Program Initialization and Termination

Aleksandar Donev

February 2000

Contents

1 Module Initialization_Termination .......................... 1
   1.1 Program Initialization .................................... 1

2 Formatting rules for HPF/F90 files ......................... 3
1 Module Initialization_Termination

This module provides routines to initialize and end a main program during a network optimization run. In fact, it is a get-together point for the modules Error_Handling and System_Monitors. Also, the random number generator seeds and possibly some other run-time parameters (such as number of processors on a parallel machine) can be initialized in StartProgram and all arrays deallocated in EndProgram. This module also contains in it the namelist ProgramOptions which sets some of the choice parameters in the modules Error_Handling and System_Monitors.

"WEAVE.f90" 1.1

MODULE Initialization_Termination
  USE Precision    // Kind parameters
  USE Error_Handling
  USE System_Monitors
  USE Random_Numbers, ONLY: UnpredictableSeeds
  IMPLICIT NONE
  PUBLIC :: StartProgram, EndProgram
PRIVATE
  NAMELIST/ProgramOptions/ max_usable_memory, print_file, log_file, append, log_level, print_level,
                          warning_action, non_critical_action
CONTAINS
  (ProgramFlow 1.1.3)
END MODULE Initialization_Termination

1.1 Program Initialization

There are some routines which need to be called to initialize every program which is based on this library, such as random seed initialization, reading some basic input files, etc. Here they are:

This macro will log the current time:

"WEAVE.f90" 1.1.2

@m_RecordTime(message, date_time)
  CALL DATE_AND_TIME(VALUES = date_time)
  WRITE(UNIT = message_log_unit, FMT = "(5(A,I2),A,I4)") message, date_time(5), ":",
         date_time(6), ":", date_time(7), " on ", date_time(3), "/", date_time(2), "/", date_time(1)
And the initialization and termination routines:

\[\text{ProgramFlow 1.1.3}\] =

```fortran
SUBROUTINE StartProgram()  // Start the program
    IMPLICIT NONE
    INTEGER, DIMENSION(8) :: date_time
    OPEN (UNIT = program_options_unit, FILE = TRIM(options_file), STATUS = "OLD",
          ACCESS = "SEQUENTIAL", ACTION = "READ", DELIM = "APOSTROPHE", IOSTAT = error_status)
    // File with all the options
    IF(error_status /= 0) CALL CriticalError(message = " Options file " || TRIM(options_file) || "
    could not be opened", caller = "StartProgram")
    READ(UNIT = program_options_unit, NML = ProgramOptions, IOSTAT = error_status)
    IF(error_status /= 0) CALL Warning(message = "NAMELIST ProgramOptions was not read
    successfully" || " from file " || TRIM(options_file), action = "PAUSE",
    caller = "StartProgram")
    CALL InitializeErrorHandling()  // Error-related I/O
    CALL CreateMonitors()  // System monitors
    CALL UnpredictableSeeds()  // Time-dependent random seeds
    WRITE(UNIT = message_log_unit,
          FMT = "(A)" "-----------------------------------------------"
     _RECORDTIME("Started program at: ", date_time)
    WRITE(UNIT = message_log_unit, NML = ProgramOptions)
END SUBROUTINE StartProgram

SUBROUTINE EndProgram()  // End the program
    IMPLICIT NONE
    INTEGER, DIMENSION(8) :: date_time
    CLOSE(UNIT = program_options_unit)
    CALL DestroyMonitors()
    CALL MemoryUsage()  // Check if there were any obvious memory leaks
    _RECORDTIME("Ended program at: ", date_time)
    WRITE(UNIT = message_log_unit,
          FMT = "(A)" "-----------------------------------------------"
     _CALL TerminateErrorHandling()
END SUBROUTINE EndProgram
```

This code is used in section 1.1.
2  Formatting rules for HPF/F90 files

These are just same auxiliary formatting rules and useful macros I use from time to time.

@m  _GenericInterface(generic_name, ...)
    interface generic_name
    module procedure #.
    end interface generic_name
@m  _DeclareLWord(...)
    integer :: #.
@m  _DeclareLWP(...)
    integer (kind = i_wp) :: #.
@m  _DeclareRWP(...)
    real (kind = r_wp) :: #.
@m  _DeclareRSP(...)
    real (kind = r_sp) :: #.
@m  _DeclareRDF(...)
    real (kind = r_dp) :: #.
@m  _FullExtent(rank) : $do (dim, 2, rank) { , } 
@m  _VarSequence(variable, start, end)
    _variable## start $do (dim, $eval (start + 1), end) { , _variable@dim } 
@m  _NestedLoopStart(variable, array, rank)
    $do (dim, rank, 1, -1) { do _variable@dim = lbound(array, dim), ubound(array, dim) } 
@m  _NestedLoopEnd(rank) $do (dim, 1, rank) { end do } 
@m  _Dummy(...) 
@m  _DisplayArray(message, array)
    if (size(array) ≤ 20) then
        write(message, unit, "(A)") message
        write(message, unit, "(2005.2)") array
    end if