f90_gc: Fortran Garbage Collector Module

July 31, 2012

1 Name

`f90_gc` — Module controlling the runtime garbage collector

2 Usage

```
USE F90_GC
```

3 Synopsis

Generic Procedures

`EXPAND_HEAP`, `GET_FREE_BYTES`, `GET_BYTES_SINCE_GC`, `GET_FULL_GC_FREQUENCY`, `GET_HEAP_SIZE`, `GET_MAX_GC_RETRIES`, `GET_MAX_HEAP_SIZE`, `SET_FULL_GC_FREQUENCY`, `SET_GC_ALLOWED`, `SET_GC_VERBOSITY`, `SET_HEAP_EXPANSION`, `SET_MAX_GC_RETRIES`, `SET_MAX_HEAP_SIZE`.

Specific Procedures

`ENABLE_INCREMENTAL_GC`, `GCOLLECT`, `NCOLLECTIONS`.

4 Procedure Description

In the description of each procedure, an argument whose `KIND` is denoted by '*' can accept any kind of that type. Other `KIND` indications use the named parameters from the `F90_KIND` module; these named parameters are not, however, exported from `F90_GC`.

```
SUBROUTINE ENABLE_INCREMENTAL_GC()
```

Enables incremental garbage collection; once enabled, it cannot be disabled. Full collections will still be performed with a frequency determined by the "Full GC frequency" setting (see `GET_FULL_GC_FREQUENCY` and `SET_FULL_GC_FREQUENCY`.

```
LOGICAL(word) FUNCTION EXPAND_HEAP(N)
INTEGER(*),INTENT(IN) :: N
```

This function attempts to expand the heap by N blocks (of 4K bytes), returning `.TRUE.` if and only if it is successful. This is unaffected by the heap expansion setting (see `GET_HEAP_EXPANSION` and `SET_HEAP_EXPANSION`).

Note that on some systems this may return a false indication of success as the operating system delays the actual allocation of memory until an attempt is made to use it — at which point the program may be aborted. Therefore this function should *not* be used to attempt to allocate all the virtual memory on a system.

```
SUBROUTINE GCOLLECT()
```

Manually initiates a garbage collection; this is a full collection even if incremental collection has been enabled (see `ENABLE_INCREMENTAL_GC`). This is not affected by the "GC allowed" setting (see `GET_GC_ALLOWED` and `SET_GC_ALLOWED`).

```
SUBROUTINE GET_BYTES_SINCE_GC(NBYTES)
INTEGER(int32 or int64),INTENT(OUT) :: NBYTES
```

Returns the number of bytes allocated since the last collection.

```
SUBROUTINE GET_FREE_BYTES(NBYTES)
INTEGER(int32 or int64),INTENT(OUT) :: NBYTES
```

Returns a conservative estimate of the number of free bytes in the heap.

```
SUBROUTINE GET_FULL_GC_FREQUENCY(FREQUENCY)
INTEGER(int16 or int32 or int64),INTENT(OUT) :: FREQUENCY
```

Returns the number of partial collections that are done between each full collection when incremental collection is enabled (see ENABLE_INCREMENTAL_GC). This has no effect on manual collection (see GCOLLECT) or when incremental collection has not been enabled. The default value in Release 5.2 is 4.

```
SUBROUTINE GET_GC_ALLOWED(ALLOWED)
LOGICAL(*),INTENT(OUT) :: ALLOWED
```

Returns the "GC allowed" setting; the default setting is .TRUE.

```
SUBROUTINE GET_GC_VERBOSITY(VERBOSITY)
INTEGER(*),INTENT(OUT) :: VERBOSITY
```

Returns the "GC verbosity" level; this has a range of 0-100, and the default level is zero. Increasing the verbosity level increases the number of informational messages producing during garbage collection. In Release 5.2, the only significant verbosity levels are 0 and 10.

```
SUBROUTINE GET_HEAP_EXPANSION(ALLOWED)
LOGICAL(*),INTENT(OUT) :: ALLOWED
```

Returns the automatic heap expansion setting; the default setting is .TRUE.. Even when automatic heap expansion is disabled, the heap will still be expanded if that is necessary to satisfy an allocation request.

```
SUBROUTINE GET_HEAP_SIZE(NBYTES)
INTEGER(int32 or int64),INTENT(OUT) :: NBYTES
```

Returns the current heap size in bytes, or -1 if the heap size cannot be represented in the integer variable (i.e. if the current heap size is more than 2G bytes and the integer variable is only a 32-bit one).

```
SUBROUTINE GET_MAX_GC_RETRIES(N_RETRIES)
INTEGER(*),INTENT(OUT) :: N_RETRIES
```

Returns the maximum number of garbage collections attempted before reporting failure to the caller; if the allocation was not initiated by an ALLOCATE statement with a STAT= clause, this will result in program termination. The default value in Release 5.2 is zero.

```
SUBROUTINE GET_MAX_HEAP_SIZE(NBYTES)
INTEGER(int32 or int64),INTENT(OUT) :: NBYTES
```

Returns the maximum heap size setting, or zero if no maximum heap size has been set. The default setting is zero.

```
INTEGER(int32) FUNCTION NCOLLECTIONS()
```

Returns the number of garbage collections that have been performed.

```
SUBROUTINE SET_FULL_GC_FREQUENCY(FREQUENCY)
INTEGER(int16 or int32 or int64),INTENT(IN) :: FREQUENCY
```

Sets the number of partial garbage collections to be done between each full collection; this has no effect unless incremental collection has been enabled (see ENABLE_INCREMENTAL_GC). A full collection may still be done if there is a substantial increase in the number of in-use blocks.

```
SUBROUTINE SET_GC_ALLOWED(ALLOWED)
LOGICAL(*),INTENT(IN) :: ALLOWED
```

Controls the "GC allowed" setting; when this setting is .FALSE., automatic garbage collection is inhibited. This does not affect manual garbage collection (see *GCOLLECT*).

```
SUBROUTINE SET_GC_VERBOSITY(VERBOSITY)
INTEGER(*),INTENT(IN) :: VERBOSITY
```

Sets the verbosity level; the default value is zero, and the range is limited to 0-100. In Release 5.2, the only significant levels are 0 and 10.

```
SUBROUTINE SET_HEAP_EXPANSION(ALLOWED)
LOGICAL(*),INTENT(IN) :: ALLOWED
```

Controls whether automatic heap expansion is allowed; if this is .FALSE., automatic heap expansion will only occur if necessary to satisfy an allocation request (normally the heap is expanded when an heuristic determines that it would be advantageous). This does not affect manual heap expansion (see EXPAND_HEAP).

```
SUBROUTINE SET_MAX_GC_RETRIES(N_RETRIES)
INTEGER(*),INTENT(IN) :: N_RETRIES
```

Sets the maximum number of garbage collections attempted before reporting out of memory after heap expansion fails (i.e. is refused by the operating system). The default setting in Release 5.2 is zero.

```
SUBROUTINE SET_MAX_HEAP_SIZE(N)
INTEGER(int32 or int64),INTENT(IN) :: N
```

Sets the maximum size of the heap to N bytes. This will prevent the heap from automatic expansion beyond the specified limit, and prevent it from automatic expansion entirely if it is already beyond the limit.

5 See Also

nag_modules(3), **nagfor**(1).

6 Bugs

Please report any bugs found to 'support@nag.co.uk' or 'support@nag.com', along with any suggestions for improvements.