-02
 1.2357E+03  1.4375E-02  8.4255E-01  1.3124E-02
 2.9799E+03  5.5123E-03  3.3223E-01  5.1751E-03
 5.6604E+03  2.4847E-03  1.2100E-01  1.8848E-03
 9.7797E+03  1.1476E-03  4.1004E-02  6.3872E-04
 1.6110E+04  5.2175E-04  1.3029E-02  2.0295E-04
    8  1     0  0  1.9975E+03  1
 3.4223E-02  5.3145E+00  1.2206E+00  2.1537E+03
 1.0069E+02  1.5214E-01  1.7781E+00  2.7697E-02
 1.2357E+03  8.4472E-03  5.1024E-01  7.9480E-03
 2.9799E+03  2.1814E-03  1.3407E-01  2.0885E-03
 5.6604E+03  6.5910E-04  3.2509E-02  5.0639E-04
 9.7797E+03  2.0448E-04  7.3672E-03  1.1476E-04
 1.6110E+04  6.2231E-05  1.5624E-03  2.4338E-05
    9  3     0  0  1.9810E+03  1
 6.9754E-02  5.1620E+00  1.2206E+00  2.1350E+03
 1.0069E+02  2.9691E-01  1.7625E+00  5.4910E-02
 1.2357E+03  1.6320E-02  4.9796E-01  1.5513E-02
 2.9799E+03  4.1721E-03  1.2907E-01  4.0209E-03
 5.6604E+03  1.2475E-03  3.0899E-02  9.6262E-04
 9.7797E+03  3.8274E-04  6.9143E-03  2.1541E-04
 1.6110E+04  1.1506E-04  1.4471E-03  4.5081E-05
```

2.2.5 `DB_AI`

```
FAC 0.7.3
Endian = 0
TSess = 1021577153
Type = 5
Verbose = 1
Se Z =  34.0
NBlocks = 1

# data blocks begin
NELE = 10
# number of transitions
NTRANS = 92
# channel number (no physical meaning)
CHANNE = 0
# free electron energy grid
NEGRID = 2
  3.43213508E+02
  5.58605513E+02
#bound 2J free 2J  Delta E     AI rate    DC strength
    2  4    0  3  3.8679E+02  1.3705E+13  1.0962E+01
    2  4    1  1  3.4322E+02  1.6996E+11  3.0642E-01
    3  0    0  3  4.0594E+02  1.2973E+13  1.9775E+00
    3  0    1  1  3.6236E+02  7.9903E+11  2.7289E-01
    4  4    0  3  4.1845E+02  2.3652E+11  1.7487E-01
    4  4    1  1  3.7487E+02  2.2905E+09  3.7807E-03
    ......
```

2.2.6 `DB_CI`

```
FAC 0.7.3
Endian = 0
TSess = 1021577194
Type = 6
Verbose = 1
Fe Z =  26.0
NBlocks = 1

# data blocks begin
```

```
NELE = 10
NTRANS = 3
QKMODE = 5
NPARAMS = 4
PWTYPE = 0
NTEGRID = 2
  1.26072567E+03
  1.39546596E+03
ETYPE = 1
NEGRID = 6
  6.64047908E+01
  8.14939607E+02
  1.96527014E+03
  3.73307079E+03
  6.44978485E+03
  1.06247665E+04
UTYPE = 1
NUSR = 8
  5.00000000E+02
  9.00000000E+02
  1.30000000E+03
  1.70000000E+03
  2.10000000E+03
  4.20000000E+03
  6.00000000E+03
  8.00000000E+03
#bound 2J  free 2J  Delta E    L
    0  0     1  3  1.2607E+03  1
# parameters in the fitting formula
 1.2549E-01  6.7308E-01 -5.4651E-01  7.2856E-01
#user egrid  coll. str.  cross sec.
 5.0000E+02  9.0588E-02  1.9535E+00
 9.0000E+02  1.5721E-01  2.7605E+00
 1.3000E+03  2.1672E-01  3.2084E+00
 1.7000E+03  2.6991E-01  3.4532E+00
 2.1000E+03  3.1773E-01  3.5785E+00
 4.2000E+03  5.0773E-01  3.5050E+00
 6.0000E+03  6.1976E-01  3.2065E+00
 8.0000E+03  7.1292E-01  2.8808E+00
    0  0     2  1  1.2737E+03  1
 6.3179E-02  3.3088E-01 -2.6847E-01  3.5591E-01
 5.0000E+02  4.4336E-02  9.4904E-01
 9.0000E+02  7.7083E-02  1.3454E+00
 1.3000E+03  1.0639E-01  1.5671E+00
 1.7000E+03  1.3263E-01  1.6894E+00
 2.1000E+03  1.5624E-01  1.7529E+00
 4.2000E+03  2.5023E-01  1.7233E+00
 6.0000E+03  3.0577E-01  1.5791E+00
 8.0000E+03  3.5203E-01  1.4205E+00
    0  0     3  1  1.3955E+03  0
 5.7526E-02  2.8628E-01 -2.4531E-01  3.1267E-01
 5.0000E+02  3.4409E-02  6.8908E-01
 9.0000E+02  6.0280E-02  9.9604E-01
 1.3000E+03  8.4082E-02  1.1823E+00
 1.7000E+03  1.0585E-01  1.2950E+00
 2.1000E+03  1.2576E-01  1.3615E+00
 4.2000E+03  2.0705E-01  1.3946E+00
 6.0000E+03  2.5609E-01  1.3005E+00
 8.0000E+03  2.9733E-01  1.1840E+00
```

2.2.7 DB_SP

```
FAC 0.7.3
Endian = 0
TSess = 1021484432
Type = 7
Verbose = 1
Fe Z =  26.0
NBlocks = 2952

# data blocks begin
NELE = 10
NTRANS = 1
# type 0 is for level population
TYPE = 000000
# block number
IBLK = 5
# block complex
ICOMP = 1*2 2*8
FBLK = 0
FCOMP =
#block level  abs. energy      population
    0     0 -3.12496016E+04  9.99987960E-01

NELE = 10
NTRANS = 1
TYPE = 000000
IBLK = 6
ICOMP = 1*2 2*7 3*1
FBLK = 0
FCOMP =
    1     0 -3.05250918E+04  5.11909866E-06

NELE = 10
NTRANS = 1
TYPE = 000000
IBLK = 7
ICOMP = 1*2 2*7 3*1
FBLK = 0
FCOMP =
    2     0 -3.05230508E+04  6.65310797E-13

NELE = 10
NTRANS = 1
```

```
TYPE = 000000
IBLK = 8
ICOMP = 1*2 2*7 3*1
FBLK = 0
FCOMP =
    3     0 -3.05122168E+04  6.89876970E-06

 ......

# this block is for line emissivity
NELE = 10
NTRANS = 113
# transition type 16->4 transition
TYPE = 001604
# initial block
IBLK = 54
# initial block complex
ICOMP = 1*2 2*7 16*1
# final block
FBLK = 42
# final block complex
FCOMP = 1*2 2*7 4*1
#upper lower  Delta E         emissivity
 1831   158  2.55395828E+02  6.58072258E-06
 1831   163  2.48938431E+02  2.54685597E-06
 1831   164  2.54269775E+02  7.99868485E-06
 1831   165  2.53695114E+02  6.73144177E-06
 1832   157  2.56149841E+02  1.01010974E-05
 1832   158  2.55389191E+02  8.33697595E-06
 1832   165  2.53688477E+02  8.52791436E-06
 1832   166  2.54721527E+02  2.41980142E-05
 1833   160  2.55734634E+02  5.87857039E-06
 1833   167  2.54585327E+02  1.21048633E-05
 1834   159  2.52442841E+02  4.88687328E-06
 1834   160  2.55741287E+02  1.01619125E-05
 1834   167  2.54591980E+02  5.23122890E-06
 ......
```

2.2.8 DB_RT

```
FAC 1.1.0
Endian = 0
TSess = 1021484432
Type = 8
Verbose = 1
Fe Z =  26.0
NBlocks = 87

# data blocks begin
NELE = 9
NTRANS = 49
# block index
```

```
IBLK = 0
# level index within the ion
ILEV = 0
ICOMP = 1*2 2*7
# electron density
EDEN =  1.00000000E+00
# electron energy distribution, 0 for Maxwellian.
EDIST = 0
# parameters for electron energy distribution
NPEDIS = 3
# temperature
  1.00000000E+00
# Emin
  1.00000000E-20
# Emax
  1.00000000E+02
# photon energy density
PDEN =  0.00000000E+00
# photon energy distribution
PDIST = 0
NPPDIS = 3
  2.00000000E+00
  1.00000000E+01
  1.00000000E+05
# population of this block
DENS =  1.00000000E+00
# statistical weight of this block
STWT     =  4.00000000E+00
# each line is contribution to this level or block by other blocks
         NB         TR         CE         RR         AI         CI
   1  2.9983E-10 6.1831E-06 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1*2 2*7
  44  6.0874E-12 0.0000E+00 0.0000E+00 0.0000E+00 8.8866E-04 0.0000E+00 1*2 2*7 6*1
  45  1.0708E-11 0.0000E+00 0.0000E+00 0.0000E+00 1.6693E-18 0.0000E+00 1*2 2*7 7*1
  46  1.7088E-11 0.0000E+00 0.0000E+00 0.0000E+00 1.0329E-27 2.0448E-37 1*2 2*7 8*1
  47  2.5457E-11 0.0000E+00 0.0000E+00 0.0000E+00 7.2230E-34 1.9952E-31 1*2 2*7 9*1
  48  3.1069E-11 0.0000E+00 0.0000E+00 0.0000E+00 3.3995E-38 3.8637E-27 1*2 2*7 10*1
  49  1.2877E-10 0.0000E+00 0.0000E+00 0.0000E+00 4.1495E-41 2.8210E-24 1*2 2*7 11*1
  50  5.2642E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 8.2326E-22 1*2 2*7 12*1
  51  5.5315E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 9.8721E-20 1*2 2*7 13*1
  52  6.3140E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 4.8804E-18 1*2 2*7 14*1
  53  7.3613E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.2070E-16 1*2 2*7 15*1
  54  8.4200E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.7049E-15 1*2 2*7 16*1
  55  9.3432E-11 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.5542E-14 1*2 2*7 17*1
  56  1.5446E-10 0.0000E+00 0.0000E+00 0.0000E+00 8.3172E+00 1.0338E-13 1*2 2*7 18*1
  57  1.2547E-10 0.0000E+00 0.0000E+00 0.0000E+00 1.9623E+00 5.2262E-13 1*2 2*7 19*1
  58  1.2279E-10 0.0000E+00 0.0000E+00 0.0000E+00 5.6143E-01 2.1193E-12 1*2 2*7 20*1
  ......
  -1  0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00   8
  -2  1.0000E+00 6.1831E-06 0.0000E+00 0.0000E+00 1.1178E+01 0.0000E+00   9
  -3  1.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 1.2304E-04  10
  ......
```

2.2.9 DB_DR

```
FAC 0.8.6
Endian = 0
TSess = 1036783874
Type = 9
Verbose = 1
Fe Z =  26.0
NBlocks = 45

# number of electrons of the recombining ion.
NELE = 9
# number of transitions in this block.
NTRANS = 22
# level index of the recombining state.
ILEV = 37
# energy of the recombining state.
E = -2.99679153E+04
# 2J value of the recombing state.
JLEV = 3
# principle quantum number of the captured electron.
NREC = 6
#AI 2J Rec 2J ibase flev fbase NREC L ERes       ETrans     AI Rate     Total Rate  Branching
275  0  37  3    39   -1    -1    6 0 1.2462E+01 0.0000E+00 1.1031E+13  1.1348E+13  2.7940E-02
......
```

2.2.10 DB_AIM

```
FAC 1.0.0
Endian = 0
TSess = 1059655031
Type = 10
Verbose = 1
Fe Z =  26.0
NBlocks = 1

# data blocks begin
NELE = 4
# number of transitions
NTRANS = 87
# channel number (no physical meaning)
CHANNE = 0
# free electron energy grid
NEGRID = 1
  4.70510700E+03
#bound 2J free 2J Delta E     nsub
   13  0   10  1  4.6448E+03  4
#AI rates    DC strength
 3.5842E+13  1.9099E+00
 3.5842E+13  1.9099E+00
   13  0   11  1  4.5958E+03  4
 5.7277E+13  3.0847E+00
 5.7277E+13  3.0847E+00
 ......
```

2.2.11 DB_CIM

```
FAC 1.0.5
Endian = 1
TSess = 1077996307
Type = 11
Verbose = 1
Se Z =  34.0
NBlocks = 1

NELE = 11
NTRANS = 1
ETYPE = 1
NEGRID = 2
  6.80000000E+01
  8.84000000E+02
UTYPE = 1
NUSR = 2
  6.80000000E+01
  8.84000000E+02
#bound 2J   free 2J  Delta E     nsub
     0  1      8  2  2.5178E+03  3
# -1/2 -> -1
 6.8000E+01  1.7855E-03  2.6242E-02
 8.8400E+02  1.8495E-02  2.0646E-01
--------------------------------------------
# -1/2 -> 0
 6.8000E+01  8.9710E-04  1.3185E-02
 8.8400E+02  9.3091E-03  1.0392E-01
--------------------------------------------
# -1/2 -> 1
 6.8000E+01  2.8326E-06  4.1632E-05
 8.8400E+02  3.1205E-05  3.4833E-04
 ......
```

3 FAC Function Reference

This chapter describes the functions available in the PFAC interface which are organized in the package named `pfac`. Each python module is documented in a separate section, where the global variables, functions, classes are listed in alphabetic order. All functions in the extension modules `fac`, `crm`, and `pol` are also available in SFAC interface with identical calling syntax using the 3 executables `sfac`, `scrm`, and `spol`, unless otherwise indicated explicitly. Their usage in SFAC interface is therefore not documented separately. Functions and classes of other python modules are generally not implemented in SFAC interface. In the documentation of each function, the arguments in brackets are optional, arguments separated by "|" are alternative forms of calling syntax, "..." in the argument list denotes variable number of arguments, and keyword arguments are indicated by *key=arg* pair.

3.1 `fac`–Core FAC Module

3.1.1 Variables

There are several global variables one may make use of.

`ATOMICMASS`:
This is a list of atomic masses of all elements in the periodic table. `ATMICMASS`[*i*] is the mass for atom with nuclear charge equal to *i*.

`ATOMICSYMBOL`:
This is a list of strings representing the atomic symbols of all elements in the periodic table. The first element of this list is empty. Therefore `ATOMICSYMBOL`[*i*] is for atom with nuclear charge equal to *i*.

`QKMODE`:
A dictionary mapping the name of radial integral computational modes to its integer values.

```
QKMODE = {
    'default':      -1,
    'exact':        0,
    'interpolate':  1,
    'fit':          2,
    'cb':           3,
    'dw':           4,
    'bed':          5}
```

The corresponding upper case names can also be used in accessing these numbers.
The modes `'exact'`, `'interpolate'`, and `'fit'` are used for collisional excitation and radiative recombination. `'exact'` requires the radial integrals are calculated on the specified collision energy grid as is. `'interpolate'` requires the calculation on the `egrid` energy grid, and interpolated to `usr_egrid` user energy grid. `'fit'` requires the radial integrals to be fitted by an analytic formula. The modes `'cb'`, `'dw'`, and `'bed'` are used for collisional ionization. `'cb'` inidcates the Coulomb-Born values to be used for the radial integrals. `'dw'` requires the radial integrals to be calculated using the distorted-wave approximation. `'bed'` requres the binary-encounter-dipole theory to be used. The mode `'default'` is one of the mode discussed above depending on the atomic process. For collisional excitation, `'default'` is `'exact'`, for radiative recombination, `'default'` is `'fit'`, for collisional ionization, `'default'` is `'bed'`.

`VERSION`:
This is a string representing the version of FAC. It is in the form of *major.minor.release*, where *major*, *minor*, and *release* are numbers.

3.1.2 Functions

The module contains the following functions.

`AIBranch`(*fn, b, f*):

Looking up the autoionization rate from b to f and the total autoionization rate from b in the binary file *fn*. It returns a Tuple consisting of the resonance energy, partial autoionization rate, and total autoionization rate.

`AITable`(*fn, b, f* [, *c*]):

Calculate the autoionization rates between the bound configuration group b and the free configuration group *f*. The results are saved in file *fn*. The optional channel number c can be supplied as an identification of the transition array.

`AITableMSub`(*fn, b, f* [, *c*]):

Calculate the magnetic sublevel autoionization rates and dielectronic capture strength between the bound configuration group b and the free configuration group *f*. The results are saved in file *fn*. The optional channel number c can be supplied as an identification of the transition array.

`AppendTable`(*fn*):

By default, when a new script is executed, existing binary files are overwritten. If instead the new data should be appended to the file, use this function to set the append flag.

`Asymmetry`(*fn, s* [, *m*]):

Calculate the photoionization asymmetry parameters for given relativistic subshells. s is a string which gives the subshells. It should be in spectroscopic notation, e.g., '1s' for $1s_{1/2}$, '2p' for $2p_{1/2,3/2}$ shells, '3p-' for $3p_{1/2}$ and '3p+' for $3p_{3/2}$, etc. The optional m specifies the maximum multipole expansion, counting in the order E1, M1, E2, M2, ..., therefore, $m = 1$ includes only E1, $m = 2$ includes E1 and M1, etc. Default is $m = 1$. The results are stored in file *fn*. For each subshell, the output starts with one line indicating which subshell it is, its nlj values, the ionization energy, the number of energy points, and the value of m used. It is followed by a block, which contains one line for each energy point and tabulates the electron energy, photon energy, total photoionization cross section (10^{-20} cm^2), cross section for electron direction perpendicular to that of photon, the total radiative recombination cross section, the cross section at 90°, and the ratio of $\sigma_\perp/\sigma_\parallel$, where $\sigma_\perp$ is the 90° radiative cross section for photons polarized in the direction perpendicular to the electron direction, and $\sigma_\parallel$ is the 90° cross section for photons polarized in the direction parallel to the electron direction. The last piece of data for each subshell are the B_λ and B_λ^ϕ parameters defined as

$$\frac{d\sigma}{d\Omega} = \frac{\sigma}{4\pi}\left[\sum_{\lambda\geq 0} B_\lambda P_\lambda(\cos\theta) - \sum_{\lambda\geq 2}\lambda^{-1}(\lambda-1)^{-1}B_\lambda^\phi P_\lambda^2(\cos\theta)\cos(2\phi)\right], \tag{3.1}$$

where σ is the total cross section, θ is the angle between the electron and photon directions, ϕ is the azimuth angle of polarization direction of the photon relative to the electron direction.

`AvgConfig`(*c*):

Setup a mean configuration for the optimization of central potential as specified by a string c. The format of the string is the same as in the function `Config`, except that the occupation can be a non-integer number in this routine. The mean configuration setup by this function is effective only if the function `OptimizeRadial` is called with no arguments. Otherwise, the configurations given in that function are used to generate the mean configuration automatically. It is important that this function be called before `OptimizeRadial` and after `ConfigEnergy(0)`, if the later is used.

`BasisTable`(*fn* [,*m*]):

Print out a table of basis wavefunctions and mixing coefficients in the file *fn*. If m is 0, then the basis table for the ordinary atom is given, otherwise, the basis table for atom in magnetic and electric fields are given.

`CECross`(*ifn, ofn, low, up, e* [, *m*]):

Calculate the collision strength and cross sections at energies given in list *e* between levels *low* and *up* by interpolating the data from CE binary data file *ifn*. The results are written in *ofn* in simple ASCII format. Before this routine is called, `MemENTable` must be called to establish the energy table that is consistent with *ifn*. The energy given in *e* is that of the scattering energy if $m = 1$, which is the default, or that of the incident energy if $m = 0$.

`CERate`(*ifn, ofn, low, up, t*):

Calculate the effective collision strength and rate coefficients at temperatures given in list *t* between levels *low* and *up* by interpolating the data from CE binary data file *ifn*. If either *low* or *up* is negative, then the level index is looped over to give rate coefficients for all transitions. The results are written in *ofn* in simple ASCII format. Before this routine is called, `MemENTable` must be called to establish the energy table that is consistent with *ifn*

`CETable`(*fn, low, up*):

Calculate the collision strength for the excitation of states in the configuration group list *low* to those in the group list*up*. The results are saved in the file *fn*.

`CETableEB`(*fn, low, up* [, *m*]):

Calculate the collision strength for the excitation of states in the configuration group list *low* to those in the group list*up*, but for atoms in magnetic and electric fields. The results are saved in the file *fn*. If *m* is 0, then incident electron is assumed to be isotropic, otherwise, cross sections at different incident directions are calculated.

`CETableMSub`(*fn, low, up*):

Calculate the magnetic sublevel collision strength for the excitation of states in the configuration groups *low* to those in the groups *up*. The results are saved in the file *fn*.

`CITable`(*fn, b, f*):

Calculate the collision strength for the ionization of states in the bound configuration groups *b* to those in the free configuration groups *f*. The results are saved in the file *fn*.

`CITableMSub`(*fn, b, f*):

Calculate the magnetic sublevel collision strength for the ionization of states in the bound configuration groups *b* to those in the free configuration groups *f*. The results are saved in the file *fn*.

`CheckEndian`([*fn*]):

Check the byte order of database file *fn*. It returns 0 for little endian and 1 for big endian. If the optional file name *fn* is omitted, the endian for the current platform is returned.

`ClearLevelTable`():

Clear the energy level table in the memory.

`ClearOrbitalTable`([*m*]):

Clear the radial orbital table in the memory. If the optional argument $m = 0$, the entire table is cleared. If $m>0$, only the continuum orbitals are cleared.

`CloseSFAC`():

Close the file containing the SFAC input file converted from the current Python script. This function must be called after `ConvertToSFAC`. Only the statements between the call to `ConvertToSFAC` and `CloseSFAC` are converted to SFAC input file. This routine is only available in PFAC interface.

`Closed`(*s, ...*):

Specify the closed shells in the electronic configurations. It takes variable number of arguments, each of them is a non-relativistic or relativistic shell in the spectroscopic notation. e.g., `2s` for $2s$ shell, `2p-` for $2p_{1/2}$ shell, and `2p+` for $2p_{3/2}$ shell.

`Config`(*c, ..., group=g* | *g, c, ...*):

Add one or more configurations to the configuration group g. In the first form, the group name g is given as a keyword, while in the second form, the first argument must be a group name instead of a configuration. It takes one or more strings for the configuration specification. A configuration c is a string comprised of one or more non-relativistic or relativistic shells in spectroscopic notation separated by white spaces. e.g., `2[p+]3` is a $2p_{3/2}$ shell with 3 electrons. If a `*` is given instead of the orbital angular momentum symbol, configurations with all legitimate values are generated. It is also possible to use `[s,p,d]` to indicate that the orbital angular momentum may take s, p, or d values. For $l > 20$, no spectroscopic symbol is available, it is specified as $[l]$ such as [21,22] for $l = 21$ and 22 shells. This numerical notation also works for $l \leq 20$ shells, for which spectroscopic symbols `s, p, d, f, g, h, i, k, l, m, n, o, q, r, t, u, v, w, x, y, z` are available. The bracket for the orbital angular momentum can be omitted if it comprises of a single character. Otherwise, the bracket must be present. For example, $2p-2$ is illegal, but $2[p-]2$ is. Each shell may be followed by multiple conditions on the occupation number, separated by ";", e.g., $3{*}10; 3s > 0; 3p > 5$ generate configuarations that have at least 1 electron in the $3s$ and 6 electrons in the $3p$ shells. The logical relations allowed in conditions include $=$, $>$, and $<$.

`ConfigEnergy`(*m* [, *n* [, *g, ...*]]):

This function should be called twice just before (with $m = 0$) and after (with $m = 1$) `OptimizeRadial` if used. If `AvgConfig` is called, then `ConfigEnergy` with m=0 must be called before that. The call with $m = 0$ performs a radial optimization for the configuration groups given by the list g, and calculate the average energy of each configuration under such potentials. Multiple optimizations are performed if more than one list are given. If none is given, the optimization are carried out for each configuration group. If $m = 0$, one may also specify an integer n to indicate that `RefineRadial(n)` should be called after the optimization. The call with $m = 1$ does not accept additional arguments. It recalculates the average energy of each configuration under the potential obtained by `OptimizeRadial` issured by the user, The diagonal elements of the Hamiltonian calculated in the `Structure` call is then adjusted by the difference of the two average energies for each configuration. The purpose of this routine is to remove some of the errors in the level energies introduced by using a single central potential for all configurations.

`ConvertToSFAC`(*fn*):

Converts the statements between this call and the `CloseSFAC` call to SFAC input file *fn*. The resulting file can then be run using the `sfac` executable. This routine is only available in PFAC interface.

`CorrectEnergy`(*fn, nmin* | *ilev, e, nmin*):

Correct the energies of certain levels by given amount. This is used if the exact energy of some levels is critical. In the first form, the indexes and the energy corrections are listed as two columns in the file *fn*. In the second form, the indexes are given in the Python list *ilev*, and the energy corrections are given in the list e. Only levels with valence electron in $n \geq nmin$ are corrected, if these levels are not constructed with `RecStates`. The corrections are given in units of eV. For the first level in the correction list, the amount is added to the energies calculated by FAC, and that level is taken as the reference level for all other entries in the list. The correction energies in the list e for these other entries with $ilev \geq 0$ are the desired energies relative to the reference level, i.e., they replace the energies calculated by FAC. However, if $ilev < 0$, then these corrections are also interpreted as shifts to be added to the energyies calculated by FAC.

`CutMixing`(*g0, g1* [, *c*]):

For each levels in the group list *g0*, eliminate all mixing components that are not in the group list *g1* or the mixing coefficients less than c.

`DROpen`(*g*):

Calculate the principle quantum numbers at which the dielectronic recombnation starts to open for the core excitations from the ground state to the states in the configuration group list g. It returns a list of integers.

`GetCFPOld`(*j, q, dj, dw, pj, pw*):

Return the coefficient of fractional parentage $(j^{q-1}, pj, pw|\}j^q, dj, dw)$, where j is the angular momentum of the shell. q is the occupation of the daughter state. pj and dj are the total angular momenta of the parent and daughter states. pw and dw are the seneority of parent and daughter states. All angular momenta appearing here are twice of their actual values.

`GetCG`(*j1, j2, j3, m1, m2, m3*):

Return the Clebsch-Gordan coefficient $<j1m1, j2m2|j3m3>$. All angular momenta appearing here are twice of their actual values.

`GetConfigNR`(*c, ...*):

This functions returns a list of non-relativisitc configurations corresponding to the supplied configuration strings, which may contain wild casts as in the function `Config`. It is useful, e.g., when one wants to know the non-relativistic configurations of `1*2 2*2 3*1`.

`GetPotential`(*fn*):

Print the radial potential obtained by `OptimizeRadial` to the file *fn*. The file starts with the parameters for the analytic fit the to potential, λ and a, in the formula

$$V_0(r) = -\frac{Z}{r} + \frac{N-1}{r}\left(1 - \frac{\exp(-\lambda r)}{1+ar}\right). \tag{3.2}$$

After that, the mean configuration used to generate the potential is printed in 3 columns representing the principle quantum number, the relativistic angular quantum number κ, and the fractional occupation number respectively. Finally, the file gives 8 columns which are i, r, $Z(r)$, $V(r)$, $V_d(r)$, $V_e(r)$, $V_e'(r)$, and $U(r)$, where i is the index for the radial grid, r is the radial grid, $Z(r)$ is the nuclear charge at radius r taking into account the nuclear charge distribution, $V(r)$ is the optimal potential, $V_d(r)$ is the direct interaction part of the potential, $V_e(r)$ is the exchange interaction part of the potential, $V_e'(r)$ is the Slater approximation of the exchange interaction, and $U(r)$ is the Uehling potential which approximates the vacuum polarization effects.

`GetW3j`(*j1, j2, j3, m1, m2, m3*):

Return the Wigner $3j$ symbol $\begin{pmatrix} j1 & j2 & j3 \\ m1 & m2 & m3 \end{pmatrix}$. All angular momenta appearing here are twice of their actual values.

`GetW6j`(*j1, j2, j3, i1, i2, i3*):

Return the Wigner $6j$ symbol $\begin{Bmatrix} j1 & j2 & j3 \\ i1 & i2 & i3 \end{Bmatrix}$. All angular momenta appearing here are twice of their actual values.

`GetW9j`(*j1, j2, j3, i1, i2, i3, k1, k2, k3*):

Return the Wigner $9j$ symbol. All angular momenta appearing here are twice of their actual values.

`Info`():

Print out the version information of FAC and contact information of the author.

`InterpCross`(*ifn, ofn, low, up, e, m*):

Interpolate cross sections in the binary file *ifn*, and save outputs in *ofn*. *low* and *up* are lower and upper indices of the transitions. If either one is -1, then loop over all indices for that level. e is a list of energies where the cross sections are needed. If m is 0, then e is interpreted as the incident energy, if m is 1, then e is interpreted as the incident energy minus transition energy.

`JoinTable`(*fn1, fn2, fn*):

Join two binary files *fn1* and *fn2* to produce a single file *fn*. *fn1* and *fn2* must have been produced on the same platform, have the same type and for the same element.

`LevelInfor`(*fn, ilev*):

If *ilev*≥ 0, look up in the energy database file *fn* for the level indexed *ilev*. It returns a tuple of length 6. The first element is the energy in atomic units, the second is the parity and the valence *nl* values, the third is the 2J value, the fourth is the complex name, the fifth is the non-relativsitic configuration, and the sixth is the relativistic configuration and the coupling. The parity and the *nl* values are encoded as $\pm n * 100 + l$, where $+$ sign is for even parity and $-$ sign is for odd parity. If *ilev*< 0, and its absolute value is less than 1000, it returns the index of the first level with number of electrons equal to $-ilev$. If *ilev*$= -1000$, it returns the index of the first level of the bare ion. If *ilev*$= -1001$, it returns the total number of levels in the file. If *ilev*$= -1002$, it returns the total number of level blocks in the file. If *ilev*≤ -2000, it returns the index of the first level of the block $-ilev - 2000$.

`ListConfig`([*fn, g*]):

Print the configurations in the list *g* to file *fn*. If *fn* is not given or if it is "`-`", the results are written to the stdout. If *g* is not given, then all configurations currently defined are printed.

`MaxwellRate`(*ifn, ofn, low, up, t*):

Calculate the Maxwellian rate coefficients for collision processes with cross section data given by the binary file *ifn*, the results are saved in *ofn*. *low* and *up* are lower and upper indices of the transitions. If either one is -1, then loop over all indices for that level. *t* is a list of temperatures where the rate coefficients are needed.

`MemENTable`(*fn*):

Build an energy level table in the memory for the `DB_EN` type file *fn*. This function must be called before any calls to `PrintTable` in verbose mode.

`ModifyPotential`(*fn*):

Modify the model central potential such that it matches the potential given in the file. The file must specify the potential in two columns corresponding to the radii and potential value in atomic units. This can be useful to import a central potential produced by a different atomic structure code.

`OptimizeRadial`([*g* [, *w*]]):

Obtain the optimal radial potential based on the mean configuration generated by the configuration group list *g* and the weight *w*, or if they are absent in the call, by the mean configuration specified by `AvgConfig`. *g* and *w* must be equal length Python lists if both present. *g* is a list of configuration groups, and *w* is a list of weights for each group when generating the mean configuration. If only *g* is present, each configuration group is given an equal weight.

`PICrossH`(*Z, E, n, l* [, *m*]):

Calculate the photionization cross section of H-like ion with nuclear charge *Z* at photon energy *E*, for the non-relativisitc subshell *nl*. The result is in unit of 10^{-20} cm^2. If the optional parameter *m* is not 0, then the differential oscillator strength is returned instead of the cross sections.

`PrepAngular`(*p* [, *q*]):

Precalculate the angular coefficients between states in *p* and *q*. *p* and *q* are lists of configuration groups. If *q* is not present, the angular coefficients are calculated between states in the *p* list. Only $Z^L(\alpha, \beta)$ and $\tilde{a_\alpha}$ coefficients are calculated. If the Bra and Ket states have the same number of electrons, Z^L is calculated, otherwise $\tilde{a_\alpha}$ is calculated. This routine should primarily be used when atomic states are construed with `RecStates`, where the angular coefficients between the base states are used many times. It is therefore more efficient to precalculate these coefficients.

`Print`(*args*):

Print out the string representation of *args*. This function exists to asist the conversion to SFAC interface, since Python's `print` statement is not converted.

`PrintTable`(*fnb, fna* [, *v*]):

Convert the binary database file *fnb* to the ASCII file *fna*. The optional argument $v = 1$ requires the conversion be done in verbose mode, otherwise it is done in simple mode. Note that before conversion in verbose mode is carried out, one must call `MemENTable` first.

`PropogateDirection`(m):

m specifies the direction in which the R-matrix solution in the outer region is propogated. $m \geq 0$ indicates that the R-matrix in the first zone is propogated outward to the final matching radius. $m < 0$ indicates that the wavefunction at the final matching radius is propogated inward to the first zone and matched to the R-matrix there. Default is $m = 0$.

`RecStates`(*fn, b, n*):

Construct recombined states by adding a spectator electron with the principle quantum number n onto the basis states in the configuration groups *b*. The orbital angular momentum of the spectator electron is set by two functions `SetRecPWLimits` and `SetRecSpectator`. The resulting energy levels are saved in file *fn*.

`RefineRadial`([*n* [, *m*]]):

This function may be called after `OptimizeRadial`, which performs a minimization of the total energy of the mean configuration by adjusting the parameters in the analytic central potential. n is the number of energy evaluations allowed in the minimization, and m controls the print out during the calculation. Default: $n = 100$, $m = 0$ (no print out).

`Reinit`(***mkey* | m):

Reinitialize some or all subsystems of FAC package. In the first form, it accepts variable number of keyword arguments, each represents one subsystem. The following keyword identifiers may be used:

`config`	Electronic configuration and coupling informations. If *config≥0*, exsisting configurations and coupling are cleared.
`recouple`	Angular recoupling package. If *recouple≥0*, all recoupling interaction array is cleared.
`structure`	Atomic structure calculatoin. If *structure≥0*, the energy level table and all angular coefficients are cleared.
`radial`	Radial wavefunction and integrals. If *radial=0*, the radial orbital table, the potential table, all radial integral tables are cleared. If *radial=1*, only continuum orbitals are cleared.
`excitation`	Collisional excitation. If *excitation≥0*, the collision energy grid and radial integrals are cleared.
`recombination`	Photoionization, recombination and autoionization. If *recombination=0*, the energy grid, radial integrals, and recombined complex table are cleared. If *recombination=1*, only the energy grid and radial integrals are cleared.
`ionization`	Collisional ionization. If *ionization≥0*, the energy grid and radial integrals are cleared.
`dbase`	Database handling. If *dbase=0*, the energy table in memory is cleared and all database headers are reinitialized. If *dbase=t* with *t>0*, only the header for the database type *t* is reinitialized.

In the second form, the integer m requires a certain combination of subsystems to be reinitialized. It may be -1, 0, or 1. If *m=0*, then all keywords are set to 0, i.e., a full reinitialization. If *m=1*, keywords `radial`, `excitation`, `recombination`, `ionization`, `structure`, and `dbase` are set to 0, `config` and `recouple` are not reinitialized. This is useful to clear all information related to radial wavefunctions, but keep the configuration data intact. If *m*=-1, all keywords are set to 1.

`ReinitConfig`(m):

Reinitialize electronic configuration module. m the same as correponding keywords in `Reinit`.

`ReinitDBase`(m):

Reinitialize database module. m the same as correponding keywords in `Reinit`.

`ReinitExcitation`(m):
Reinitialize collisional excitation module. m the same as correponding keywords in `Reinit`.

`ReinitIonization`(m):
Reinitialize collisional ionization module. m the same as correponding keywords in `Reinit`.

`ReinitRadial`(m):
Reinitialize radial module. m the same as correponding keywords in `Reinit`.

`ReinitRecombination`(m):
Reinitialize recombination module. m the same as correponding keywords in `Reinit`.

`ReinitRecouple`(m):
Reinitialize recouple module. m the same as correponding keywords in `Reinit`.

`ReinitStructure`(m):
Reinitialize atomic structure module. m the same as correponding keywords in `Reinit`.

`RestorePotential`(*fn*):
Restore the model central potential from a file generated by the `SavePotential` function. This can be useful when one job needs to use the exact same potential as a previous one.

`RMatrixBasis`(*fn, k, nb*):
Calculate the R-matrix basis functions, and buttle corrections, and save in file *fn*. k is the maximum orbital angular momentum of the continuum electron. *nb* is the number of basis functions per-κ orbital.

`RMatrixBoundary`(*r0, r1, b*):
Set the R-matrix boundary conditions. When $r0 = 0$, it sets the normal boundary condition via a call to `SetBoundary`(n, *r1*, b), where n is the maximum principle quantum numbers of the existing states, i.e., the R-matrix target states. When $r0 \neq 0$, it sets the boundary condition in the propagation zone via a call to `SetBoundary`(-100, *r1*, *r0*). In addition, the boundary condition is reset to b.

`RMatrixCE`(*fn, bfn, rfn, emin, emax, de* [, *m*]):

Solve the outer region wavefunction, calculate the collision strength. *fn* is the file for saving results. *bfn* is a list a R-matrix basis files for each R-matrix zone. *rfn* is a list of R-matrix surface files for each R-matrix zone. By default, the R-matrix in the first zone is propogated outward to the last zone, and matched with the outer region solution to obtain S-matrix. If the propogation direction has been set inward in `PropogateDirection`, then the outer region solution is propagated inward to the first zone, and matched with the R-matrix there. *emin*, *emax*, and *de* specifies the energy grid. The entire energy grid is divided into batches to be processed in once pass of all symmetris. The number of energy points in one batch does not exceed 100 by default, or the value set by `RMatrixNBatch`. *m* specfies the output content. If $m = 0$, only the total collision strengths are output. If $m = 1$, the partial collision strengths are output in addition. If $m = 2$, the R-matrix and T-matrix elements are output. If $m = 3$, both partial collision strengths and R-matrix, T-matrix elements are output.

R-matrix, T-matrix output format: each line tabulate, isym, its0, ka0, it1, ka1, et, Real(T), Imag(T), Omega, R, where isym is the symmetry ID. its0 is the target state index for the initial channel, ka0 is the κ value of the continuum electron in the initial channel. its1 and ka1 are corresponding values for the final channel. et is the energy if the system relative to the initial target energy. Real(T) and Imag(T) are the real and imaginary part of the T-matrix, Omega is the partial collision strength of the symmetry. R is the R-matrix. These data are repeated for every energy and every symmetry.

Total and partial collision strength output format: for each transition, first line lists, initial target state index, 2J of initial state, final state index, 2J of final state, transition energy, number of energy points in this block, and the number of partial-waves. It is followed by the block for total collision strengths in two columns: energy, collision strengths. If partial collision strength output is requested, it is followed by three columns: energy, orbital angular momentum of the partial wave, and partial collision strength. The above data are repeated for each block of the energy points processed in the individual batches.

For details of using the R-matrix functions, see the example in `demo/rmatrix/` directory.

`RMatrixConvert`(*ifn, ofn, m*):

Convert R-matrix basis and surface files between binary and ascii format. If $m = 0$, convert binary basis file *ifn* to ascii file *ofn*, if $m = 1$, convert ascii basis file *ifn* to binary file *ofn*, if $m = 2$, convert binary surface file *ifn* to ascii file *ofn*, if $m = 3$, convert ascii surface file *ifn* to binary file *ofn*.

`RMatrixExpansion`(*m* [, *r*]):

Set how the channel wavefunctions at the outmost radius should be calculated. *m* is the number of terms in the Gailitis asymptotic expansion. $m = 0$ indicates use the Dirac-Coulomb functions, which is the default. *r* is the outmost radius where the wavefunctions should be calculated. If outer boundary of the last R-matrix zone, *a* is smaller than *r*, the the wavefunctions at *r* is integrated inward until *a*, and matched with the R-matrix there. If $r \leq a$, then *a* is used as the outmost radius.

`RMatrixFMode`(*m*):

If $m = 0$, use binary format for the R-matrix output files, if $m = 1$, use ascii format for the R-matrix output files.

`RMatrixNBatch`(*n*):

n is the number of energy points to process at one pass of all symmetries in `RMatrixCE`. Default is $n = 100$.

`RMatrixNMultipoles`(*m*):

The maximum multipoles included in the outer region potential. If *m* is larger than the number of multipoles saved in the R-matrix surface file, that smaller value is used. The default is $m = 2$.

`RMatrixSurface`(*fn*):

Calculate the R-matrix surface amplititudes and save in file *fn*.

`RMatrixTargets`(*t* [, *c*]):

Set the target and correlation states. *t* and *c* are lists of configuration groups, for target and correlation configurations respectively. The atomic structure for both *t* and *c* must have been solved with `Structure`.

`RRCrossH`(*Z, E, n, l*):

Calculate the radiative recombination cross section of bare ion with nuclear charge Z at electron energy E, onto the non-relativisitc subshell *nl*. The result is in unit of 10^{-20} cm^2.

`RRMultipole`(*fn, b, f* [, *m*]):

Calculate the bound-free multipole matrix elements. Because `RRTable` does not calculate the matrix elements directly, this function exisits to provide such data. The output are written to the file *fn* in ASCII format. The format is described in the header of the file itself. The remaining arguments are identical to those in `RRTable`

`RRTable`(*fn, b, f* [, *m*]):

Calculate the bound-free differetial oscillator strengths between the bound configuration groups b and free groups *f*, which are related to radiative recombination and photoionization cross sections. The optional multipole type m is set to -1 (E1) by default. In almost all cases, no other multipole types should be important. The results are saved in file *fn*.

`SavePotential`(*fn*):

Save the model central potential in a binary file, which can then be restored using the `RestorePotential` function in a later job

`SetAICut`(c):

Set the autoionization rate cutoff threshold in the output. Only autoionization rates greater than c a.u. are output. The default is 10^{-16} a.u. or ~ 4.13 s^{-1}, if this routine is not called.

`SetAngZCut`(c):

Set the cutoff threshold for the mixing basis in the calculation of recoupling coefficients. Only the basis functions with mixing coefficients $>c$ are included. The default is 10^{-6} if this routine is not called.

`SetAtom`(*asym* [, *z* [, *m* [,*r*]]]):

This function set the atomic element to *asym*, where *asym* is the standard elemental symbol. The nuclear charge *z*, atomic mass *m*, and the nucleus radius of the element can be set optionally. If they are not set, the standard values are used.

`SetBoundary`(n [, p, b]):

Set the boundary condition for high-n orbitals, so that they can represent the continuum. In the normal mode when $n > 0$, n is the maximum principle quantum number represents the bound states. Above n, the orbitals have the boundary conditions imposed, which is in the form $Q/P = (b+\kappa)/2ac$, where Q and P are the small and large components of the wave function, κ is the angular quantum number of the orbital, b is given by the input, c is the speed of light, and a is the boundary radius, which is determined by the criterion that the bound states have densities $< p$ beyond the boundary. The default value of p is 0.001. When $n < 0$ and $n \neq -100$, p is interpreted as the actual radius of the boundary, and $-n$ is interpreted as the maximum principle quantum number of the bound states. When $n == -100$, it sets two boundaries. The radius of the outer boundary is given by p, and that of the inner boundary is given by *b*, if $b > 0$. Otherwise, the inner boundary takes the value of a previous call to `SetBoundary`. This mode is used to set the R-matrix boundary conditions in the propagation zones.

`SetBreit`(n [,m]):

Set the calculaiton mode and maximum principle quantum number of the orbitals for which the Breit interaction should be included in the Hamiltonian. If $n < 0$, then the Breit interaction involving all bound and continuum states is included. Default is 5. m is the calculation mode. *m=0* uses a simplified zero-energy approximaiton for the Breit term. *m=1* uses the full frequency dependent Breit interaction. *m=2* uses the same *m=1* logic, but set the energy of the exchanged photon to 0.

`SetCEBorn`(*e* [,*x* [, *x1*]]):

e specifies the asysmtotic energy where the plane-wave Born approximation is expected to be valid, and is used to deduce the constant component of the collision strength at high energies. If $e > 0$, it is in unit of the characteristic transition energy of the array. If $e < 0$, then its absolute value is the energy in unit of eV. x and $x1$ set the energy boundary between the distroted-wave Born approximation and plane-wave Born approximation for the collisional excitation. If $x > 0$ and the scattered electron energy is less than xE_b, DW method is chosen, where E_b is the binding energy of the electron being excited. If x is negative, the switch between DW and PW occurs when the estimated high partial-wave contributions becomes larger than $-x$ times the low partial-waves calculated explicitly. If $x = 0$, then PW is always used. default for x is -0.5, i.e., use DW when the estimated high partial-wave contributions represents no more than 50% of the explicitly calculated low partial-wave sum. The contributions of high partial waves for monopole and dipole transitions are calculated with coulomb-bethe approximation, their cutoff values are treated differently with the switch *x1*, whose default is -1.0.

`SetCEGrid`(*g* | *n* [, *e0, e1*]):

Set the collsion energy grid for collsional excitation. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points, from *e0* to *e1*. The energies are specified for the scattered electron energy, and in units of eV. This routine does not need to be called. A default is constructed for a given transition array with 6 points, minimum and maximum energies specified by `SetCEGridLimits`. Calling this routine with n=0, reset the grid to system default.

`SetCEGridLimits`(*e0, e1*):

Set the minimum and maximum collision energy for collisional excitation for the automatic construction of the grid. They are in units of average threshold energy of the transition array being considered. The default is 0.05 and 8.0 if this routine is not called.

`SetCELCB`(*m*):

Set the orbital angular momentum for Coulomb-Bethe approximation.

`SetCELMax`(*m*):

Set the maximum of orbital angular momentum for the partial-wave expansion in collisional excitation.

`SetCELQR`(*m*):

Set the maxmum orbital angular momentum for quasi-relativistic approximation in collisional excitation. The default is 0, i.e., always use quasi-relativistic approximation.

`SetCEQkMode`(*mode*):

Set the computation mode for the excitation radial integrals. *mode* may be a string or an integer specifying the mode. These values are listed in the variable `QKMODE`.

`SetCIEGrid`(*g* | *n* [, *e0, e1*]):

Set the collsion energy grid for collsional ionization. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points, from *e0* to *e1*. The energies are specified for the total energy of the final electrons, and in units of eV. This routine does not need to be called. A default is constructed for a given transition array with 6 points, minimum and maximum energies specified by `SetCIEGridLimits`. Calling this routine with n=0, reset the grid to system default.

`SetCIEGridLimits`(*e0, e1*):

Set the minimum and maximum collision energy for collisional ionization for the automatic construction of the grid. They are in units of average threshold energy of the transition array being considered. The default is 0.05 and 8.0 if this routine is not called.

`SetCILCB`(*m*):

Set the orbital angular momentum for the Coulomb-Bethe approximation in DW collisional ionization.

`SetCILevel`(m):

Set the level of configuration interaction space. By default, $m = 0$, the configuration space is determined by the configuration groups passed to the `Structure` function. CI can be further refined by the value of m. If $m = -1$, then no CI is included, the Hamiltonian is assumed to be diagonal. If $m = 1$, only CI within the same relativistic configuration is included. If $m = 2$, only CI within the same non-relativisitc configuration is included. If $m = 3$, only CI within the same configuration group is included.

`SetCILMax`(m):

Set the maximum orbital angular momentum for the partial-wave expansion in DW collisional ionization.

`SetCILMaxEject`(m):

Set the maximum orbital angular momentum for the ejected electron in DW collisional ionization.

`SetCILQR`(m):

Set the orbital angular momentum for quasi-relativistic approximation in DW collisional ionization.

`SetCITol`(m):

Set the tolerance factor in the partial-wave expansion of DW collisional ionization.

`SetCIQkMode`(*mode*):

Set the computation mode for the ionization radial integrals. *mode* may be a string or an integer specifying the mode. These values are listed in the variable `QKMODE`.

`SetFields`(*b, e, a* [*, m*]):

Set the magnetic and electric fiedls. *b* is the magnetic fields in Gauss, *e* is the electrific fields in Volts/cm. *a* is the angle between the magnetic and electric fields. If the optional *m* is 1, then the diamagnetic effects is ignored in the Hamiltonian.

`SetHydrogenicNL`([*n,* [*l*]]):

Set the principle quantum number *n* and the orbital angular momentum *l*, beyond which, the hydrogenic approximation for the E1 multipole integrals should be used. If this routine is not called or the argument is not given, the default is *n*=8, and *l*=7.

`SetIEGrid`(*g* | *n* [*, e0, e1*]):

Set the ionization threshold energy grid for the collisional ionization. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points from *e0* to *e1*. This routine does not need to be called. A 3 point grid is constructed according to the transition array being considered by default. Calling this routine with *n*=0, reset the grid to system default.

`SetMaxRank`(k):

Set the maximum rank in the expansion of Slater integrals. The default is 6, if this routine is not called.

`SetMixCut`(c):

Set cutoff threshold of the mixing basis in the wavefunction. Only the basis with mixing coefficients greater than c are included in the wavefunction expansion. Default is 10^{-5}.

`SetMS`(*nms, sms*):

Set flags for normal mass shift and specific mass shift contributions to the Hamiltonian. 0—disable, 1—enable. Defaults: 1.

`SetNStatesPartition`([*n*]):

Set the number of states in one partition. A partition is used to organize angular integrals. Angular coefficients between partitions are calculated and tabulated in batches. The more states in partitions, the more efficient the lookup of these coeffients, and the more memory it takes. The default value of 512 is normally appropriate.

`SetOptimizeControl`(*t, s, m* [, *p*]):

Set the options for radial potential optimization. *t* is the tolerance for the self-consistent field iteration. *s* is the stablizer for the iteration, a number from 0 to 1. *m* is the maximum number of iterations allowed. *p* specifies whether diagnostic information should be printed out during the optimization. This routine does not need to be called. The default for *t* is 10^{-6}, *s* is determined dynamically according to the type of ion, *m* is 100, and *p* is 0 for no printing out of information.

`SetOptimizeMaxIter`(*m*):

Set the maximum itneration in `OptimizeRadial`.

`SetOptimizePrint`(*m*):

Set the printing option in `OptimizeRadial`.

`SetOptimizeStabilizer`(*m*):

Set the stablilizer factor in `OptimizeRadial`.

`SetOptimizeTolerance`(*m*):

Set the tolerance factor in `OptimizeRadial`.

`SetPEGrid`(*g* | *n* [, *e0, e1*]):

Set the free electron energy grid for photoionization, radiative recombination and autoionization. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points from *e0* to *e1*. The energies are in units of eV. This function does not need to be called. A 6 point grid is constructed according to the transition array being considered by default. Calling this function with *n*=0 reset the grid to system default.

`SetPEGridLimits`(*e0, e1*):

Set the minimum and maximum collision energy for photoionization and radiative recombination for the automatic construction of the grid. They are in units of average threshold energy of the transition array being considered. The default is 0.05 and 8.0 if this routine is not called.

`SetRadialGrid`(*n* [, *r0* [,*r1*] [,*rmin*]]):

Set the radial grid properties. *n* is the number of radial grid points. It must be an even number and less than the macro `MAXRP`. *r0* specifies the ratio of successive radial points near origin, which is approximately logarithmic. *r1* specifies the number of mesh-points per oscillation wavelength for very high-*n* orbitals at large radii. *rmin* divided by the nuclear charge is the starting point of the radial mesh.

`SetRRTEGrid`(*g* | *n* [, *e0, e1*]):

Set the transition energy grid for photoionization and radiative recombination. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points from *e0* to *e1*. This routine does not need to be called. For E1 type transitions, the transition energy does not enter the calculation, a 1-point grid is constructed by default. for other types of multipoles, a 3-point grid is constructed. Calling this function with *n*=0 reset the grid to system default.

`SetRecPWLimits`(*l0, l1*):

Set the orbital angular momentum range for the spectator electron in the recombined states to [*l0*, *l1*] inclusive. The default is [0, 12].

`SetRecPWOptions`(*lmax*):

Set maximum orbital angular momentum of the spectator electron in the recombined states to *lmax*. The allowed values are also limited by the setting of `SetRecPWLimits`. The default is 12.

`SetRecQkMode`(*mode*):

Set the computation mode for the photoionization and radiative recombination radial integrals. *mode* may be a string or an integer specifying the mode. These values are listed in the variable `QKMODE`.

`SetRecSpectator`(*nmin* [, *nfrozen*]):

Set the minimum principle quantum numbers *nmin* and *nfrozen* for the spectator electron. States with $n > nmin$ are constructed with `RecStates` function, and those with $n > nfrozen$ are treated with frozen core approximation. Default for both *nmin* and *nfrozen* are 8.

`SetScreening`(*ns* [,*c* [,*k*]]):

Set the orbital parameters of screening electrons. *ns* is a list of integers which are the principle quantum numbers of the screening orbitals. The optional *c* is the total charge to be screened, whose default is 1.0. *k* is the either 1, -1, or 0. If $k{=}-1$, then the *l=0* orbitals are used. If $k{=}0$, then the the $l{=}ns/2$ orbitals are used. If $k{=}1$, then the $l{=}ns$-1 nodeless orbital is used. The default is $k{=}1$. This function is usually used when additional screening charge is desired for the mean configuration generating the optimal central potential. It is quite experimental, and therefore not recommended for general use.

`SetSE`(*n* [,*m*]):

Set the mode of calculatin and maximum principle quantum number of the orbitals for which the self-energy correction should be included in the Hamiltonian. Default is 5. *n* is the maximum quantum number, *m* is the mode. *m=0* use the hydrogenic screening based on the expectation value of the Uheling potential. *m=1* use the screening based on the expection value of the 2nd derivative of the central potential.

`SetSlaterCut`(*k1, k2*):

Set the calulation modes of Slater integrals. *k1* and *k2* are orbital angular momentum values. When one of the orbitals has $l > k1$, then exchange integrals are not calculated. When one has $l > k2$, the direct integrals are evaluated with the multipole moments.

`SetTEGrid`(*g* | *n* [, *e0, e1*]):

Set the transition energy grid for collisional excitation. In the first form, the grid is given by a Python list *g*. In the second form, the grid is constructed with *n* points from *e0* to *e1*. This routine does not need to be called. A 3-point grid is constructed by default. Calling this function with $n{=}0$ reset the grid to system default.

`SetTransitionCut`(*c*):

Set the cutoff threshold for the radiative transition rates output. Only rates greater than *c* times the total decay rate of the upper level is written to the output file. The default is 10^{-3} if this routine is not called.

`SetTransitionGauge`(*m*):

Set the gauge for radiative transition.

`SetTransitionMaxE`(*m*):

Set the maximum rank of electric multipole transitions.

`SetTransitionMaxM`(*m*):

Set the maximum rank of magnetic multipole transitions.

`SetTransitionMode`(*m*):

Set the mode for the radiative transition.

`SetTransitionOptions`(*g, m*):

Set the options for the radiative transition calculation. *g* is the gauge to be used, 1 for Coulomb gauge (velocity form) and 2 for Babushkin gauge (length form, which is the default). *m* is the mode for the multipole integral, 0 for fully relativistic and 1 for non-relativistic approximation (the default is 1).

`SetUsrCEGrid`(*g* | *n* [, *e0, e1*]):

Set the user collision energy grid for collisional excitation. The collsion strengths on this grid are output. It is forced to be same as that set by `SetCEGrid` if the `QKMODE` is `'exact'`.

`SetUsrCIEGrid`(*g* | *n* [*, e0, e1*]):

Set the user collision energy grid for collisional ionization. The collsion strengths on this grid are output.

`SetUsrPEGrid`(*g* | *n* [*, e0, e1*]):

Set the user electron energy grid for photoionization and radiative recombination. The bound-free differential oscillator strengths on this grid are output. It is forced to be same as that set by `SetPEGrid` if the `QKMODE` is `'exact'`.

`SetUTA`(*m* [*, ci*]):

Set the flag for configuration average models. If $m = 1$, all calculations are carried out in the configuration average approximation. The radiative transition rates output contains three additional fields, the transition energy including the UTA shift, the Gausian standard deviation, and the correction to the line strengths due to the configuration interaction within the same non-relativisitc configurations. If $m = 0$, which is the default, the usual detailed term accounting method is used. This function should be called in the beginning of the script, before `Config` function, since it disables the angular momentum coupling performed by `Config`. The optional argument *ci* indicates whether the configuration interaction correction factors should be included when reading the `DB_TR` files. These correction factors are always calculated in `TransitionTable` irrespective of the value of *ci*.

`SetVP`(*vp*):

Set the flag for vacuum polarization correction to the Hamiltonian. 0—disable, 1—only include 2nd order term, 2—include the 4th order term as well. Default is 2.

`SlaterCoeff`(*fn, g, a, b*):

Calculate the expansion coefficients of the exchange radial integral in the Coulomb energy of each state in the configuration list *g*. The Coulomb energy betwee electrons of a state can be expanded as

$$E = \sum g_k G^k(\alpha\alpha', \beta\beta') + \text{direct terms}, \tag{3.3}$$

where α and α' are the interacting orbitals in the bra and ket states, which have the same l values. Because we work in the jj-coupling basis, α and α' may have different j values. The results are stored in the text file *fn*. These coefficents are useful in determining the radiative transition rates from level to average configurations. *a* and *b* are the orbital lists contain α and β, respectively. They are specified as configuration strings. The format of the file is as follows. For each state, a line starts with "#" gives the level index, parity, $2J$-value, and the configuration label. It is followed by a block of lines. In this block, the first column is the level index. The 2rd, 3th, and 4th columns are the n, *kappa*, l values of the α orbital. The 5th column is either 0 or 1, indicating whether α and α' have the same j values. The 6-9th columns are the corresponding values for the β orbital. The 10th column is the expansion coefficients g_1. The 11th column is g_2. The 12th column is the the number to be added to g_1 to form the relative intensities of level to configuraiton transitions for dipole transitions. The 13th column is the number to be added to g_2 to form relative intensities for quadrupole tranisions.

`Structure`(*fn, g* [*, p* [*, ip*]]):

Diagonalize the Hamiltonian for configurations in the groups *g*. The configurations in the optional groups *p* are allowed to interact with *g* but only states within *g* are added to the energy level table. If *ip*=0, the interaction between *g* and *p* are treated exactly, if *ip*=1, this interaction is treated approximately in a way that the non-diagonal elements within *p* are neglected. The energy levels are output to the file *fn*.

`Structure`(*p, j*):

This is the second form of the `Structure` function. It distinguishes itself from the first form because the first argument is an integer. It sets which πJ symmetry and/or which level to include in the structure calculation. By default, all symmetries are processed. But if this function is called with $p = 0$, or 1; $j \geq 0$ or j is a list of integers, then only the specified symmetry is processed. j or integers in the list if j is a list, is twice the actual value of the total angular momentum. The symmetry restrictions specified here also applies to the `StructureMBPT` function.

`StructureEB`(*fn, g*):

Calculate the atomic structure for atom in magnetic and electric fields. The levels belong to the configuration group *g* are allowed to mix in the external fields.

`StructureMBPT`(*efn, hfn, g, kmax, n1, n2, n0*):

There are five different forms of this function with different argument list. This is the primary form, and the rest are documented below. This function calculates the level energies of the states belonging to the configuration groups in the *g* list. The energy values are stored in the binary file *efn* the same way as `Structure` function. The effective Hamiltonian of the second order MBPT is stored in the file *hfn*. *kmax* is the maximum orbital angular quantum number of the virtual states. *n1* and *n2* are two lists for the grid of principle qunatum numbers of the virtual states. The *n1* grid is used for one electron excitations, and the first electron of the double excitations. The *n2* grid is for the second electron of the double excitation. *n0* is an integer specifying the number of configuration groups in the *g* list to be included in the MBPT calculation, and the remaining are included for all-order perturbation treatment.

`StructureMBPT`(*fn, de, eps, g, kmax, n1, n2, n3, n4, gn*):

This is the second form of the `StructureMBPT` function. This function is depricated. It examines the configurations by exciting the electrons in the *g* list to orbitals with principle quantum numbers in the *n1* and *n2* lists from those in the *n3* and *n4* lists. If its estimated CI mixing to the states in *g* is larger than *eps*, and the excitation energy is less than *de*, these configurations are put in to the group named *gn*. The final list of configurations are written to the file *fn*. *kmax* is the maximum orbital angular momentum of the excited electrons.

`StructureMBPT`(*efn, hfn1, hfns, g, n0*):

This is the third form of the `StructureMBPT` function. It reads the effective Hamiltonians in a list of files *hfns*, that are produced in previous `StructureMBPT` calls in the first form, and combines the contributions from different *n1* and *n2* values for the virtual state and sum them up to form the total effective Hamiltonian, which is saved to the text file *hfn1*. It then diagonalizes the effective Hamiltonian to obtain the energy values, which are stored in *efn* the same way as in `Structure`. The configuration list *g* and *n0* must be the same as those used in generating the *hfns* files.

`StructureMBPT`(*m*):

This is the fourth form of the `StructureMBPT` function. If *m* is an integer, it sets an option to indicate whether extrapolation beyond the maximum *n*-values in the *n1* and *n2* grid is carried out in forming the effective Hamiltonian. If $m = 0$, no extrapolation, which is the default; if $m > 0$, with extrapolation. *m* may also be a list of integers, which means that only levels in this list for each symmetry is to be corrected with MBPT.

`StructureMBPT`(*i, m, c*):

This is the fifth form of the `StructureMBPT` function. It sets three options. *i* indicates inclusion of some part of 3rd order corrections. *m* indicates whether only single excitation or double excitation should be included. If $m = 0$, both single and double excitations are included; if $m = 1$, only single excitations are included, and if $m = 2$, only double excitations are included. *c* is the cutof of the mixing coeffients. The effective Hamiltonian matrix element h_{ij} is not calculated, if $max(|b_{ki}b_{kj}|) < c$. The default is $c = 10^{-4}$.

`TotalCICross`(*ifn, ofn, ilev, energy* [*, imin, imax*]):

Calculate the total collisional ionization cross sections of level *ilev* by reading the data from the `DB_CI` database file *ifn*. The results are written to the ASCII file *ofn* in a two column format. The output cross sections are in units of 10^{-20} cm^2. *ofn* may be "-", which means `stdout`. *energy* is a list or tuple giving the incident electron energies in eV where CI cross sections are needed. The optional *imin* and *imax* specifies the index range of the ionized states. Only ionization to states with $imin \le i \le imax$ are included. If *imin*<0, it is assumed to be 0, and if *imax*<0, it is assumed to be the largest index in the level file. Both *imin* and *imax* default to -1, i.e., include all ionized states. Before calling this function, `MemENTable()` must be called.

`TotalPICross`(*ifn, ofn, ilev, energy* [*, imin, imax*]):

Calculate the total photoionzation cross sections of level *ilev* by reading the data from the `DB_RR` database file *ifn*. The results are written to the ASCII file *ofn* in a two column format. The output cross sections are in units of 10^{-20} cm^2. *ofn* may be "-", which means `stdout`. *energy* is a list or tuple giving the incident photon energies in eV where PI cross sections are needed. The optional *imin* and *imax* specifies the index range of the ionized states. Only ionization to states with $imin \leq i \leq imax$ are included. If *imin*<0, it is assumed to be 0, and if *imax*<0, it is assumed to be the largest index in the level file. Both *imin* and *imax* default to -1, i.e., include all ionized states. Before calling this function, `MemENTable()` must be called.

`TotalRRCross`(*ifn, ofn, ilev, energy* [*, n0, n1, nmax, imin, imax*]):

Calculate the total radiative recombination cross sections onto level *ilev* by reading the data from the `DB_RR` database file *ifn*. The results are written to the ASCII file *ofn* in a two column format. The output cross sections are in units of 10^{-20} cm^2. *ofn* may be "-", which means `stdout`. *energy* is a list or tuple giving the electron energies in eV where the RR cross sections are needed. The data for recombination onto $n0 < n < n1$ and $n > n1$ are not present in the data file *ifn*, and the hydrogenic approximation for $n0 < n < n1$ and $n1 < n \leq nmax$ are added to the calculated cross sections. The optional *imin* and *imax* specifies the index range of the recombined states. Only recombination to states with $imin \leq i \leq imax$ are included. If *imin*<0, it is assumed to be 0, and if *imax*<0, it is assumed to be the largest index in the level file. Both *imin* and *imax* default to -1, i.e., include all recombined states. Before calling this function, `MemENTable()` must be called.

`TransitionMBPT`(*m, n*):

m specifies the maximum multipole for radiative transitions calculated with MBPT. *n* specifies the number transition energy points used in the relativistic radial multipole integrals.

`TransitionMBPT`(*fn, low, up*):

This function specify the transitions between *low* and *up* configuration groups to be calcualted with MBPT, with transition rates saved in the file *fn*. This function must be issued before the 3rd form of `StructureMBPT`, as the actual calculations are carried out in that function.

`TransitionTable`(*fn, low, up* [*, m*]):

Calculate the weighted oscillator strength and radiative transition rates from states in *up* groups to states in *low* groups with multipole type *m*. The default for *m* is -1, i.e., E1 transitions. The results are saved to file *fn*.

`TRBranch`(*fn, upper, lower*):

Looking up the radiative decay rate from *upper* to *lower* and the total decay rate of *upper* in the binary file *fn*. It returns a Tuple consisting of the transition energy, partial decay rate, and total decay rate.

`TRTable`(*fn, low, up* [*, m*]):

Same as `TransitionTable`

`TRTableEB`(*fn, lo, up* [*, m*]):

Calculate radiative transition rates between levels in external magnetic and electric fields.

`TRRateH`(*z, n0, l0, n1, l1* [*, m*]):

Calculate the radiative transition rate or weighted oscillator strength of the transition from state (*n1, l1*) to state (*n0,l0*) in the non-relativistic hydrogenic approximation for the ion with charge *z*. If *m*=0, the transition rate is returned, which is the default, if *m*=1, the weighted oscillator strength is returned, and if *m*=2, the dipole radial integral is returned.

`WaveFuncTable`(*fn, n, k* [*, e*]):

Print the radial wavefunction of the orbital with the principle quantum number *n*, relativistic angular quantum number κ=*k* to the file *fn*. If *n*=0, the orbital is a continuum state. In this case, the optiontal *e* must be a positive number for the energy of the continuum orbital in unit of eV.

`Y5N`(*n, l, r*):
Calculate the Seaton's $y5$ (normalized to give the Coulomb wavefunction) function for negative energies. This is basically the Coulomb wavefunction with principle quantum number n and orbital angular momentum l at radius r. It returns a tuple (y5, y5p, err), where y5 is the value, y5p is the derivative. err is an error code returned by coulcc, which is called internally to do the calculation.

3.2 `crm`–Collisional Radiative Model

The module `crm` implements a collisional radiative spectral model for optically thin plasmas. It uses an iterative linear equation solver to invert the level population equations. This method is capable of including a very large number of atomic states in the model.

3.2.1 Functions

`AddIon`(*n, den, pref*):
Add an ion to the spectral model. n is the number of electrons for the ion to be added. *den* is the density of the ion, and *pref* is the base file name for the binary data files associated with this ion. The standard file extensions are assumed. i.e., `.en` for `DB_EN`, `.tr` for `DB_TR`, `.ce` for `DB_CE`, `.rr` for `DB_RR`, `.ai` for `DB_AI`, `.ci` for `DB_CI`, `.sp` for `DB_SP`, and `.rt` for `DB_RT`. This function is called multiple times to add more than one ion to the model. However, the order must be such that the number of electrons are in consecutive increasing order.

`Cascade`():
Carry out the cascade iteration. Some of the levels in the spectral model may be treated approximately using the cascade matrix.

`CBeli`(*Z, n, E*):
Calculate the ionization cross sections using the Aladdin database. data were compiled by Bell et al. (J. Phys. Chem. Ref. Data., 12, 891, 1983). Z is the nuclear charge of the ion. n is the number f electrons, and E is the electron energy in eV. It returns a tuple of length 4. First element is the total ionization cross section, 2nd the excitation autoionization contribution, 3rd the direct ionization contribution, and the 4th is a relative error estimate in pencentages.

`CFit`(*Z, n, Te*):
Calculate the ionization rates using the Fortran subroutine `cfit` of D. A. Verner for direct ionization, and `colfit` for exciation autoionization. Z is the nuclear charge of the ion, n is the number of electrons, and Te is the electron temperature in eV. It returns a tuple of length 3. First element is the total ionization rate coefficients in 10^{-10} cm^3 s^{-1}., 2nd is the excitation autoionization contribution, and the 3rd is the direct ionization contribution.

`CheckEndian`([*fn*]):
Check the byte order of database file *fn*. It returns 0 for little endian and 1 for big endian. If the optional file name *fn* is omitted, the endian for the current platform is returned. This function exists in module `fac` as well.

`CloseSCRM`():
Close the file containing the SFAC input file converted from the current Python script. This function must be called after `ConvertToSCRM`. Only the statements between the call to `ConvertToSCRM` and `CloseSCRM` are converted to SFAC input file. This routine is only available in PFAC interface.

`ColFit`(*Z, n, Te* [, *s*]):

Calculate the direct ionization rates and excitation autoionization rates using the Fortran subroutine `colfit` of D. A. Verner. *Z* is the nuclear charge of the ion, *n* is the number of electrons, and *Te* is the electron temperature in eV. The optional argument *s* specifies the subshell for which the rates are to be calculated. 0 for total. 1 for $1s$, 2 for $2s$, 3 for $2p$, 4 for $3s$, 5 for $3p$, 6 for $3d$ and 7 for $4s$. The function returns a tuple of length 3. The first element is the total rate, the second is the excitation autoionization rate, and the third is the direct ionization rate. The result is in unit of 10^{-10} cm^3 s^{-1}.

`ConvertToSCRM`():

Converts the statements between this call and the `CloseSCRM` call to SFAC input file *fn*. The resulting file can then be run using the `scrm` executable. This routine is only available in PFAC interface.

`DRBranch`():

Calculate the radiative branching ratios of all autoionizing states. It is calculated iteratively using:

$$B_i = \frac{\sum_j A^r_{ij} B_j}{\sum_j A^a_{ij} + \sum_k A^r_{ik}}, \tag{3.4}$$

where A^r_{ij} is the radiative decay rate from state i to state j, A^a_{ik} is the autoionization rate from state i to state k. The branching ratios of non-autoionizing states is 1.0. This function must be called before `DRStrength()` with *mode*=0.

`DRFit`(*Z, n, Te*):

Calculate the dielectronic recombination rates using the fitting formula and data table of P. Mazzotta. *Z* is the nuclear charge of the ion, *n* is the number of electrons, and *Te* is the electron temperature in eV. The result is in unit of 10^{-10} cm^3 s^{-1}.

`DRStrength`(*fn, n* [, *m, i*]):

Tabulate the dielectronic recombination resonance strengths onto state *i* of the ion with *n* electrons. *fn* specifies the file name of the database file. The optional *m* indicates the mode of calculation. If *mode*=0, then the total DR strengths are calculated; if *m*=1, then individual DR satellite lines are caclulated, and if *m*=2, then resonance excitation is calculated. *i* is optional, whose default is 0. When $i \geq 0$, then it is the index of the recombining state relative to the ground state of the recombining ion; when $i<0$, then its negative value is the true index of the recombining state.

`DumpRates`(*fn, nele, m* [*, imax* [*, a*]]):

Dump rate coefficients to a file. *fn* is the output file name. *nele* is the number of electrons of the ion whose rates are needed. *m* specifies which rate is output.

$m = 0$: Dump the level indexes, 2J values, energies, etc. The record for each level contains 11 fields of the type: short, int, int, int, short, short, short, double double, double, double. They are number of electrons of the ion, level index, block index of the level, level index within the block, 2J value of the level, base level of the level, valence *nl* of the level, energy, population, total decay rate, and branching ratio computed in `DRBranch` (if that has been called) respectively.

$m = 1$: Dump the radiative transition rates. Each record contains 4 fields of type: int, int, double, double. They are upper level index, lower level index, decay rate, and photo-excitation rate, respectively.

$m = 2$: Dump the 2-photon transition rates. Each record contains 4 fields of type: int, int, double, double. They are upper level index, lower level index, decay rate, and inverse rate, respectively.

$m = 3$: Dump the collisional excitation rates. Each record contains 4 fields of type: int, int, double, double. They are lower level index, upper level index, excitation rate, and deexcitation rate, respectively.

$m = 4$: Dump the radiative recombination rates. Each record contains 4 fields of type: int, int, double, double. They are continuum level index, recombined level index, recombination rate, and photoionization rate, respectively.

$m = 5$: Dump the autoionization rates. Each record contains 4 fields of type: int, int, double, double. They are bound level index, continuum level index, autoionization rate, and dielectronic capture rate, respectively.

$m = 6$: Dump the collisional ionization rates. Each record contains 4 fields of type: int, int, double, double. They are bound level index, continuum level index, ionization rate, and three-body recombination rate (not implemented), respectively.

The units of the transition rates are s^{-1}, and those of the rate coefficients are 10^{-10} cm^3 s^{-1}. The optional augument *imax* indicates the maximum levels to be included in the dump. a negative number means all levels, which is the default. *a* specifies whether the file should be in a binary format (0) or ascii format (1). Default is binary format.

`EBeli`(*Z, n*):

Calculate the ionization threshold of ion with nuclear charge *Z* and number of electrons *n*. It returns a double in eV. Data are from aladdin data base.

`EColFit`(*Z, n, s*):

Calculate the ionization threshold of ion with nuclear charge *Z* and number of electrons *n* for shell *s*. *s* has the same meaning as in `ColFit`. It returns a double in eV. Data are from subroutine `ColFit`.

`EleDist`(*fn, n*):

Calculate the electron energy distribution function, and print it to file *fn*, using *n* energy points. The distribution must be already set.

`EPhFit`(*Z, n, s*):

Calculate the ionization threshold of ion with nuclear charge *Z* and number of electrons *n* for shell *s*. *s* has the same meaning as in `PhFit`. It returns a double in eV. Data are from subroutine `PhFit`.

`FracAbund`(*Z, Te* [*im, rm*]):

Calculate the fractional abundance of the charge states for element with nuclear charge *Z* at temperature *Te* for collisionally ionized plasmas. The optional arguments *im* and *rm* specifies the functions used for ionization rates and recombination rates respectively. *im* is passed to the function `Ionis`, and *rm* is passed to the function `Recomb`. The default for both parameters are 1. It returns a list of length $Z{+}1$ indexed by the number of electrons the ion have.

`InitBlocks`():

Initialize the superlevel blocks of the spectral model.

`IonDensity`(*fn, k*):
Read the `DB_SP` file *fn* and return the total density of the ion with number of electrons *k*.

`Ionis`(*Z, n, Te* [*, m*]):
Caculate the ionization rate coefficients for the ion with nuclear charge Z and number of electrons n, at the temperature *Te*. It returns a list of length 3. The first element is the total ionization rate, the second is the excitation-autoionization rate, and the third is the direct ionization rate. The data used are from **??** if the optional argument m is 0. The function `ColFit` is used if m is 1. The function `CFit` is used for the direct ionization rate and `ColFit` for the autoionization rate, if m is 2. The data from Aladdin database is used if m is 3. The default is 1. The results are in unit of 10^{-10} cm^3 s^{-1}.

`LevelPopulation`():
Solve the level population using the superlevel block method.

`MaxAbund`(*Z, n* [*,im, rm, a*]):
Calculate the temperature where the fractional abundance of the ion with nuclear charge Z and number of electrons n reaches maximum. The optional arguments *im* and *rm* specifies the functions used for ionization rates and recombination rates respectively. *im* is passed to the function `Ionis`, and *rm* is passed to the function `Recomb`. The default for both parameters are 1. The parameter a specifies the relative accuracy for the solution, which has the default of 10^{-4}. This functions returns a tuple of length 2. The first element is the temperature found, the second is a list for the fractional abundances of all ions with nuclear charge Z at the found temperature.

`NormalizeMode`(*m*):
Set the mode for normalizing the ion densities. If m is 0, the density of the ground state of each ion is fixed at the value given by `SetAbund`. If m is 1, the total density of the ion is fixed at that value.

`NDRFit`(*Z, n, Te*):
Calculate the dielectronic recombination rates using the data calculated with FAC for H-like through Ne-like ions of Mg, Si, S, Ar, Ca, Fe, and Ni. Z is the nuclear charge of the ion, n is the number of electrons, and *Te* is the electron temperature in eV. The result is in unit of 10^{-10} cm^3 s^{-1}.

`NRRFit`(*Z, n, Te*):
Calculate the radiative recombination cross sections using the Fortran subroutine `nrrfit` using the data calculated with FAC for Bare through F-like ions of Mg, Si, S, Ar, Ca, Fe. Z is the nuclear charge of the ion, n is the number of electrons of the recombining ion, and *Te* is the electron temperature in eV. The result is in unit of 10^{-10} cm^3 s^{-1}.

`PhFit`(*Z, n, E, s*):
Calculate the photoionization cross section at energy E of subshell s for the ion with nuclear charge Z, and number of electrons n. The meaning of s are: 1 for 1s, 2 for 2s, 3 for 2p, 4 for 3s, 5 for 3p, 6 for 3d and 7 for 4s. The Fortran subroutine `phfit2` of D. A. Verner is used. The result is in unit of 10^{-20} cm^2

`PhoDist`(*fn, n*):
Calculate the photon energy distribution function, and print it to file *fn*, using n energy points. The distribution must be already set.

`PlotSpec`(*ifn, ofn, n, t, e0, e1, de* [*, s*]):
Print the spectrum of lines in the `DB_SP` file *ifn* of a given type t to the file *ofn*. See `SelectLines` for the meaning of this parameter except when $t = 0$, in which case all lines are included in the output. The lines are convolved with a Gausian with FWMH of *de* in unit of eV. *e0* and *e1* are the spectral range to be considered. The optional s gives a cutoff threshold for the lines. Lines weaker than s times the strongest line are not included.

`Print`(*args*):

Print out the string representation of *args*. This function exists to asist the conversion to SFAC interface, since Pythoon's `print` statement is not converted.

`PrintTable`(*fnb, fna* [, *v*]):

Convert the binary database file *fnb* to the ASCII file *fna*. The optional argument $v = 1$ requires the conversion be done in verbose mode, otherwise it is done in simple mode. Note that before conversion in verbose mode is carried out, one must call `MemENTable` first. This function also exists in the module `fac`.

`RateTable`(*fn* [, *cfg* [, *m*]]):

Output rates for all processes included in the spectral model to the `DB_RT` database file *fn*. It may contain an additional argument, which is a list of complex names. m controls the amount of information to be output.

`RBeli`(*Z, n, T*):

Calculate the ionization rate coefficients using the Aladdin data base. data were compiled by Bell et al. (J. Phys. Chem. Ref. Data., 12, 891, 1983). Z is the nuclear charge of the ion. n is the number f electrons, and T is the electron temperature in eV. It returns a tuple of length 3. First element is the total ionization rate coefficients in 10^{-10} cm^3 s^{-1}., 2nd is the excitation autoionization contribution, and the 3rd is the direct ionization contribution.

`Recomb`(*Z, n, Te* [, *m*]):

Caclulate the recombination rate coefficients for the ion with nuclear charge Z and number of electrons n at temperature *Te*. It returns a tuple of length 3. The first element is the total recombination rate, the second is the radiative recombination rate, and the third is the dielectronic recombination rate. The data used are from **??** if the optional argument m is 0. The functions `RRFit` and `DRFit` are used if m is 1. If m is 2, then the New RR and DR rate coefficients are used for Bare through Ne-like ions of Mg, Si, S, Ar, Ca, Fe, and Ni calculated with `NRRFit` and `NDRFit`, for other ions, `RRFit` and `DRFit` are used. The default is 1. The results are in unit of 10^{-10} cm^3 s^{-1}.

`ReinitCRM`([*m*]):

Reinitialize the module `crm`. m=0 by default, which requests a full reinitialization. If m=1, only the database headers are reinitialized and the rates are cleared. If m=2, only the database headers are reinitialized and the rates that depend on electron energy distribution are cleared. If m=3, only the database headers are reinitialized.

`RRFit`(*Z, n, Te*):

Calculate the radiative recombination cross sections using the Fortran subroutine `rrfit` of D. A. Verner. Z is the nuclear charge of the ion, n is the number of electrons, and *Te* is the electron temperature in eV. Note that in this routine, n is the number of electrons of the recombining ion, while in the original subroutine of Verner, it is for the recombined ion. The result is in unit of 10^{-10} cm^3 s^{-1}.

`RRRateH`(*Z, n, Te*):

Caculate the radiative recombination rates of H-like ions with nuclear charge Z at temperature *Te*. n is the principle quantum number of the recombined electron. This function returns a tuple of length 2. The first element is the RR rate on to all states with principle quantum number n. The second is the RR rate on to all states with principle quantum numbers $>n$.

`SelectLines`(*ifn, ofn, n, t, e0, e1* [*, s*]):

Print the selected lines from the `DB_SP` database file *ifn* to the file *ofn*. n is the number of electrons of the ion. *e0* and *e1* are the energy range in units of eV. s is the cutoff threshold for the lines. If $s<0$, then *e0* and *e1* are interpreted as the lower and upper level indexes for the line selected. In this case a single line will be output, and t is ignored. Otherwise, t is the type of the lines to be selected. If t is 0, then all transition types are allowed. Otherwise, the value of t is decomposed into 4 fields, say, *t0*, *t1*, *t2*, and *t3*, where *t0* is the lowest 2 decimal digits of t, *t1* is the next 2 digits, and so on. e.g., if t=1000201, then *t0*=1, *t1*=2, *t2*=0, and *t3*=1. If *t3*=0, then the lines have type equal to $10000t2+100t1+t0$ are selected. If *t3*=1, then lines of type $10000q+100t1+t0$ with $q \geq t2$ are selected. If *t3*<0, then lines of type q with $q \geq t0$ are selected. The physical meaning of line types are discussed in §2.1.15

`SetAIRates`(*inv*):

Set autoionization rates in the spectral model. If *inv*=1, the rates for the inverse process, dielectronic capture, are also set.

`SetAIRatesInner`(*fn*):

Read the AI file *fn* to obtain the inner Auger transition rates. This file must only contain the the rates of such transitions. and the energy level indexes of the bound states must be continuous, and have exactly the same order as the corresponding levels in the level table already setup.

`SetAbund`(*n, den*):

Set the abundance of the ion with number of electrons n to *den*. This overrides the settings given by `AddIon`.

`SetBlocks`([*den, pref*]):

Read the energy levels and setup the superlevel blocks for the spectral model. The optional *den* is the abundance of the ion with one less electron than the lowest charge state included in the model, and *pref* is the base file name for that ion. If *den* is < 0, only processes connects the levels within the same ion are included. If *pref* is not specified, the transitions in that ion are not included in the model, which may cause convergence problems in some cases.

`SetCERates`(*inv*):

Set the collisional excitation rates. If *inv*=1, the inverse process, collisional deexcitation rates are also set.

`SetCIRates`(*inv*):

Set the collisional ionization rates. If *inv*=1, the inverse process, three-body recombination rates are also set (not implemented yet).

`SetCascade`(*c* [*, a*]):

If c=1, some levels should be treated in cascade approximation. The optional a specifies the accuracy in the cascade iteration. The default is 10^{-4}.

`SetEleDensity`(*den*):

Set the electron density in unit of 10^{10} cm^{-3}.

`SetEleDist`(*i, ...*):

Set the electron energy distribution. The first argument i specifies the type of distributions, and the remaining give the parameters of the distribution. The avaliable distributions and their parameters are listed in §2.1.18. In addition of these distributions, one can use a text file to define a distribution. It is specified with $i = -1$ and a file name as the parameters. The text file must contain three integers in the first line, specifying the number of energy points, n, the scale of energy and distribution (1 for log scale, 0 for linear scale). It is then followed by n lines giving energy in eV, and distribution function values.

`SetExtrapolate`(*i*):

Set if the extrapolation of the high-n levels should be carried out. If *i* is negative, then no extrapolate is carried out, if *i* is non-negative, then only the DR channels <= *i* will be included in the extrapolation. The DR channels are defined in the order KLn, KMn, ... for K-shell ions, and LLn, KLn, LMn, MMn, ... for L-shell ions. If unsure how to extrapolate, set it to -1 to disable it.

`SetInnerAuger`(*i*):

Set if the inner-shell Auger transitions should be extrapolated to those levels that are not explicitly calculated. There are several different ways such extrapolation can be done. *i*=0 indicates no extrapolation. *i*=1 is to extrapolate by averaging the Auger rates of the last calculated complex. *i*=2 is to extrapolate by reading an extra file that contains the Auger rates in the core, which must be supplied by calling `SetAIRatesInner`. *i*=3 is to extrapolate by reading the same AI file that contains the normal AI rates. In this case, the AI file must contain those rates between the core. *i*=4 is to read the rates from the AI file of the next higher charge state.

`SetIteration`(*a* [, *s*, *m*]):

Set the options for population iteration. *a* is the accuracy, which defaults to 10^{-4}. *s* is a stablizer defaults to 0.75, and *m* is the maximum number of iterations allowed, which defaults to 256.

`SetNumSingleBlocks`(*n*):

Set the number of states in the beginning of the energy table that should not be grouped into a super block.

`SetPhoDensity`(*den*):

Set the photon energy density in unit of erg cm^{-3}.

`SetPhoDist`(*i, ...*):

Set the photon energy distribution. The first argument *i* specifies the type of distributions, and the remaining give the parameters of the distribution. The avaliable distributions and their parameters are listed in §2.1.18. A text file as described in `SetEleDist` can also be used to specify the distribution.

`SetRRRates`(*inv*):

Set the radiative recombination rates. If *inv*=1, the inverse process, photoionization rates are also set.

`SetRateAccuracy`(*r* [, *a*]):

Set the accuracy for the numerical integration in the calculation of rate coefficients. *r* is the relative accuracy, *a* is the absolute accuracy. The default for *r* is 0.01, that for *a* is 10^{-8} in unit of 10^{-10} cm^3 s^{-1}.

`SetTRRates`(*inv*):

Set the radiative transition rates. If *inv*=1, the inverse process, photo-excitation rates are also set.

`SetUTA`(*m* [, *ci*]):

Set the flag for configuration average models. If $m = 1$, all calculations are carried out in the configuration average approximation. The radiative transition rates output contains three additional fields, the transition energy including the UTA shift, the Gausian standard deviation, and the correction to the line strengths due to the configuration interaction within the same non-relativisitc configurations. If $m = 0$, which is the default, the usual detailed term accounting method is used. This function should be called in the beginning of the script, before `Config` function, since it disables the angular momentum coupling performed by `Config`. The optional argument *ci* indicates whether the configuration interaction correction factors should be included when reading the `DB_TR` files. These correction factors are always calculated in `TransitionTable` irrespective of the value of *ci*.

`SpecTable`(*fn* [, *rrc*]):

Output the level populations and line emissivities to the `DB_SP` database file *fn*. If *rrc*=1, the radiative recombination continuum strength should also be included. If *rrc*=−1, only level populations are output.

`TwoPhoton`(*z*, *t*):

Calculate the two-photon decay rate of H-like and He-like transitions $2s_{1/2} \rightarrow 1s_{1/2}$ and $1s2sS_0 \rightarrow 1s^2S_0$ for nuclear charge *z*. $t = 0$ is for H-like, and 1 is for He-like.

3.3 `pol`–Line Polarizations

This module is used to calculate line polarizations due to directional electrons. It takes into account the radiative cascades effects if the atomic data provided include the necessary transitions.

3.3.1 Functions

`CloseSPOL`():

Close the file containing the SFAC input file converted from the current Python script. This function must be called after `ConvertToSPOL`. Only the statements between the call to `ConvertToSPOL` and `CloseSPOL` are converted to SFAC input file. This routine is only available in PFAC interface.

`ConvertToSPOL`():

Converts the statements between this call and the `CloseSPOL` call to SFAC input file *fn*. The resulting file can then be run using the `spol` executable. This routine is only available in PFAC interface.

`Orientation`([*etrans* [,*fn*]]):

Calculate the orientation parameters with optional transverse energy component *etrans*. This routine must be called after `PopulationTable` and before `PolarizationTable`. The orientation parameters are output in file *fn* if it is given.

`PolarizationTable`(*fn* [, *ifn*]):

Calcuates the line polarizations, and output the results to file *fn* in a simple ASCII format. It contains 8 columns, which are number of electrons in the ion, upper level index, lower level index, multipole type (as in the output of `TransitionTable`), transition energy in eV, total line emissivity, emission anisotropy factor at 90°, and linear polarization, respectively. *ifn* is an optional file name which specifies the transitions whose polarizations are to be calculated. It should contain 4 columns, number of electrons, upper level, lower level, and multipole type, respectively. Each of the columns may be in the form "`*`", which matches any transitions. The first 3 columns may also specify a range in the form, e.g., "0-5", which matches any number between 0 and 5 inclusive. Either limit of the range may also be "`*`", so that that limit is not enforced. The second argument may also be given as a list of strings instead of a file name, which correspond to rows of the equivelent file.

`PopulationTable`(*fn*):

Calculate the magnetic sublevel populations, and output the results to file *fn* in a simple ASCII format. For each level, it tabulates the total population, the orientation parameters B_λ, and magnetic sublevel fractions.

`Print`(*args*):

Print out the string representation of *args*. This function exists to asist the conversion to SFAC interface, since Python's `print` statement is not converted.

`SetDensity`(*d*):

Set the electron density in 10^{10} cm^{-3} for collsional radiative model.

`SetEnergy`(*e* [, *s*]):

Set the electron energy *e* in eV. If the optional *s* is positive, the energy distribution is a Gaussian with standard deviation $\sigma = s$ in eV, and mean *e*.

`SetIDR`(*idr* [, *p*]):

Set if the DR satellites polarization to be calculated for a specific target level *idr*. The routine must be called after `SetMLevels`. If *idr* is set to be a valid level, then only DR onto this target will be allowed. The optional *p* specifies the sublevel population of *idr*. *p* must be a List or Tuple with number of elements equal to the number of sublevels of level *idr*. In counting number of sublevels, $\pm M$ are treated as a single sublevel. When *p* is not given, sublevels are assumed to be equally populated.

`SetMIteration`(*a* [, *m*]):

Set the accuracy *a* and maximum iteration *m* for iterative solution of the level population.

`SetMaxLevels`(*m*):

Set the maximum number of levels to be retained in the rate matrix. Levels higher than that are treated iteratively. 0 means retain all levels.

`SetMAIRates`(*fn*):

Set the magnetic sublevel autoionization rates and dielectronic capture rates by reading data from binary file *fn*

`SetMCERates`(*fn*):

Setup the magnetic sublevel excitation rates. Excitation cross sections are interpolated from the binary data file *fn*.

`SetMLevels`(*efn, tfn*):

Setup the magnetic sublevel table and the radiative transitions rates between them. *efn* is the binary data file for energy levels, and *tfn* is that of transition rates.

3.4 `util`–Utility Functions

This module contains some unitity functions which may be useful in various situations.

3.4.1 Functions

`Spline`(*x, y* [, *dy1, dy2*]):

Prepare cubic spline table. *x* and *y* are the independent and dependent variables to be interpolated. *dy1* and *dy2* specifies the optional boundry conditions at the two ends in the form of first derivatives. The default is to use natural cubic spline. It returns a list of second derivatives to be used in `Splint`.

`Splint`(*x, y, y2, x0*):

Calculate the interpolated value at a single point *x0*. *y2* is a list of second derivatives returned by `Spline` using the same *x* and *y*. This routine and `Spline` are adapted from Numerical Recipe.

`UVIP3P`(*x, y, x0* [,*n*]):

Local piece-wise 3rd polynomial interpolation from *x*, *y* to a list of independent values in the list *x0*. If *n* is present, an *n*-th order polynomial interpolation function will be used instead of 3rd order. It returns a list of the results. The Akima interpolation method is used.

3.5 `config`–Electronic Configuration Specification

This module is written in pure Python. It is deprecated now. It includes three main functions. `closed`, `config`, and `avgconfig`. They have the same calling syntax as `Closed`, `Config`, and `AvgConfig` in the module `fac`. One should always use these later ones instead of the counterparts in the `config` module.

3.6 `const`–Physical Constants

This module defines some useful physical constants. They are listed below:

```
Hartree_eV = 27.2113962            # Hartree in eV
Rate_AU = 4.13413733E16            # Atomic Rate Unit in s-1
Rate_AU10 = 4.13413733E06          # Atomic Rate Unit in 10^10 s-1
Rate_AU12 = 4.13413733E04          # Atomic Rate Unit in 10^12 s-1
Area_AU20 = 2.80028560859E3        # Atomic Area Unit in 10^{-20} cm2
Alpha = 7.29735308E-3              # Fine Structure Constant
Ryd_eV = 13.6056981                # Rydberg in eV
RBohr = 0.529177249                # Bohr radius in A
FWHM = 2.35482005                  # conversion from sigma to FWHM
hc = 1.239842E4                    # hc in eV*A
hbc = 1.97327053E3                 # h_bar*c in eV*A
Me_eV = 5.1099906E5                # electron mass in eV
Me_keV = 5.1099906E2               # electron mass in keV
Mp_MeV = 9.38271998E2              # proton mass in MeV
Mp_keV = 9.38271998E5              # proton mass in keV
c  = 2.99792458E10                 # speed of light in cm/s
c10 = 2.99792458                   # speed of light in 10^10 cm/s
e = 1.60217733E-19                 # electron charge in Coulomb
e19 = 1.60217733                   # electron charge in 10^-19 Coulomb
erg_eV = 6.241506363E-13           # erg in eV
re = 2.81794092                    # electron classical radius in fm
sig_t = 6.6524616                  # Thompson cross section in 10^{-25} cm2
kb = 8.617385E-5                   # Boltzman constant in ev/K
```

3.7 `table`–Text Tabulation

3.7.1 Format of Text Table

The `TABLE` class implemented in this module creates and manipulates text tables in a format similar to those of machine-readable tables in AAS journals. It is independent of FAC, and may be used elsewhere. A table contains a header providing the byte-by-byte description of the columns and other explanatory materials. The body of the table are arranged in the conventional multi-column format. The following list shows an example created by `TABLE`:

```
Title:   Total Ionization and Recombination Rate Coefficients
Authors: M. F. Gu
========================================================================
byte-by-byte description of file: trates.tbl
------------------------------------------------------------------------
     Bytes Format                Units      Label Explanation
------------------------------------------------------------------------
    1-   2     I2                 None       NELE Num. of Electrons
    4-   7   F4.2                  [K]       Temp Temperature
    9-  16   E8.2       10^-10^cm^3^/s         DR Total DR rate coefficients
   18-  25   E8.2       10^-10^cm^3^/s      DR_AR Total DR Arnaud & Raymond
   27-  34   E8.2       10^-10^cm^3^/s         RR Total RR rate coefficients
   36-  43   E8.2       10^-10^cm^3^/s      RR_AR Total RR Arnaud & Raymond
   45-  52   E8.2       10^-10^cm^3^/s         CI Total DCI rate coefficients
   54-  61   E8.2       10^-10^cm^3^/s     DCI_AR Total DCI Arnaud & Raymond
   63-  70   E8.2       10^-10^cm^3^/s         EA Total EA rate coefficients
   72-  79   E8.2       10^-10^cm^3^/s      EA_AR Total EA Arnaud & Raymond
------------------------------------------------------------------------
 2 6.80 3.77E-05 3.77E-05 5.94E-02 5.55E-02 0.00E+00 1.44E-09 0.00E+00 0.00E+00
 2 6.95 3.10E-04 3.05E-04 4.45E-02 4.14E-02 0.00E+00 1.93E-07 0.00E+00 0.00E+00
 2 7.10 1.25E-03 1.21E-03 3.31E-02 3.06E-02 0.00E+00 6.32E-06 0.00E+00 0.00E+00
 2 7.25 3.00E-03 2.88E-03 2.44E-02 2.26E-02 0.00E+00 7.80E-05 0.00E+00 0.00E+00
 2 7.40 4.89E-03 4.66E-03 1.78E-02 1.65E-02 0.00E+00 4.79E-04 0.00E+00 0.00E+00
 2 7.55 6.03E-03 5.69E-03 1.28E-02 1.20E-02 0.00E+00 1.78E-03 0.00E+00 0.00E+00
 2 7.70 6.05E-03 5.66E-03 9.11E-03 8.71E-03 0.00E+00 4.62E-03 0.00E+00 0.00E+00
 2 7.85 5.23E-03 4.86E-03 6.42E-03 6.27E-03 0.00E+00 9.01E-03 0.00E+00 0.00E+00
 2 8.00 4.07E-03 3.75E-03 4.47E-03 4.49E-03 0.00E+00 1.49E-02 0.00E+00 0.00E+00
 3 6.80 1.08E-01 1.25E-01 5.49E-02 4.98E-02 2.11E-03 3.21E-03 1.84E-07 2.42E-07
 3 6.95 8.63E-02 1.01E-01 4.10E-02 3.70E-02 7.32E-03 1.06E-02 5.61E-06 7.08E-06
 3 7.10 6.61E-02 7.70E-02 3.03E-02 2.73E-02 1.82E-02 2.50E-02 6.07E-05 7.43E-05
 3 7.25 4.91E-02 5.51E-02 2.21E-02 2.00E-02 3.57E-02 4.57E-02 3.14E-04 3.79E-04
 3 7.40 3.58E-02 3.76E-02 1.59E-02 1.46E-02 5.89E-02 7.14E-02 9.65E-04 1.16E-03
 3 7.55 2.58E-02 2.47E-02 1.14E-02 1.06E-02 8.55E-02 9.81E-02 2.05E-03 2.52E-03
 3 7.70 1.83E-02 1.58E-02 8.00E-03 7.62E-03 1.13E-01 1.23E-01 3.37E-03 4.27E-03
 3 7.85 1.26E-02 9.91E-03 5.56E-03 5.46E-03 1.39E-01 1.45E-01 4.63E-03 6.12E-03
 3 8.00 8.44E-03 6.12E-03 3.82E-03 3.88E-03 1.61E-01 1.62E-01 5.64E-03 7.81E-03
 4 6.80 1.86E-01 1.79E-01 4.71E-02 4.42E-02 5.44E-03 8.22E-03 3.89E-07 0.00E+00
 4 6.95 1.46E-01 1.40E-01 3.51E-02 3.27E-02 1.79E-02 2.58E-02 1.17E-05 0.00E+00
 4 7.10 1.09E-01 1.03E-01 2.58E-02 2.40E-02 4.31E-02 5.87E-02 1.25E-04 0.00E+00
 4 7.25 7.89E-02 7.20E-02 1.88E-02 1.75E-02 8.24E-02 1.04E-01 6.36E-04 0.00E+00
 4 7.40 5.52E-02 4.84E-02 1.35E-02 1.27E-02 1.33E-01 1.59E-01 1.92E-03 0.00E+00
 4 7.55 3.79E-02 3.15E-02 9.54E-03 9.13E-03 1.90E-01 2.15E-01 4.02E-03 0.00E+00
 4 7.70 2.55E-02 2.00E-02 6.67E-03 6.53E-03 2.46E-01 2.64E-01 6.50E-03 0.00E+00
 4 7.85 1.68E-02 1.24E-02 4.59E-03 4.65E-03 2.97E-01 3.03E-01 8.82E-03 0.00E+00
 4 8.00 1.09E-02 7.65E-03 3.12E-03 3.28E-03 3.38E-01 3.31E-01 1.07E-02 0.00E+00
```

3.7.2 Class Attributes and Methods

The `TABLE` class has a number of attributes, which may be set during or after the initialization, and a few method functions for the creation and manipulation of the table:

`fname`:
The file name of the text table. It may be set through the keyword argument *fname* when the `TABLE` instance is created.

`title`:
The title of the table. Set through the keyword *title*.

`authors`:
A list of string for the authors of the table. Set through the keyword *authors*.

`date`:
A string for the date when the table is created. During creation, the date returned by the Python function `time.localtime()` is used. It may be set to other values though the keyword *date*.

`separator0`:
The string separates the title and authors information from the byte-by-byte description. May be set through the keyword *separator0*. The default is `'='*72`.

`separator`:
The string separates the header and the body. Set through the keyword *separator*. The default is `'-'*72`.

`add_column`(***c*):

Add a column to the table. The variable length keyword arguments specify the column attributes. It must contain *label* and *format*. A label is a short identifier of the column. The format is a string starts with `A`, `I`, `F`, or `E` for characters, integers, decimal floating points, and exponential floating points, and followed by a width and possible precision specfication. e.g. `'A10'`, `'F10.3'`, and `'E11.4'`. Other possible keywords are *unit*, *description* and *note*. *unit* specifies the unit of the column (use the string `'None'` for dimensionless quantities). *description* is a short description of the column. *note* are some lengthy explanation for the column which cannot be fit in the byte-by-byte description section. The notes are arranged in the end of the table header.

`open`(*mode* [, *fname*]):

Open the file associated with the table. *mode* is either `'r'` for read or `'w'` for write. The optional *fname* changes the file name of the table.

`close`():

Closes the file associated with the table.

`write_header`():

Write the table header to the file.

`write_row`(*...*):

Write a row of data to the file. It must contain as many arguments as the columns, and the arguments are arranged in the same order as the columns were added.

`read_header`():

Read the header of the table.

`read_columns`(*index* [, *filter, start, stop*]):

Read columns from the table. *index* is a list of integers for columns to be read counting from 0. *filter* is a Python logical expression to be evaluated for each row. Only rows that pass this expression are read. The k-th column is referred to as `c[k]` in the expression. e.g., *filter*=`'c[0]>0'` only reads the rows that have the first column greater than 0. *start* is the row number where the read should begin, and *stop* is the row number where the read should stop, all counting from 0. If *stop*=−1, read through the end of the table. This function returns a list with each element being the column read.

`convert2tex`(*fn, index* [, *filter, start, stop*]):

Convert the table to LaTeX format and save the results to file *fn*. The remaining arguments have the same meaning as those in `read_columns`. Note that this function only adds the LaTex column delimiter and the line break for the table contents. The results are meant to be cut and pasted to LaTex documents.

`rewind`():

Reset the file position to the start of table body.

3.7.3 Example

The example table in §3.7.1 is generated with the following script:

```
from pfac.table import *

####
# create an instance of the class.
####
tbl = TABLE(fname=dfile,
    title='Total Ionization and Recombination Rate Coefficients',
            authors=['M. F. Gu'])
####
```

```
# add each column to the table.
####
d = 'Num. of Electrons'
tbl.add_column(label='NELE', unit='None',
               description=d, format='I2')
d = 'Temperature'
tbl.add_column(label='Temp', unit='[K]',
               description=d, format='F4.2')
d = 'Total DR rate coefficients'
tbl.add_column(label='DR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total DR Arnaud & Raymond'
tbl.add_column(label='DR_AR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total RR rate coefficients'
tbl.add_column(label='RR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total RR Arnaud & Raymond'
tbl.add_column(label='RR_AR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total DCI rate coefficients'
tbl.add_column(label='CI', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total DCI Arnaud & Raymond'
tbl.add_column(label='DCI_AR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total EA rate coefficients'
tbl.add_column(label='EA', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
d = 'Total EA Arnaud & Raymond'
tbl.add_column(label='EA_AR', unit='10^-10^cm^3^/s',
               description=d, format='E8.2')
####
# open the file to write.
####
tbl.open('w')

####
# write the table header
####
tbl.write_header()

####
# write each row of data here
# ......
####

tbl.close()
```

3.8 `atom`–Application of `fac` Module

This is a quite extensive application of `fac` module to the calculation of atomic processes for K-shell and L-shell ions. It implements a class `ATOM` that may be customized through its attributes. The details of this class is not to be documented, since this a highly specialized case. The users are encouraged to write their own scripts to deal with specific problems. Hower, one function from this module is described here which may come to be handy to obtain atomic data for K-shell and L-shell ions.

`atomic_data`(*nele, asym* [*, iprint, dir,* ***kw*]):

This function calculates all collsional and radiative atomic data for K-shell and L-shell ions. The atomic processes includes radiative transition, collisional excitation, photoionization, radiative recombination, autoionization, and collisional ionization. *nele* is a list of integers specify the number of electrons for each ion to be calculated. *asym* is the elemental symbol of the atom to be calculated. *iprint* specifies if the binary data files need to be converted to ASCII format. If it is -1, no conversion is carried out, if it is 0, the conversion is in simple format, if it is 1, the conversion is in verbose format. The default is 1. *dir* is an existing directory where the data files to be stored. Each ion will have a directory named `asym##` under this directory, where `##` is a 2-digit number equal to the number of electrons the ion has. The remaining keywords are used to specify the processes to be excluded in the calculation and other parameters customizing the `ATOM` class. Here we only mention 5 of them, `no_ce`, `no_tr`, `no_rr`, `no_ci`, and `no_ai`. If any of these keywords is set to 1, the corresponding processes will be ignored in the calculation. By default, none of them are set. Note that in typical situations, the computation of collsional excitation and autoionization takes most of the time. If these data are not needed, `no_ce` and `no_ai` should be set to 1.

3.9 `spm`–Application of `crm` Module

This is an application of `crm` module for the spectral modeling. Most of the functions in this module are highly specialized, therefore not documented here. One of them, `spectrum`, is relatively more general, and with minor modifications may be used in various situations to construct simple spectral models for single temperature, optically thin plasmas. The inclusion of this module in the distribution is mainly for demonstration purpose. Anyone interested has to read the source code directly.

4 Frequently Asked Questions (FAQ) To FAC

4.1 General

Q1: Where can I obtain FAC?
FAC is free software. It can be used, modified and redistributed without restriction. Currently, it can be obtained from anonymous **ftp://space.mit.edu/pub/mfgu/fac/**, or one may request to **mfgu@space.mit.edu** for a copy through email.

Q2: What operating systems does FAC run on?
FAC is written in a mixture of ANSI C, Fortran 77, and Python, all of them are in principle platform independent. However, the mixed language programming and the dynamically loadable Python modules makes it more easyly installed in modern UNIX-like systems than others. So far, it has been tested to work under solaris, linux, Mac OS X, and windows with the UNIX API emulation provided by Cygwin.

Q3: How does FAC differ from other atomic codes?
The theoretical methods used in FAC are similar to some other distorted-wave atomic codes, such as HULLAC and differ from more elaborate programs based on close-coupling approximations, such as the Belfast R-Matrix code. The biggest advatage of FAC is its ease of use and its scriptability.

Q4: How does FAC differ from plasma synthetic codes?
There are various plasma codes widely used in X-ray astronomy, such as APEC, MEKAL, SPEX, XSTAR and Cloudy. These codes use the existing atomic data to construct spectral models under different physical conditions. FAC is an atomic code, whose primarily purpose is to generate atomic data, which can be used in these plasma codes. However, FAC includes a collisional radiative model that is able to compute spectral models for optically thin plasmas at a given electron temperature and density. Non-Maxwellian distribution of electron energy may be easily implemented as well. A power law ionizing continuum radiation may also be included.

Q5: Can FAC be incorporated into XSPEC or other spectral analysis programs?
In principle, the collisional radiative model comes with FAC can be used in XSPEC or similar spectral analysis programs. However, this is not practicle, as the model inlcudes a large number of atomic states, especially, the doubly excited states to treat the resonant processes, and is therefore very time consumming. The best strategy of incorporating the FAC results to external spectral models is to extract basic atomic or plasma parameters and use them to build table models or implement dedicated subroutines.

Q6: Which atomic processes can or cannot be calculated with FAC?
FAC can calculate energy levels, radiative transition rates of arbitrary multipole type, collisional excitation and ionization cross sections by electron impact, photionization and radiative recombination cross sections and autoionization rates. In the currect form, FAC does not treat two-photon decay, although such decay rates of $2sS_{1/2}$ state of H-like ions and $1s2sS_0$ state of He-like ions are included in the collisional radiative model using interpolation formulae taken from literature. The three-body recombination are not implemented as well.

Q7: What are the typical accuracies of the atomic parameters caculated with FAC?
The ions other than H-like, the accuracy of energy levels are usually a few eV, which translates to 10–30 mÅ for the wavelength at ∼10Å. For radiative transition rates and cross sections, the accuracies are ∼10–20%. Data for near-neutral ions or atoms may have even larger errors.

Q8: Can FAC be used to calculate atomic parameters for non-X-ray (UV, optical, etc.) lines?
For multiply-charged ions, non-X-ray lines usually result from the transitions within the same complex, which usually have large relative uncertainties in the calculated wavelengths and transition rates. The accuracy for UV and optical lines from near-neutral ions is also very limited.

Q9: Are there standard references to FAC?
Currently, no papers have been published describing the code, though drafts have been written, which are included in the FAC distribution. I have been trying to find a suitable journal willing to publish them. I originally submitted them to Computer Physics Communications (CPC). However, the referee and editor complained about the lack of documentation and code comments. The documentation has improved since then, the code commenting still needs extensive work, which may not be done soon. The first paper that used results of FAC is Gu, M.F., 2003, ApJ, 582, 1241, which can be used as the reference.

Q10: How do I report bugs, make suggestions and get updated about new versions?
Please contact the author at **mfgu@space.mit.edu** for bugs and suggestions. I maintain a small email address list of people who expressed interest in FAC and send release anouncements to them. Let me know if you want to be added to this list. If the list ever grows to the point when I can no longer put it under my personal address book, we may have to create a dedicated mailing list.

4.2 Atomic Structure

Q1: How do I specify a bare ion?
The bare ion is indicated by a call to `fac.Config('', group='b')`, i.e., the first argument of `Config` is an empty string.

Q2: Which configurations should be used as basis for the mean cofiguration which optimizes the radial potential?
The function `fac.OptimizeRadial` accepts a list of configurations, which form the basis for the construction of the mean configuration. Usually, only the lowest lying configurations should be used in `fac.OptimizeRadial` for the construction of mean configuration. I have found that using configurations corresponding to the ground complex is always a good idea. Sometimes, it maybe worthwile to inlcude the first excited configurations. However, it is always a bad practice to inlcude very highly excited configurations, especially those inner shell excited ones.

Q3: Can I use a specific mean configuration to be used in potential optimization?
The function `fac.AvgConfig` may be used to set the mean configuration for the potential optimization. In this case, the function `fac.OptimizeRadial` must be called with no arguments.

Q4: How do I know what mean configuration is used in the potential optimization, If one is not given specificly by `AvgConfig`?
The function `fac.GetPotential` may be called after `OptimizeRadial` to obtain the mean configuration used, and the resulting radial potential.

Q5: When calculating ionization or recombination processes, should I use the mean configuration for recombined (ionizing) or recombining (ionized) ion?
Usually, it does not matter for highly charged ions, and I usually use the recombined ion. The difference of one more or less electron screening the nuclear charge may be substaintial for low-Z elements, in which case, one may have to make a decision by comparing the results to experimental values or other theoretical works.

Q6: How do I determine which configurations should be interacting?
This depends on the computer resource available, and the desired accuracy. The dimension of the Hamiltonian matrix increases very rapidly as the number of interacting configurations grows. The convergence with respect to the configuration interaction is usually slow. For applications which FAC is primirily designed for, one typically inlcudes only configurations within the same complex, except for some low-lying configurations.

Q7: What relativistic effects are included?
The standard Dirac-Coulomb Hamiltonian is used in FAC, which means that the spin-orbit interaction, mass-effect and other leading relativistic effects are fully treated. However, higher-order QED effects, such as retardation and recoil are only included in the Breit interaction with zero energy limit for the exchanged photon. Vacuum polarization and self-energy corrections are treated in the screened hydrogenic approximation.

Q8: Why the transition rate between two specific states is not in the output file, although it should have been calculated?
Not every calculated transition rate are output. Some weak transitions are discarded to avoid very large files. A small number, which may be set by the function `fac.SetTransitionCut`, controlls this behavior. If the transition rate of 2 → 1 devided by the total decay rate of state 2 is less than this number, this rate is not output. The default for this number is 10^{-4}.

4.3 Collisional Excitation

Q1: Why is there multiple data blocks in the output corresponding to a single call of `CETable` sometimes?
Sometimes, the transition array corresponding to a given call to `CETable` include transitions with a wide range of excitation energies. This typically happens for transitions within a single complex or transitions between more than 2 complexes are mixed together in one call of `CETable` (which should generally be avoided). Since the excitation radial integrals are only calculated on a few-point transition energy grid, it is undesirable to have a very wide range in the actual transition energies. `CETable` avoid this by subdivide the transitions in groups. Within each group, the transition energies does not vary by more than a factor of 5. A different transition energy grid and collision energy grid are used for different groups, and therefore, corresponding to different data blocks in the output.

Q2: Why is the default `QKMODE` for excitation is `EXACT`, not `FIT`?
It used to be in `FIT` mode. However, the fitting formulae sometimes fail to reproduce the calculated collision strengths. After playing with different fitting formulae for a while, I decided that the user should do the fitting (if one is desired) on the case by case basis. The function `SetCEQkMode` may be used to specify a different `QKMODE`.

Q3: Can I use a different collision energy grid?
A collision energy grid in terms of the energy of the scattered electron is automatically constructed if one is not specified prior to calling `CETable`. One may use the function `SetCEGrid` to specify a different grid. Or one may use `SetUsrCEGrid` to have a user grid different from the grid on which the collision strengths are calculated, and use `INTERPOLATE` mode for the `QKMODE`.

Q4: The collision strengths at very high energies are incorrect?
Due to the limited radial grid size, the collision strengths at energies much higher than the excitation energy (more than a few hundred times higher) are unlikely to be reliabe. However, the high energy collision strengths should not be calculated directly. The Bethe and Born limit parameters in the output should be used to obtain them.

4.4 Photoionization and Radiative Recombination

Q1: When can I use the function `RecStates` to construct the recombined states instead of specifying their configurations by `Config`?
The function `RecStates` can only be used if the free electron is captured to an empty orbital.

Q2: Why is the bound-free oscillator strength differ from some other theoretical calculation by a constant factor?
The bound-free differential oscillator strengths calculated by FAC have units of Hartree^{-1}. Also the values depend on the normalization of continuum orbitals. It is therefore possible that they differ from other theorectial calculations by a constant factor. One should always use the formula in this manual or the accompanying papers to convert them to photoionization or radiative recombination cross sections.

Q3: Can I use only the fitting formula and ignore the tabulated gf values?
The fitting formula and the parameters given for the bound-free oscillator strengths is only valid at high energies (beyond the largest energy of the photo-electron energy grid). This means that the gf values at energies within the photo-electron energy grid should be calculated by interpolation instead of the fitting formula. However, this is often only necessary for the ionization of valence shells of near neutral ions and atoms, since only in these cases, the near threashold behavior of the gf values differ from the fitting formula significantly.

4.5 Autoionization and Dielectronic Recombination

Q1: What does the channel number in the call to `AITable` mean?
It does not mean anything. It is simply an identifier which may be useful sometimes to tag a certain autoionization channel.

Q2: How do I improve the resonance energies of low-lying $\Delta n = 0$ resonances?
The energyies of low-lying $\Delta n = 0$ resonances can not the calculated accurately. This greatly reduces the reliability of the resulting dielectronic recombination rates. In FAC, there is a method to improve the accuracy of these energies by adjusting the core transition energies according to the experimental values. The function `CorrectEnergy` is used to modify the calculated energy levels by specified amount.

Q3: Why is it advised to make seperate calls to `AITable` for different bound or free state complexes?
The autoionization radial integrals are only calculated for a few free electron energies. The actual values are interpolated from them. This only works well if the the free electron energy after autoionization does not vary widely. Therefore, one should avoid calling `AITable` with bound or free states in different complexes. e.g., KLL and KLM resonances should be calculated with two seperate calls to `AITable`.

4.6 Collisional Ionization

Q1: Why are FAC ionization cross sections calculated with `BED` mode usually much smaller at near threshold energies as compared with distorted-wave calculations?
When using `BED` mode to calculate the ionization radial integrals, the total ionization cross sections are scaled by a factor $E/(E+I)$, where E is the energy of incident electron, and I is the ionization threshold energy. The result of this scalling is usually desirable as distorted-wave method overestimates the near threshold cross sections.

Q2: Why is the `DW` mode so slow as compared with `BED` and `CB` modes?
In the `DW` mode, the ionization radial integrals are calculated by summing up partial wave contributions. This is a time consuming process as now there are two continuum electrons involved. In the `CB` mode, the radial integrals are simply looked up in a table, is therefore the fastest method. In the `BED` mode, the radial integrals are calculated using the bound-free differential oscillator strengths, which can be computed much faster than the `DW` ionization radial integrals.

4.7 Collisional Radiative Model

Q1: What kinds of electron energy distributions are built in the `crm` module?
Currently, the Maxwellian and Gaussian energy distributions are supported for electrons. The Gaussian distribution is meant to simulate a monoenergetic electron beam with finite energy width.

Q2: Are photoionization and photo-excitation supported?
A power law ionization continuum can be used for the photoionization and photo-excitation sources. However, the radiative transfer effects are not implemented.

Q3: How can I add more electron energy and/or photon energy distributions?
Take a look at the C header and source files `faclib/rates.h` and `faclib/rates.c`, and follow the implementation of built-in distributions. The FAC package must be recompiled and relinked for added distributions to work.