CPU time

Since Fortran 95 standard programmers are equipped with subroutines for measuring the CPU and wall time. Both subroutines may be used to time the execution of the program and report the used time.

CPU time may be measured with subroutine `cpu_time()`. The subroutine takes one `real` argument and returns the elapsed time in seconds. The difference of two values measured at two different points of execution will give a CPU time used the given section of code. In general the `cpu_time()` subroutine usage may look like this

```
real :: start_time, end_time

call cpu_time(start_time)
...
call cpu_time(end_time)
write(output,'(a,f20.3,a)') 'Total CPU  time : ',end_time-start_time,' s'
```

An complete example looks like this

```
program cpu_time_example

  use iso_fortran_env

  implicit none
  real :: start_time, end_time
  integer :: i, i_max, val

  i_max = 1000000000

  call cpu_time(start_time)

  do i=1, i_max
    val = i**2
  end do

  call cpu_time(end_time)
  write(output_unit,'(a,f20.3,a)') 'Total CPU  time :  ',end_time-start_time,'
s'

end program cpu_time_example
```

which gives an output

```
Total CPU  time :                2.726 s
```

Wall time

In analogy to `cpu_time()` subroutine we have `system_clock()` subroutine for accessing the wall time. The usage looks very similar to the `cpu_time()` subroutine, with this difference that the variables are now of `integer` type. Every call returns a count of processor clock since undefined time in the past. `count_rate` determines number of clock tics per second. An example usage

```
integer :: system_time_start, system_time_end
integer :: count_rate

call system_clock(system_time_start, count_rate)
...
call system_clock(system_time_end)
write(output,'(a,f20.3,a)') 'Total Wall time : ',&
   dble(system_time_end-system_time_start)/dble(count_rate),' s'
```

A full program in analogy to the CPU time (but this time measuring the Wall time) may look like this

```
program wall_time

  use iso_fortran_env

  implicit none
  integer :: start_time, end_time, count_rate
  integer :: i, i_max, val

  i_max = 1000000000

  call system_clock(start_time, count_rate)
  do i=1, i_max
    val = i**2
  end do

  call system_clock(end_time)
  write(output_unit,'(a,f20.3,a)') 'Total Wall time :  ',dble(end_time-
start_time)/dble(count_rate),' s'

end program wall_time
```

which gives

```
Total Wall time :                2.777
```

Date and time

Fortran 95 standard brings also a time and date subroutine `date_and_time` . The most general call for the subroutine looks like

```
call data_and_time(values=variable)
```

where `variable` is an `integer` vector of lenght 8. The meaning of particular entries of the `variable` output is summarized in the table below

Element	Description
`value(1)`	Year
`value(2)`	Month
`value(3)`	Day
`value(4)`	Time diff. with UTC [minutes]
`value(5)`	Hour
`value(6)`	Minutes
`value(7)`	Seconds
`value(8)`	Miliseconds

The subroutine `date_and_time` may be used to indicate the date and time when the program was started of when it has finished, for example

```
integer, dimension(8) :: time_values

call date_and_time(VALUES=time_values)
write(output_unit,'(2(a,i2.2),a,i4.4,3(a,i2.2))') &
 'Job finished   : Date:
',time_values(3),'/',time_values(2),'/',time_values(1), &
               '   Time: ',time_values(5),':',time_values(6),':',time_values(7)
```

A whole example program may look like this

```
program date_time

  use iso_fortran_env

  integer, dimension(8) :: time_values

  call date_and_time(VALUES=time_values)

  write(output_unit, '(a,3x,i4)') '        Year : ', time_values(1)
  write(output_unit, '(a,3x,i4)') '       Month : ', time_values(2)
  write(output_unit, '(a,3x,i4)') '         Day : ', time_values(3)
  write(output_unit, '(a,3x,i4,3x,a,3x,i4,a)') &
                                  '    UTC diff : ', time_values(4), '-> ',
time_values(4)/60,' hours '
  write(output_unit, '(a,3x,i4)') '        Hour : ', time_values(5)
  write(output_unit, '(a,3x,i4)') '     Minutes : ', time_values(6)
  write(output_unit, '(a,3x,i4)') '     Seconds : ', time_values(7)
  write(output_unit, '(a,3x,i4)') ' Miliseconds : ', time_values(8)


  write(output_unit, '(2(a,i2.2),a,i4.4,3(a,i2.2))') &
     'Program finished   : Date:
',time_values(3),'/',time_values(2),'/',time_values(1), &
                        '   Time:
',time_values(5),':',time_values(6),':',time_values(7)

end program date_time
```

with the following output

```
        Year :    2015
       Month :       7
         Day :       5
    UTC diff :    -240   ->      -4 hours
        Hour :      23
     Minutes :       1
     Seconds :      46
 Miliseconds :     473
Program finished   : Date: 05/07/2015   Time: 23:01:46
```