log

> MPAS message logging manager

Introduction

This module contains the routines for managing the writing of messages
to log files.
The log module operates around a variable named `mpas_log_info` that contains
all of the external information needed to perform logging operations
for the current model configuration.
`mpas_log_info` is a module level pointer to the active core's `mpas_log_type`
instance. This allows the core's domain instance to "own" the log manager,
while the log module has internal access to it. This approach has two
benefits:

1. Calls to `mpas_write_log` do not require any configuration metadata
   (e.g., a unit number) to be passed in, simplifying the call API.

2. Because the log module uses a pointer to a `mpas_log_type` instance,
   different instances can be swapped in and out from higher levels of
   the modeling framework during execution. This is necessary to allow
   multiple different MPAS cores to operate in the same climate model,
   but would also apply to multiple MPAS cores running in a single MPAS
   executable (if this is ever supported), or multiple domain instances
   of the same MPAS core running together. (It is the responsibility of
   the higher-level driver code(s) to manage any swapping of `mpas_log_type`
   instances.)

module variable:

```
type(mpas_log_type), pointer, public :: mpas_log_info => null()
```

public procedures or functions:

1. mpas_log_init
2. mpas_log_open
3. mpas_log_write
4. mpas_log_finalize

public procedures or functions

1 mpas_log_init

> Initializes log manager

details

This routine initializes the log manager for the active core on **each task** by assigning appropriate values to the members of mpas_log_info, based on the model configuration. mpas_log_info is a module level pointer of the active core's mpas_log_type instance. This allows the core domain instance to "own" the log manager, while the log module has internal access to it.

note
isOpen: logical variable belong to type `mpas_log_file_type` indicating if the log file has already been opened.
isActive: logical variable belong to type `mpas_log_file_type` indicating if the log file is **active on this processor**.

1.1 main procedures

1. create log instance

```
allocate(coreLogInfo)
allocate(coreLogInfo % outputLog)
allocate(coreLogInfo % errorLog)
!these variables are all type
```

2. initialize all log info members

```
coreLogInfo % errorLog % isActive = .true. ! error files are always active
```

3. point the module-level log instance to the **initialized** core's instance

```
mpas_log_info => coreLogInfo
!update values based on model conf
mpas_log_info % coreName = domain % core % coreName
maps_log_info % taskID = doamin % dminfo % my_proc_id
mpas_log_info % nTasks = domain % dminfo % nprocs
```

4. setup isActive for output log and error log

```
#ifdef MPAS_DEBUG
      ! in debug mode, all tasks have active logs
      mpas_log_info % outputLog % isActive = .true.
#else
      ! in non-debug mode, only task 0 has active log
      if (mpas_log_info % taskID == 0) then
         mpas_log_info % outputLog % isActive = .true.
      else
         mpas_log_info % outputLog % isActive = .false.
      endif
#endif
```

4. Generate proposed (**default**) file names, `log.atmosphere.0000.out`

details

that we will use if an open file has not already been supplied.
We might need the task number as a string to build filenames below.
Expand zero padding based on number of total tasks.

5. Whether use prescribed unit numbers or default name.

1.2 Conclusions

This subroutine assign individual member info (taskID, filename, nProcs, Active status) to `mpas_log_info` instance

2 mpas_log_open

> Connects to a log file for the log module to use, *simple open a file exist in Linux filesystem*

details

By default, a new file with a new file unit is opened.
Optionally, a fileUnit can be passed in that must already be attached to an appropriate file.

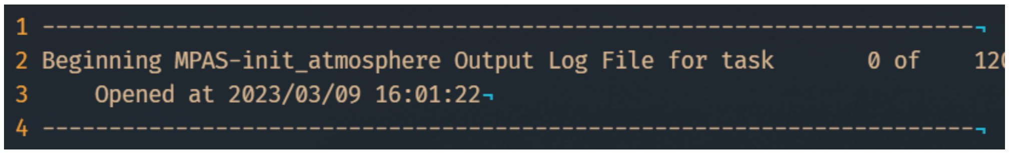
By default, **an output log file is opened**, but **optionally this can be an error file.**
This routine assumes that the log manager has been initialized and all log settings are set correctly, including filename, active status, etc. **Because** it uses the module-level pointer instance of the log manager.

2.1 main procedures

1. Determine if this is an output or error file and get correct object

```
if (present(openErrorFile)) then
   if (openErrorFile) then
      thisIsOutputLog = .false.
   endif
endif
if (thisIsOutputLog) then
   logFileInfo => mpas_log_info % outputLog
else
   logFileInfo => mpas_log_info % errorLog
endif
```

2. Get attributes needed after `mpas_log_init` , `unitNumber, activeFile, alreadyOpen, fileName`
3. Open log file (if active)
   1. Open new log file (`isOpen = .false.`)
   2. If already open, make sure it's valid.
4. Write a header message, and flush immediately.



```
----------------------------------------------------------------------
Beginning MPAS-init_atmosphere Output Log File for task       0 of    12
    Opened at 2023/03/09 16:01:22
----------------------------------------------------------------------
```

2.2 Conclusions

By default, this subroutine open a new log file and connect this to log module for use.

3 mpas_log_write

> write message to the log file

details

This routine writes a message to the log file. The message is the only required argument. A number of optional arguments control additional
behaviors:
messageType: MPAS_LOG_OUT = output (default)
MPAS_LOG_WARN = warning
MPAS_LOG_ERR = error (written to both output and error log files)
MPAS_LOG_CRIT = critical error (written to both output and error log files and kills model)

masterOnly: flag indicating only the master task should write this message, *regardless of if all tasks have open log files.*
flushNow: flag indicating the message should be flushed immediately. Note: error and critical error messages are always flushed immediately.
intArgs, realArgs, logicArgs: arrays of variable values to be inserted into the message to replace the following characters: $i, $r, $l. See routine log_expand_string below for details.

messageType: default set to `MPAS_LOG_OUT`
masterOnly: default set to **.false.**
flushNow: default set to **.true.**

3.1 Main procedures

1. Mainly call `log_expand_string` to replace `$i` `$r` `$l` with specified integer, real, and logical vairable. **Notable:** one dollar character only corresponding one value rather than arrays.

2. It also count different type message.

3. Handle error/critical error and open new error log file, if critical error, masterOnly set false and kill this model call `log_abort`

4. Optional, if critical error occur, firstly write log info to log output and open error file using `mpas_log_open`, write error info to this file and kill model using `log_abort`

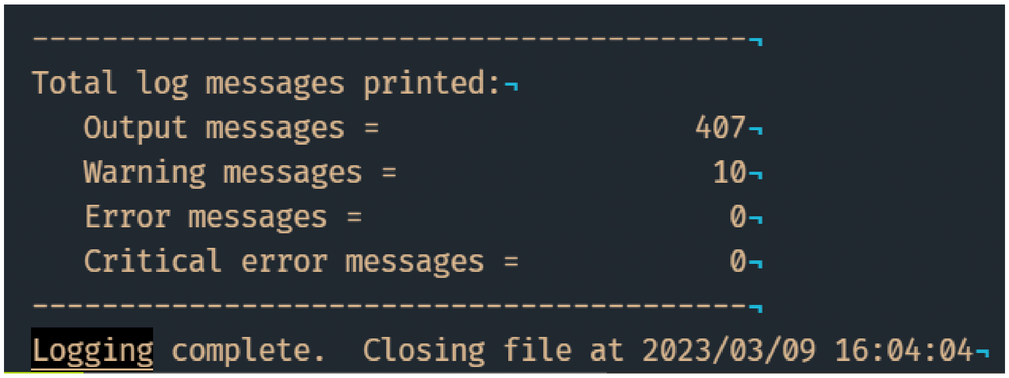
5. If call `log_abort`, first call `mpas_log_finalize` to close log file and output `loggling complete` and close error file.

4 mpas_log_finalize

> finalize the log manager

4.1 Main procedures

1. Print out statistics of how many of each message type were printed.



```
-----------------------------------------¬
Total log messages printed:¬
   Output messages =                  407¬
   Warning messages =                  10¬
   Error messages =                     0¬
   Critical error messages =            0¬
-----------------------------------------¬
Logging complete.  Closing file at 2023/03/09 16:04:04¬
```

2. close the log file (if active and opened by log manager)