1. The “f17” file format for data exchange

Beginning from year 2017, the EOS and opacity tables in the RALEF code are generated in the “f17” format described below. This format has been developed with the primary goal of providing a possibility to read and extract various numerical data from a file with a minimum amount of advance knowledge about the format, used for writing these data, combined with the possibility to add at any time any amount of new data of any type, supported by the Fortran-90 language.

1. Any file `sample.f17` in the “f17” format is supposed to be of the *unformatted* form in all the I/O statements, i.e. created by the statements of the form

   ```
   write(unit) a,b,...
   ```

2. Any file in the “f17” format is structured as a sequence of an arbitrary number of **items**; every individual **item** consists of two (and only two) Fortran *records* representing its two elements, namely, its **tag** and **entity**:

   item # 1 ::= **tag** #1
   entity # 1

   item # 2 ::= **tag** # 2
   entity # 2

   ...

3. The **entity** of an item is either a Fortran *array*, or a Fortran *scalar variable* of stored data, written into the `sample.f17` file as a single Fortran *record*, i.e. by a single use of the `write` statement.

4. The **tag** of an item contains all the information about the Fortran *type* and *shape* of the **entity**, associated with this item. Every **tag** is written into the `sample.f17` file as a single Fortran *record*, i.e. by a single use of the `write` command. Every **tag** has the same fixed structure, namely, it is composed of two `character(32)` variables, followed by eight `integer(4)` numbers.

   Example of code for writing a single **tag**:

   ```
   character(32) :: ent_name,ent_type
   integer(4) :: ent_shape(8)
   integer :: luw

   luw=21
   open(luw,file='sample.f17',form='unformatted')
   ent_name='header'
   ent_type='character(128)'
   ent_shape=(/1,10,0,0,0,0,0,0/)
   write(luw) ent_name,ent_type,ent_shape
   ```

In this example, the character variable `ent_name` contains the *name* of the **entity**, represented by the current **tag** (of no more than 32 characters). The character variable `ent_type` contains the Fortran *type* of the **entity** (like `integer(4)`, `real(8)`, `complex(16)`, etc.; 32 characters are sufficient to describe any intrinsic type in the Fortran 90).

Integer `ent_shape(1)` is equal to the *rank* of the **entity**, represented by the current **tag**. Possible values are `ent_shape(1) = 0, 1, ..., 7`. The value `ent_shape(1) = 0` means that the **entity** is a *scalar variable*; the value `ent_shape(1) = 1` means that the **entity** is a *rank-1 array* with one dimension; `ent_shape(1) = 2` means that the **entity** is a *rank-2 array* with two dimensions; etc.

Integer `ent_shape(2)` is the *extent* of the **entity** array along dimension 1; integer `ent_shape(3)` is the *extent* of the **entity** array along dimension 2; etc.

In the above example, the **tag** represents an **entity** named `header`, which is a one-dimensional *array* of type `character(128)`, containing 10 *elements*.

5. *Example* of code for writing two **items**, namely, an *integer* array `ia(1:100,1:200)` and a *real(4)* array `pr(1:500,1:400,1:10)`:

```
character(32) ::  ent_nt(2)
integer(4) ::  ent_sh(8)

integer ::  luw,ia(100,200)
real(4) ::  pr(500,400,10)

luw=21
open(luw,file='sample.f17',form='unformatted')

ia= ...
ent_nt(1)='ia'
ent_nt(2)='integer'
ent_sh=(/2,100,200,0,0,0,0,0/)
write(luw) ent_nt,ent_sh
write(luw) ia

pr= ...
ent_nt(1)='pressure'
ent_nt(2)='real(4)'
ent_sh=(/3,500,400,10,0,0,0,0/)
write(luw) ent_nt,ent_sh
write(luw) pr
```

The only *a priori* information that a customer has to know in order to be able to use a given "f17" file, is the format of the **tag** *records*. Once the latter is known, one easily generates a list of all items in the given "f17" file; in the RALEF code this can be done by calling a service routine

```
call FINDALL_F17(lur,luw,ierr) ,
```

where `lur` is the *unit* number of the opened "f17" file to be explored, `luw` is the *unit* number of the opened output file (for the standard output `luw=6`), `ierr` is the `integer(4)` error flag.

Having obtained the full list of **items**, one readily extracts the **entity** of a needed **item** by making use of the `FIND_F17(ent_name,lur,luw,ierr)` service routine as, for example, could be done for the previously written `sample.f17` file:

```
integer :: lur
real(4) :: pr(500,400,10)

lur=21
open(luw,file='sample.f17',form='unformatted')

call FIND_F17('pressure',lur,6,ierr)
read(lur) pr
```

It is recommended that any "f17" file includes a `character(128)`-type **item** (a rank-1 array) with a standard name `header`, where all the explanations are given concerning the physical meaning, units of measurement, etc. for all the **items** stored in a given "f17" file. If the `header` is the first **item** in a given "f17" file, the format of its **tags** can be explained in the first line of the `header`, and then a customer can read it out as

```
integer :: lur
character(1) :: dummy
character(128) :: char

lur=21
open(luw,file='sample.f17',form='unformatted')
read(lur) dummy
read(lur) char
```

(i.e. a customer has only to know that the first item is a `character(128)` variable or array), and then, having learned the format of **tags** from the contents of `char`, to read out the full list of items in the explored file. The command `read(lur) dummy` only serves to pass to the second record, which is the **entity** of the `header` **item**.

To minimize the probability of errors by reading out **entities** from the "f17" files, one can add a special **control character** (or a **control word**) to the end of each **entity** *record*, like, for example,

```
character(32) :: ent_nt(2)
integer(4) :: ent_sh(8)

integer :: luw
real(4) :: pr(500,400,10)

character, parameter :: ccr='|'
luw=21
open(luw,file='sample.f17',form='unformatted')

pr= ...
ent_nt(1)='pressure'
ent_nt(2)='real(4)'
ent_sh=(/3,500,400,10,0,0,0,0/)
write(luw) ent_nt,ent_sh
write(luw) pr,ccr
```

This allows to check directly whether the corresponding `read` statement has attempted to read out less data (by an attempt to read more data, an error message is generated anyway) than was originally written into the corresponding **entity**. The actual value of the **control character** (**control word**) is also recommended to be included into the first line of the `header`.