Earth System Modeling Framework

LogErr design

Shep Smithline and Erik Kluzek

August 7, 2003

NASA Earth Science Technology Office
Computational Technologies Project
CAN 00-OES-01
http://www.esmf.ucar.edu

Contents

1 Synopsis

The **ESMC_Log** object provides an interface for user to print out log information, as well as error and warning information. This information may either go to standard out or to files.

2 Object Model

All functions of the Log component are handles by the Log object.

3 Global Parameters and Definitions

ESMF_SINGLE_FILE: write log output to a single file

ESMF_MULT_LOG_FILE: write log output to multiple files

ESMF_LOG_TRUE: integer to signify true statement

ESMF_LOG_FALSE: integer to signify false statement

ESMF_LOG_UNIT_NUMBER: used to create fortran unit numbers

ESMF_LOG_FORT_STDOUT: standard out for Fortran

ESMF_LOG_UPPER: upper bound for fortran unit number
ESMF_FATAL : Fatal error

ESMF_WARNING : Non-Fatal error

ESMF_SINGLE_ERR_FILE : Send errors to single file for all PE's

ESMF_MULT_ERR_FILE : All PE's write to seperate files.

ESMF_ERR_RETURN : Return on error

ESMF_ERR_REPORT : Print a detailed error report.

ESMF_WARNINGS_FATAL : Execution will terminate on warnings.

ESMF_WARNINGS_NOT_FATAL : Execution will not terminate on warnings

These are the generic error codes. The user is free to add additional messages within the source code.

ESMF_ERR_MEM : Unable to allocate requested memory

ESMF_ERR_SUP : No support for requested operation

ESMF_ERR_SIG : Signal received

ESMF_ERR_FP : Floating point exception

ESMF_ERR_COR : Corrupted ESMF object detected

ESMF_ERR_LIB : Error in library called by ESMF

ESMF_ERR_PLIB : ESMF generated inconsistent data
ESMF_ERR_MEMC : Memory corrupted
ESMF_ERR_BUSY : Resource is busy
ESMF_ERR_SYS : System call error
ESMF_ERR_ARG_SIZ : Non-comforming object sizes used in operation
ESMF_ERR_ARG_IDN : Two arguments not allowed to be the same
ESMF_ERR_ARG_WRONG : Wrong argument
ESMF_ERR_ARG_CORRUPT : Null or corrupeted ESMF object sent in as argument
ESMF_ERR_ARG_OUTOFRANGE : Input argument out of range
ESMF_ERR_ARG_BADPTR : Invalid pointer argument
ESMF_ERR_ARG_NOTSAMETYPE : Two arguments must be same object type
ESMF_ERR_ARG_NOTSAMECOMM : Two arguments must have the same communicators
ESMF_ERR_ARG_WRONGSTATE : Object in argument is in wrong state
ESMF_ERR_ARG_INCOMP : Arguments are incompatible
ESMF_ERR_FILE_OPEN : Unable to open file
ESMF_ERR_FILE_READ : Unable to read from file
ESMF_ERR_FILE_WRITE : Unable to write to file
ESMF_ERR_FILE_UNEXPECTED : Unexpected data in file
ESMF_ERR_FILE_CLOSE : Unable to close file
ESMF_ERR_INIT : Init method not called
ESMF_ERR_FILE_ACTIVE : Instrumented region is still active

4 LogErr Design

4.1 Description

The Log class consists of a variety of methods for writing error, warning, and miscellaneous log data to files or to standard out. The user may set the code to make the output verbose or not, to halt on encountering errors or warnings, and to flush output buffers after every write.

4.2 Design

Although the class contains a large number of methods, the most Fortran/C++ important methods are ESMF_LogSet()/ESMC_LogSet() and ESMF_LogGet()/ESMC_LogGet(), for setting and getting data; ESMF_LogOpenFile()/ESMC_LogOpenFile() and ESMF_CloseFile()/ESMC_LogCloseFile(), for opening and closing output files; and ESMF_LogErr()/ESMC_LogErr(), and ESMF_LogWarn()/ESMC_LogWarn(), for writing error and warning information. In addition, there are a variety of other methods that allow the user to control other behavior such whether or not to stop execution on encountering errors or warnings, whether output should be flushed from buffers to files automatically, or whether output should be verbose or not.
The Log class was implemented in C/C++, but uses the Fortran I/O libraries when the class methods are called from Fortran. We forced the C/C++ methods to use the Fortran I/O library by creating utility functions that are written in Fortran, but callable from Log's C++ methods. These utility functions call the standard Fortran write, open and close functions. If you call the Log methods from C/C++ code, you bypass the utility functions and all I/O is done with the C I/O libraries.

4.2.1 Class Definition

See the source code for the class definition.

4.2.2 Restrictions

There are a few restrictions to keep in mind. First, the error handling and warning routines, ESMF_LogErr()/ESMC_LogErr(), ESMF_LogErrMsg()/ESMC_LogErrMsg(), ESMF_LogWarn()/ESMC_LogWarn(), and ESMF_LogWarnErr()/ESMC_LogWarnErr(), are expanded using a macro's that adds the predefined symbolic constants __LINE__ and __FILE__ to the argument list of the above error handling routines. Using these constants, we can determine the line number and file that ther error occurred in. If your preprocessor is not working properly, this expansion will not be done properly and the line and file names will not appear correctly when writing out error and warning information. We also rely on the C and the Fortran preprocessor to set a variety of symbolic constants (defined in ESMF_LogConstants.inc and ESMF_ErrConstants.inc). Again, if your preocessor is not working properly, these constants will not be set properly.

5 LogErr F90 Interface

5.1 Use and Examples

5.1.1 Example 1. Simple Example.

In this example we use the default ESMF_Log_World object and illustrate the ESMF_LogGetUnit()to write to the log file aLog.txt.

```
program test_log
  use ESMF_Mod
  implicit none
#include "ESMC_LogErr.inc"
  call ESMF_LogOpenFile(ESMF_Log_World,ESMF_SINGLE_LOG_FILE,"aLog.txt")
  write(ESMF_LogGetUnit(ESMF_Log_World),*)"This is a test."
  call ESMF_LogCloseFile(ESMF_Log_World)
end program
```

5.1.2 Example 2. Simple Error Handling.

Here we illustrate simple error handling. We define an error handler - anErr - and error output goes to the file anErr.txt. There are four options that can be set with ESMF_LogSet() and we set them all here: verbose (whether an error message should be written for anErr), flush (whether output should be flushed), haltOnErr and haltOnWarn (whether the code should halt on an error or warning). While the values set here are the default values, so they need not be set with ESMF_Set(), the example illustrates how to ESMF_SET().

```
#include "ESMC_LogErr.inc"
type(ESMF_Log) :: anErr
integer returnCode
call ESMF_LogSet(anErr, verbose=ESMF_TF_TRUE, flush=ESMF_TF_FALSE,   &
                 haltOnErr=ESMF_TF_TRUE, haltOnWarn=ESMF_TF_FALSE)
call ESMF_LogOpenFile(anErr,ESMF_SINGLE_LOG_FILE,"anErr.txt")
returnCode=foo()
if (returnCode == ESMF_FATAL) call ESMF_LogErr(anErr, returnCode)
call ESMF_LogCloseFile(anErr)
```

5.1.3 Example 3. More error handling.

In this example, we illustrate more complex error handling. We use ESMF_LogGet() to get the value of verbose and if it's set to ESMF_TF_TRUE then turn off verbose midway through the code. This allows you to put in lots of debugging code, and then if you want to turn them off, you can just call an ESMF_LogSet() rather than commenting out all the error handling calls. We also illustrate the use of ESMF_LogErrMsg() which writes an additional message to the err file.

```
#include "ESMC_LogErr.inc"
type(ESMF_Log) :: anErr
type(ESMF_Logical) :: myVerbosity
character(len=32) :: myMsg
integer returnCode
call ESMF_LogSet(anErr, verbose=ESMF_TF_TRUE, flush=ESMF_TF_FALSE,   &
                 haltOnErr=ESMF_TF_TRUE, haltOnWarn=ESMF_TF_FALSE)
call ESMF_LogOpenFile(anErr,ESMF_SINGLE_LOG_FILE,"anErr.txt")
returnCode=foo()
myMsg="This will be written out."
if (returnCode == ESMF_FATAL) call ESMF_LogErrMsg(anErr, returnCode,myMsg)
ESMF_LogGet(anErr,verbose=myVerbosity)
if (myVerbosity==ESMF_TF_TRUE) call ESMF_LogSet(verbose=ESMF_TF_FALSE)
myMsg="This will not be written out."
returnCode=foo()
if (returnCode == ESMF_FATAL) call ESMF_LogErrMsg(anErr, returnCode,myMsg)
call ESMF_LogCloseFile(anErr)
```

==========

5.2 Fortran: Module Interface Fortran Interface to Log class. (Source File: ESMF_LogErr.F90)

The Fortran interface to the `ESMF_Log` class is written in both Fortran and C/C++. This file contains the interface code written in Fortran. It also contains some utility functions used by the `ESMF_Log` class.

5.2.1 ESMF_LogCloseFile - closes a file from Fortran code

INTERFACE:

```
subroutine ESMF_LogCloseFile(aLog)
```

ARGUMENTS:

```
type(ESMF_Log), intent(in) :: aLog
```

DESCRIPTION:

Calls c_esmf_logclosefile() (defined in ESMC_LogInterface.C), the wraper for the method `ESMC_LogCloseFileForWrite` which closes aLog's log file.
The arguments are

aLog an ESMG_Log object

5.2.2 ESMF_LogOpenFile - opens a log file

INTERFACE:

```
subroutine ESMF_LogOpenFile(aLog, numFile, name)
```

ARGUMENTS:

```
type(ESMF_Log) :: aLog

integer :: numFile

character(len=*) :: name
```

DESCRIPTION:

This routine finds the first space in the array name and inserts a a null character. It then calls `ESMC_LogOpenFileForWrite` an `ESMC_Log` method for opening files.
The arguments are:

aLog Log object.

numFile Set to either ESMF_SINGLE_FILE or ESMF_MULTIPLE_FILE

name name of file

5.2.3 ESMF_LogSet - initialize an error object.

INTERFACE:

```
subroutine ESMF_LogSet(aLog, verbose, flush, haltOnErr, haltOnWarn)
```

ARGUMENTS:

```
type(ESMF_Log), intent(in) :: aLog
type(ESMF_Logical), intent(in), optional :: verbose
type(ESMF_Logical), intent(in), optional :: flush
type(ESMF_Logical), intent(in), optional :: haltOnErr
type(ESMF_Logical), intent(in), optional :: haltOnWarn
```

DESCRIPTION:

With the exception of the `ESMF_Log` object, all the arguments are optional. See the Examples section of the document for a discussion of how to use the routine.
The arguments are:

verbose If set to ESMF_TF_TRUE, output written to Log file.

flush If set to ESMF_TF_TRUE, output is flushed.

haltOnWarn If set to ESMF_TF_TRUE, code stops on warnings.

haltOnErr If set to ESMF_TF_TRUE, code stops on errors.

5.2.4 ESMF_LogGet - gets attributes of log object

INTERFACE:

```
subroutine ESMF_LogGet(aLog, verbose, flush, haltOnErr, haltOnWarn)
```

ARGUMENTS:

```
type(ESMF_Log), intent(in) :: aLog
type(ESMF_Logical), intent(out), optional :: verbose
type(ESMF_Logical), intent(out), optional :: flush
type(ESMF_Logical), intent(out), optional :: haltOnWarn
type(ESMF_Logical), intent(out), optional :: haltOnErr
```

DESCRIPTION:

With the exception of the ESMF_Log object, all the arguments are optional. See the Examples section of the document for a discussion of how to use the routine.
The arguments are:

verbose If present, return value in argument.

flush If present, return value in argument.

haltOnWarn If present, return value in argument.

haltOnErr If present, return value in argument.

5.2.5 ESMF_LogWarnMsg - writes a warning message to the log file

INTERFACE:

```
subroutine ESMF_LogWarnMsg_(aLog, errCode, line,file,dir,msg)
```

ARGUMENTS:

```
type(ESMF_Log) :: aLog

integer, intent(in) :: errCode

character(len=*), intent(in) :: msg

integer, intent(in) :: line


character(len=*), intent(in) :: file


character(len=*), intent(in) :: dir
```

DESCRIPTION:

This routine calls c_esmf_logerrmsg in ESMC_LogErrInterface.C to write a warning message to the log file. This warning message consists of the erroCode, a description of the warning, the line number, file, and directory of the error, and a message. A preprocessor macro adds the predefined preprocessor symbolic constants __LINE__, __FILE__, and __DIR__ when `ESMF_LogWarnMsg` is called user code. Note, the value of __DIR__ must be suppliled by the user (usually done in the makefile.). By default, execution continues after encountering a warning, but by calling the routine ESMF_LogWarnHalt(), the user can halt on warnings.
The arguments are

errCode integer value for error code

msg msg written to log file

line line number of warning; argument supplied by macro

file file where warning occurred in;argument supplied by macro

dir directory where warning occurred in; argument supplied by macro

---

5.2.6 ESMF_LogWarn - writes a warning message to log file

INTERFACE:

```
subroutine ESMF_LogWarn_(aLog, errCode,line,file,dir)
```

ARGUMENTS:

```
type(ESMF_Log) :: aLog

integer, intent(in) :: errCode

integer, intent(in) :: line

character(len=*), intent(in) :: file

character(len=*), intent(in) :: dir
```

DESCRIPTION:

This routine is identical to `ESMF_LogWarnMsg`, except a msg is not written to the log file.
The arguments are:

errCode Error code

line line number of warning; argument supplied by macro

file file where warning occurred in; argument supplied by macro

dir directory where warning occurred in; argument supplied by macro

---

5.2.7 ESMF_LogErr - writes a error message to log file

INTERFACE:

```
subroutine ESMF_LogErrMsg_(aLog, errCode,line,file,dir,msg)
```

ARGUMENTS:

```
type(ESMF_Log) :: aLog

integer, intent(in) :: errCode

character(len=*), intent(in) :: msg

integer, intent(in) :: line

character(len=*), intent(in) :: file

character(len=*), intent(in) :: dir
```

DESCRIPTION:

This routine is identical to `ESMF_LogErrMsg`, except a msg is not written to the log file.
The arguments are:

errCode Error code

line line number of warning; argument supplied by macro

file file where warning occurred in; argument supplied by macro

dir directory where warning occurred in; argument supplied by macro

---

5.2.8 ESMF_LogErr - writes a error message to log file

INTERFACE:

```
subroutine  ESMF_LogErr_(aLog, errCode,line,file,dir)
```

ARGUMENTS:

```
type(ESMF_Log) :: aLog

integer, intent(in) :: errCode

integer, intent(in) :: line

character(len=*), intent(in) :: file
```

```
character(len=*), intent(in) :: dir
```

DESCRIPTION:

This routine is identical to `ESMF_LogErrMsg`, except a msg is not written to the log file.
The arguments are:

errCode Error code

line line number of warning; argument supplied by macro

file file where warning occurred in; argument supplied by macro

dir directory where warning occurred in; argument supplied by macro

---

5.2.9 ESMF_LogOpenFortran

INTERFACE:

```
subroutine ESMF_LogOpenFortran(isOpen, unitNumber, nameLogFile)
```

ARGUMENTS:

```
type(ESMF_Logical), intent(out) ::  isOpen


integer, intent(inout) ::  unitNumber

character (len=32), intent(in) :: nameLogFile
```

DESCRIPTION:

This routine opens the log file and is called by `ESMC_LogWrite`. See the ESMC_LogErr.C file for more details. This routine is not a module procedure because F90 mangles the names of functions inside modules and this routine is called by ESMC_LogWrite() - a C++ method.
The arguments are:

isOpen If file successfully opened isOpen set to ESMF_TF_TRUE otherwise set to ESMF_TF_FALSE

unitNumber standard Fortran unit number for I/O

nameLogFile name of log file

5.2.10 ESMF_LogCloseFortran

INTERFACE:

```
subroutine ESMF_LogCloseFortran(unitNumber)
```

ARGUMENTS:

```
integer, intent(in) :: unitNumber
```

DESCRIPTION:

This routine closes any log files that have been written to using the Fortran interface. It is called by by the C/C++ Log method ESMC_LogFinalize().
The arguments are;

unitNumber standard Fortran unit number for I/O

5.2.11 ESMF_LogPrintNewLine - prints a newline character

INTERFACE:

```
subroutine ESMF_LogPrintNewLine(unitNumber,flushSet)
```

ARGUMENTS:

```
type(ESMF_Logical), intent(in)::flushSet
integer, intent(in)::unitNumber
```

DESCRIPTION:

Prints a newline character.
The arguments are:

flushSet If set to ESMF_TF_TRUE, out flushed.

unitNumber standard Fortran unit number for I/O

5.2.12 ESMF_LogPrintString - prints a string

INTERFACE:

```
subroutine ESMF_LogPrintString(unitNumber,stringToPrint,flushSet)
```

ARGUMENTS:

```
type(ESMF_Logical), intent(in)::flushSet
integer, intent(in) :: unitNumber
character(len=*), intent(in)::stringToPrint
```

DESCRIPTION:

This routine, is used by ESMC_LogPrint() and ESMC_LogPrintHeader() in the Log class to print a string. Ordinarily, these Log routines would have just used fprintf. However, because we need to allow one to use the Fortran I/O libraries when Fortran I/O is selected (see the discussion about the class design), we need to call a Fortran routine to do the printing and this is the one!
! The arguments are:

unitNumber standard Fortran unit number for I/O

stringToPrint String to be printed.

flushSet If set to ESMF_TF_TRUE, out flushed.

6 LogErr C++ Interface

6.1 Use and Examples

6.1.1 Example 1. Simple Example.

In this example we use the default ESMF_Log_World object and illustrate the ESMF_LogGetUnit()to write to the log file aLog.txt.

```
#include "ESMC_LogErr.inc"
ESMC_Log_World.ESMC_LogOpen(ESMF_Log_World,ESMF_SINGLE_LOG_FILE,"aLog.txt")
fprintf(ESMC_Log_Wold.ESMC_GetFileHandle(),"This is a test");
ESMC_Log_World.ESMC_LogClose();
```

6.1.2 Example 2. Simple Error Handling.

Here we illustrate simple error handling. We define an error handler - anErr - and error output goes to the file anErr.txt. There are four options that can be set with ESMF_LogSet() and we set them all here: verbose (whether an error message should be written for anErr), flush (whether output should be flushed), haltOnErr and haltOnWarn (whether the code should halt on an error or warning). While the values set here are the default values, so they need not be set with ESMC_Set(), the example illustrates how to use ESMC_SET(). Note that verbose, flush, haltOnErr, and haltOnWarn are symbolic constants defined in ESMC_LogErr.inc.

```
#include "ESMC_LogErr.inc"
ESMC_Log anErr;
int returnCode;
anErr.ESMC_LogSet(verbose,ESMF_TF_TRUE,flush,ESMF_TF_TRUE,haltOnErr,
                  ESMF_TF_TRUE,haltOnWarn,ESMF_TF_FALSE);
anErr.ESMC_LogOpenFile(ESMF_SINGLE_LOG_FILE,"anErr.txt");
returnCode=foo();
if (returnCode == ESMF_FATAL) anErr.ESMC_LogErr(ESMF_FATAL);
anErr.ESMC_LogCloseFile();
```

6.1.3 Example 3. More error handling.

In this example, we illustrate more complex error handling. We use ESMF_LogGet() to get the value of verbose and if it's set to ESMF_TF_TRUE then turn off verbose midway through the code. This allows you to put in lots of debugging code, and then if you want to turn them off, you can just call an ESMF_LogSet() rather than commenting out all the error handling calls.

```
#include "ESMC_LogErr.inc"

\begin{verbatim}
#include "ESMC_LogErr.inc"
ESMC_Log anErr;
ESMC_Logical myVerbosity;
char myMsg[32]="This will be written out.";
int returnCode;
anErr.ESMC_LogSet(verbose,ESMF_TF_TRUE,flush,ESMF_TF_FALSE,
                  haltOnErr,ESMF_TF_TRUE, haltOnWarn,ESMF_TF_FALSE)
anErr.ESMC_LogOpenFile(ESMF_SINGLE_LOG_FILE,"anErr.txt")
returnCode=foo();
if (returnCode == ESMF_FATAL) anErr.ESMC_LogErrMsg(anErr, returnCode,myMsg)
anErr.ESMF_LogGet(verbose,myVerbosity)
if (myVerbosity==ESMF_TF_TRUE) anErr.ESMF_LogSet(verbose=ESMF_TF_FALSE)
myMsg="This will not be written out."
returnCode=foo()
```

```
if (returnCode == ESMF_FATAL) anErr.ESMF_LogErrMsg(anErr, returnCode,myMsg)
anErr.ESMF_LogCloseFile()
```

6.2 Parameters

6.3 C++: Class Interface ESMC_Log - C++ interface to Log (Source File: ESMC_LogErr.h)

The code in this file defines the C++ Log members and declares all class data and methods. All methods, except for the Set and Get methods, which are inlined, are defined in the companion file ESMC_LogErr.C

USES:

```
#include <ESMC_Base.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>
#include <ctype.h>

#include "ESMF_LogConstants.inc"
#include "ESMF_ErrConstants.inc"
#include "ESMC_UtilityFunctions.h"

class ESMC_Log {
  private:
  !PRIVATE MEMBER FUNCIONS:
    void ESMC_LogFormName();
    void ESMC_LogPrintHeader(int fortIO);
    void ESMC_LogPrint(int fortIO, int errCode, int line, char file[],
                       char dir[], char msg[]=NULL);
    void ESMC_LogGetErrMsg(int errCode, char msg[]) const;
    bool ESMC_LogNameValid(char name[], int FortIO);
```

PRIVATE TYPES:

```
    ESMC_Logical oneLogErrFile;
                                // If log data written to one log file,
                                // this is set to true.  Otherwise set to false.
                                // ESMC_OpenFile can override
                                // this value

    ESMC_Logical  standardOut;  // if log data written to standard out,
                                // this variable
                                // is set to true. Otherwise set to false.
                                // ESMC_OpenFile
                                // can over-ride this value.

    ESMC_Logical fortIsOpen;    // used to to a file with Fortran
                                // I/O libraries

    int unitNumber;             // fortran unit number for log/err file when
                                // ESMC\_LogGetUnit
                                // is used  Can be overwritten by
                                // ESMC_OpenFileFortran
```

```
    int numFilePtr;             // index into global array of File pointers
                                // for C++ I/O.


    int numFileFort;            // index into global array of unit numbers for
                                // Fortran I/O

    ESMC_Logical verbose;       // If set to ESMC_TF_TRUE, log
                                // messages written out.

    ESMC_Logical flush;         // If true, all output is flushed

    ESMC_Logical haltOnWarn;    // Code will stop executing on
                                // encountering a warning

    ESMC_Logical haltOnErr;     // Code will stop executing on
                                // encountering an error


    char nameLogErrFile[32];    // name of logfile.
                                // If multiple files are written out,
                                // PE rank is automatically
                                // appended to name.
  public:
```

PUBLIC MEMBER FUNCTIONS:

```
  (see ESMC\_LogErr.C for a description of these methods)
    void ESMC_LogInfo(char* fmt,...);
    void ESMC_LogInfoFortran(char fmt[],
    char charData[],char strData[][32],int intData[], double floatData[]);
    void ESMC_LogOpenFile(int numLogFile,char name[]);
    void ESMC_LogOpenFortFile(int numLogFile, char name[]);
    int ESMC_LogGetUnit();
    void ESMC_LogCloseFile();
    void ESMC_LogCloseFortFile();
    void ESMC_LogSetFlush();
    ESMC_Logical ESMC_LogGetFlush() const;
    void ESMC_LogSetNotFlush();
    void ESMC_LogSetVerbose();
    ESMC_Logical ESMC_LogGetVerbose() const;
    void ESMC_LogSetNotVerbose();
    void ESMC_LogSetHaltOnErr();
    ESMC_Logical ESMC_LogGetHaltOnErr() const;
    void ESMC_LogSetNotHaltOnErr();
    void ESMC_LogSetHaltOnWarn();
    ESMC_Logical ESMC_LogGetHaltOnWarn() const;
    void ESMC_LogSetNotHaltOnWarn();
    void ESMC_LogWarnMsg_(int errCode, int line, char file[],
                      char dir[], char msg[]);
    void ESMC_LogWarn_(int errCode, int line, char file[],
                      char dir[]);
    void ESMC_LogWarnFortran(int errCode, int line, char file[],
         char dir[], char msg[]);
    void ESMC_LogErr_(int errCode, int line, char file[], char dir[]);
    void ESMC_LogErrMsg_(int errCode, int line, char file[],
```

```
                    char dir[], char msg[]);
    void ESMC_LogErrFortran(int errCode,int line,char file[],char dir[],char msg[]);
    void ESMC_LogExit();
    void ESMC_LogSet(char* option, ESMC_Logical value, ...);
    void ESMC_LogGet(char* option, ESMC_Logical value, ...);
    FILE* ESMC_Log::ESMC_GetFileHandle();
    ESMC_Log();
};
```

6.3.1 ESMC_Log() - constructor

INTERFACE:

```
inline ESMC_Log::ESMC_Log(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
   none

   )
```

DESCRIPTION:

This is the constructor. Sets verbose, flush, haltOnErr and haltOnWarn to their default values.

6.3.2 ESMC_LogSetFlush() - set the flushSet variable.

INTERFACE:

```
inline void ESMC_Log::ESMC_LogSetFlush(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     none

    )
```

DESCRIPTION:

Causes output to be flushed.

6.3.3 ESMC_LogGetFlush() - returns the flush variable

INTERFACE:

```
 ESMC_Logical ESMC_Log::ESMC_LogGetFlush (
   !RETURN VALUE
    Value of flush
```

ARGUMENTS:

```
     none

    ) const
```

DESCRIPTION:

Returns the flush variable

6.3.4 ESMC_LogSetNotFlush() - output not flushed

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetNotFlush(
   !RETUN VALUE:
    none
   !ARGUMENTS
     none

  )
```

DESCRIPTION:

Causes output not to be flushed.

6.3.5 ESMC_LogSetVerbose - make output verbose

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetVerbose(
```

RETURN VALUE:

```
    none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If theVerbosity is set to ESMF_TF_TRUE, messages are printed out.

6.3.6 ESMC_LogGetVerbose - return verbose

INTERFACE:

```
 ESMC_Logical ESMC_Log::ESMC_LogGetVerbose(
```

RETURN VALUE:

```
    value of verbose
   !ARGUMENTS
     none
   ) const
```

DESCRIPTION:

Returns verbose value

6.3.7 ESMC_LogSetNotVerbose - output not verbose

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetNotVerbose(
   RETURN VALUE:
    none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If theVerbosity is set to ESMC_TF_FALSE, no messages are printed out.

6.3.8 ESMC_LogSetHaltOnErr - code will stop on encountering an error

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetHaltOnErr(
   RETURN VALUE:
    none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If haltOnErr is set to ESMC_TF_TRUE, code will stop executing when encountering an error.

6.3.9 ESMC_LogGetHaltOnErr - returns haltOnErr

INTERFACE:

```
 ESMC_Logical ESMC_Log::ESMC_LogGetHaltOnErr(
   !RETURN VALUE
    haltOnErr
   !ARGUMENTS
     none
   ) const
```

DESCRIPTION:

Returns haltOnErr

6.3.10 ESMC_LogSetNotHaltOnErr - code will not stop on encountering

an error

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetNotHaltOnErr(
```

RETURN VALUE:

```
    none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If haltOnErr is set to ESMC_TF_FALSE, code will not stop executing when encountering an error.

6.3.11 ESMC_LogSetHaltOnWarn - code will stop on encountering

a warning

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetHaltOnWarn(
```

RETURN VALUE:

```
   none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If haltOnWarn is set to ESMC_TF_TRUE, code will stop executing when encountering an error.

6.3.12 ESMC_LogGetHaltOnWarn - returns haltOnwarn value

a warning

INTERFACE:

```
 ESMC_Logical ESMC_Log::ESMC_LogGetHaltOnWarn(
   !RETURN VALUE
    Value of HaltOnWarn
   !ARGUMENTS
     none
   ) const
```

DESCRIPTION:

Returns haltOnWarn

6.3.13 ESMC_LogSetNotHaltOnWarn - code will not stop on encountering

a warning

INTERFACE:

```
 inline void ESMC_Log::ESMC_LogSetNotHaltOnWarn(
```

RETURN VALUE:

```
    none
   !ARGUMENTS
     none
   )
```

DESCRIPTION:

If haltOnWarn is set to ESMC_Tf_FALSE, code will not stop executing when encountering an error.

6.4 Class API

6.4.1 ESMC_LogOpenFile - opens a Log object

INTERFACE:

```
 void ESMC_Log::ESMC_LogOpenFile(
```

RETURN VALUE:

```
     none
```

ARGUMENTS:

```
      int numLogFile, //number of log files written (input). Set
       // to either ESMF_SINGLE_LOG_FILE or
       // ESMF_MULT_LOG_FILE .

      char name[]     //string to form name of log file (input)
```

```
 )
!DESCRIPTION
{\tt ESMC\_LogErrOpenFile} takes two
arguments.  The first should be set to ESMF\_SINGLE\_LOG\_FILE or
ESMF\_MULT\_LOG\_FILE. These are symbolic constants, defined in
ESMF\_LogConstants.h, set whether one file should be written for all
processes (ESMF_SINGLE\_LOG\_FILE), or whether one file per process should
be written (ESMF\_MULT\_LOG\_FILE).
The second argument is a string and is used to form the name of the
logfile.
This routine is called from native C or C++ code. C I/O libraries are used.
```

6.4.2 ESMC_LogOpenFortFile - opens a Log object

INTERFACE:

```
 void ESMC_Log::ESMC_LogOpenFortFile(
```

RETURN VALUE:

```
     none
```

ARGUMENTS:

```
      int numLogFile, //number of log files written (input). Set
       // to either ESMF_SINGLE_LOG_FILE or
       // ESMF_MULT_LOG_FILE .

      char name[]     //string to form name of log file (input)

 )
!DESCRIPTION
{\tt ESMC\_LogErrOpenFile} takes two
arguments.  The first should be set to ESMF\_SINGLE\_LOG\_FILE or
ESMF\_MULT\_LOG\_FILE. These are symbolic constants, defined in
ESMF\_LogConstants.h, set whether one file should be written for all
processes (ESMF_SINGLE\_LOG\_FILE), or whether one file per process should
be written (ESMF\_MULT\_LOG\_FILE).
The second argument is a string and is used to form the name of the
logfile.
This routine is called from native Fortran code. Fortran I/O libraries are
used.
```

6.4.3 ESMC_LogNameValid - checks to see if a name has been used before

INTERFACE:

```
bool ESMC_Log::ESMC_LogNameValid(
  !RETURN VALUE
     none
  !ARGUMENTS
```

```
       char name[],          // name of file
       int FortIO            //are we doing FortIO?
       )
   !DESCRIPTION
      Checks to see if a file of the name name has been opened by {\tt ESMC\_Log}.
      If it has the function returns a false value.  Note: this function
      use a global array that all {\tt ESMC\_Log} objects have access to.
   EOP
 {
   int i;
   if (FortIO == ESMF_TF_FALSE) {
     for(i=0; i< numCFiles; i++)
      if (strcmp(name,listOfCFileNames[i])  == 0)
  return false;
     strcpy(listOfCFileNames[i-1],name);
     return true;
   } else {
     for(i=0; i< numFortFiles;i++)
       if (strcmp(name,listOfFortFileNames[i])  == 0)
           return false;
     strcpy(listOfFortFileNames[i-1],name);
     return true;
   }
 }
  ------------------------------------------------------------------------------
%////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection [ESMC\_LogInfo] {ESMC\_LogInfo - print contents of a Log}



\bigskip{\sf INTERFACE:}
\begin{verbatim}
 void ESMC_Log::ESMC_LogInfo(
```

RETURN VALUE:

```
      none
```

ARGUMENTS:

```
      char* fmt, // c-style character format; subsequent arguments
      ...
     )
```

DESCRIPTION:

`ESMC_Log_Info` works similar to C's printf statement. The first argument is a character string that can include text to be written as well as a description of the number and kind of characters to be printed out. The format of this string follows the C format description convention, though not every feature is supported. The current conversion specifiers are supported: d (signed decimal integer), f (double values), and s (string).
Any number of data items may be passed to `ESMC_Log_Print`.
The items are printed on a single line. Widths, precision, and escape sequences are not supported. If you specify these, the code ignores them.

6.4.4 ESMC_LogInfoFortran - print contents of a Log from Fortran

INTERFACE:

```
 void ESMC_Log::ESMC_LogInfoFortran(
```

RETURN VALUE:

```
      none
```

ARGUMENTS:

```
      char fmt[],           // array for c-style character format (input)

      char charData[],      // holds character data
      char strData[][32],   // two dimensional array for string data;
                            // strDat[i][j] is the jth
     // character of string i. (input)

      int intData[],        // array storing integer data to be printed (input)

      double floatData[]    // array storing double data to be printed (input)

     )
```

DESCRIPTION:

`ESMC_LogInfoFortran` is the version of `ESMC_LogInfo` used for the Fortran interface. Instead of printing the data from the stack as `ESMC_LogInfo` does, `ESMC_LogInfoFortran` prints the data stored in the "Data" arrays. The routine is called from `ESMF_LogPrint` which is defined in ESMF_LogInterface.C. `ESMF_LogPrint` is callable from the user's from Fortran code. It is this routine that takes the data from the stack and stores it in the "Data" arrays.

6.4.5 ESMC_LogPrintHeader - prints header data

INTERFACE:

```
    void ESMC_Log::ESMC_LogPrintHeader(
```

RETURN VALUE:

```
      none
```

ARGUMENTS:

```
      int fortIO       //if set to ESMF_TF_TRUE, use fortran io libraries

    )
   !Description:
   This is a private method, used by LogErr's various
   print methods to print header
   data to the log file.  This data consists of a time stamp, an ascii string
   holding the day, month, time, and year. If the code is running in parallel
   with MPI, the header also consists of the PE number.
```

6.4.6 ESMC_LogGet - get value of verbose, flush, haltOnErr, and/or haltOnWarn

INTERFACE:

```
void ESMC_LogGet(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
An arbitrary number of "option, value" pairs

char* option, ESMC_Logical value,...)
```

DESCRIPTION:

This method returns the value of the argument option in the variable value. Option may be set tot he string verbose, flush,haltOnErr or haltOnWarn

---

6.4.7 ESMC_LogSet - sets the value of verbose, flush, haltOnWarn and/or haltOnErr

INTERFACE:

```
void ESMC_LogSet(

!RETURN VALUE
  none
!ARGUMENTS
 An arbitrary number of "option, value" pairs
  char* option, ESMC_Logical value,...)
```

DESCRIPTION:

This method returns the value of the argument option in the variable value. Option may be set tot he string verbose, flush,haltOnErr or haltOnWarn

---

6.4.8 ESMC_GetFileHandle - used in conjunction with fprintf to write

data to the log file.

INTERFACE:

```
FILE* ESMC_Log::ESMC_GetFileHandle(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
none
)
```

DESCRIPTION:

This method is used in conjunction with fprintf to write data to the log file, eg. fprintf(aLog.ESMC_GetUnit(),"Hi") The routine writes header data, and then returns a point to a file structure to fprintf

---

6.4.9 ESMC_LogGetUnit - used in conjunction with standard Fortran write

to write data to log file.

INTERFACE:

```
 int ESMC_Log::ESMC_LogGetUnit(
```

RETURN VALUE:

```
      unit number
```

ARGUMENTS:

```
       none.
```

```
    )
```

DESCRIPTION:

This is method is used with the standard Fortran write function, eg. write(ESMC_LogGetUnitNumber(aLog),*) 'Hi '. The routine writes header data, and then returns a unit number to the Fortran write function.

6.4.10 ESMC_LogFormName - private method that forms

the name of the LogErr file to be written when multiple LogErr files are to be written.

INTERFACE:

```
 void ESMC_Log::ESMC_LogFormName(
```

RETURN VALUE:

```
      none
```

ARGUMENTS:

```
       none.
```

```
    )
```

DESCRIPTION:

This routine forms the names the LogErr files by appending the PE number to base name specified in the Open method. This method is used only when running in parallel on multiple PEs.

6.4.11 ESMC_LogCloseFortFile - closes log file.

INTERFACE:

```
 void ESMC_Log::ESMC_LogCloseFortFile(
   ! RETURN VALUE:
      none
```

ARGUMENTS:

```
    none

   )
  ! DESCRIPTION:
  This routine simply closes the log file(s).  It also removes
  file from the global file array. The routine is called from native
  Fortran code (File is closed with Fortran I/O libraries.)
```

6.4.12 ESMC_LogCloseFile - closes log file.

INTERFACE:

```
 void ESMC_Log::ESMC_LogCloseFile(
   ! RETURN VALUE:
      none
```

ARGUMENTS:

```
    none

   )
  ! DESCRIPTION:
  This routine simply closes the log file(s).  It also removes
  file from the global file array. The routine is called from native
  C/C++ code (File is closed with C I/O libraries.)
```

6.4.13 ESMC_Log - native C++ constructor for log class

INTERFACE:

```
 ESMC_Log::ESMC_Log(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
    none

  )
```

DESCRIPTION:

Native C++ constructor

6.4.14 ESMC_LogErr - write error message to log file

INTERFACE:

```
void ESMC_Log::ESMC_LogErr_(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
int errCode,     // Error code
int line,        // Line number
char file[],     // Filename
char dir[]      // Directory
)
```

DESCRIPTION:

Prints error code, corresponding message, line number, file, directory that error occurred at.

6.4.15 ESMC_LogErrMsg - write error message to log file

INTERFACE:

```
void ESMC_Log::ESMC_LogErrMsg_(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
int errCode,     // Error code
int line,        // Line number
char file[],     // Filename
char dir[],     // Directory
char msg[]
)
```

DESCRIPTION:

Prints error code, corresponding message, line number, file, directory that error occurred at. Can also print a message.

6.4.16 ESMC_LogWarn – Print warning message

INTERFACE:

```
void ESMC_Log::ESMC_LogWarn_(
```

ARGUMENTS:

```
      int errCode,     // Error code
      int line,        // Line number
      char file[],     // Filename
      char dir[]      // Directory
    )
```

DESCRIPTION:

Same as ESMC_LogErr, except execution is not stopped after printing message, except when haltOnWarn set to true

---

6.4.17 ESMC_LogWarnMsg – Print warning message

INTERFACE:

```
      void ESMC_Log::ESMC_LogWarnMsg_(
```

ARGUMENTS:

```
      int errCode,     // Error code
      int line,        // Line number
      char file[],     // Filename
      char dir[],      // Directory
      char msg[]
    )
```

DESCRIPTION:

Same as ESMC_LogErr, except execution is not stopped after printing message, except when haltOnWarn set to true

---

6.4.18 ESMC_LogExit() - private routine uses by Log to stop program

INTERFACE:

```
 void ESMC_Log::ESMC_LogExit(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
    none
   )
```

DESCRIPTION:

Used by ESMC_Log to exit program.

6.4.19 ESMC_LogErrFortran - called by fortran wrapper to write error msg

INTERFACE:

```
 void ESMC_Log::ESMC_LogErrFortran(
```

ARGUMENTS:

```
     int errCode,          //error code
     int line,             //line number
     char file[],          //file error occurred in
     char dir[],           //directory error occurred in
     char msg[]            //msg
   )
```

DESCRIPTION:

Similar to `ESMC_LogErr`, except called by the fortran wrapper esmf_logerr which is defined in ESMC_Interface.C. The major difference between this routine and `ESMC_LogErr` is that this routine prints the printf style data from the Data arrays not the stack.

6.4.20 ESMC_LogWarnFortran - called by fortran wrapper to write warnings

INTERFACE:

```
 void ESMC_Log::ESMC_LogWarnFortran(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
      int errCode,
      int line,
      char file[],
      char dir[],
      char msg[]

 )
```

DESCRIPTION:

Similar to `ESMC_LogWarn`, except called by the fortran wrapper esmf_logerr which is defined in ESMC_Interface.C

6.4.21 ESMC_LogPrint - prints to the log file

INTERFACE:

```
 void ESMC_Log:: ESMC_LogPrint(
```

ARGUMENTS:

```
    int fortIO,          //if set to ESMF_TF_TRUE use Fortran IO libraries
    int errCode,
    int line,            // see LogErr for a definition
    char file[],         // of these variables
    char dir[],
    char msg[]            // optional msg

   )
```

DESCRIPTION:

This is a private routine, used by many methods of ESMC_Log to print data to the log file. If used from Fortran, then Fortran I/O libraries are used. Otherwise C I/O libraries are used.

6.4.22 ESMC_LogGetErrMsg – Return error message for given error code.

INTERFACE:

```
void ESMC_Log::ESMC_LogGetErrMsg(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
int errCode,
char msg[]
) const
```

DESCRIPTION:

ESMC_GetErrMsg returns a string corresponding to the error code

6.5 Wrapper Functions

These wrapper function are called by the FORTRAN routines defined in the file ESMF_LogErr.F90. The structure is: ESMF_LogFoo(aLog) —> C_ESMF_LogFoo(aLog) —-> aLog.ESMC_LogFoo() In other words, these routines actually call the C++ methods. You can't call the C++ methods directly from FORTRAN. You must go thru this intermediate step.

6.5.1 C_ESMF_LogCloseFile - closes a file from Fortran code

INTERFACE:

```
void FTN(c_esmf_logclosefile)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

Calls the method ESMC_LogCloseFileForWrite to close aLog's log file.

6.5.2 C_ESMF_LogOpenFile - opens a log file

INTERFACE:

```
void FTN(c_esmf_logopenfile)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log *aLog,
int *numFiles,
char name[],
int namelen)
```

DESCRIPTION:

This routine finds the first space in the array name and inserts a a null character. It then calls `ESMC_LogOpenFileForWrite` an `ESMC_Log` method for opening files.

6.5.3 ESMF_LogInfo - writes miscellaneous information to a log file

INTERFACE:

```
void FTN(esmf_loginfo)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog,
char* fmt,...)
```

DESCRIPTION:

This routine allows the user to write miscellaneous information the `ESMC_Log` file. It uses a printf style character descriptor, e.g. `ESMC_LogInfo`(aLog,"Hi there, a character string. The routine takes a variable number of arguments, so that any number of data items can be written to the `ESMC_Log` file. Currently, only character, strings, integers, and reals are supported. However, field widths, precisions, and flags are ignored.

6.5.4 C_ESMF_LogWarnMsg- writes warning messages

INTERFACE:

```
void FTN(c_esmf_logwarnmsg)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
        ESMC_Log *aLog,
        int *errCode,
        int *line,
        char file[],
        char dir[],
        char msg[],
        int filelen,
        int dirlen,
        int msglen)
```

DESCRIPTION:

This routine is called by `ESMF_LogWarnMsg` (defined in ESMF_LogErr.F90). `C_ESMF_LogWarnMsg` calls the C++ method that actually writes the warning.

6.5.5 C_ESMF_LogWarn - writes warning messages

INTERFACE:

```
       void FTN(c_esmf_logwarn)(
   !RETURN VALUES:
   none
```

ARGUMENTS:

```
        ESMC_Log *aLog,
        int *errCode,
        int *line,
        char file[],
        char dir[],
        int filelen,
        int dirlen)
```

DESCRIPTION:

This routine is called by `ESMC_LogWarn` (defined in ESMF_LogErr.F90). `C_ESMC_LogWarn` calls the C++ method that actually writes the warning.

6.5.6 C_ESMF_LogSetFlush - flushes output

INTERFACE:

```
        void FTN(c_esmf_logsetflush)(
```

RETURN VALUE:

```
       none
```

ARGUMENTS:

```
       ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls the `ESMC_Log` method that flushes output.

6.5.7 C_ESMF_LogSetNotFlush - prevents output from being flushed

INTERFACE:

```
void FTN(c_esmf_logsetnotflush)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls the Log method `ESMC_LogNotFlush()` which sets a flag that turns off flushing. By default, this flag is set.

6.5.8 C_ESMF_LogSetVerbose - causes output to be written to the Log

INTERFACE:

```
void FTN(c_esmf_logsetverbose)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

This routine sets a flag that causes all output associated with the aLog `ESMC_Log` handle to be written.

6.5.9 C_ESMF_LogNotVerbose - causes output not to be written to the Log

INTERFACE:

```
void FTN(c_esmf_logsetnotverbose)(
```

RETURN VALUE:

```
NONE
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

This routine sets a flag that forces all output associated with the aLog `ESMC_Log` handle from being written.

6.5.10 C_ESMF_LogGetUnit - Fortran style method to write to log file.

INTERFACE:

```
int  FTN(c_esmf_loggetunit)(
```

RETURN VALUE:

```
A Fortran Unit Number
```

ARGUMENTS:

```
ESMC_Log *aLog)
```

DESCRIPTION:

This function called from with a Fortran write statement, e.g. write(LogWrite(aLog),*)"Hi". The `ESMC_LogWrite` function appends some header information (time,date etc.) to what ever is printed out from the write, e.g. Hi.

6.5.11 C_ESMF_LogErrMsg - writes warning messages

INTERFACE:

```
void FTN(c_esmf_logerrmsg)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log *aLog,
int *errCode,
int *line,
char file[],
char dir[],
char msg[],
int filelen,
int dirlen,
int msglen)
```

DESCRIPTION:

This routine is called by `ESMF_LogErrMsg` (defined in ESMF_LogErr.F90). `C_ESMF_LogErrMsg` calls the C++ method that actually writes the warning.

6.5.12 C_ESMF_LogErr - writes warning messages

INTERFACE:

```
void FTN(c_esmf_logerr)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log *aLog,
int *errCode,
int *line,
char file[],
char dir[],
int filelen,
int dirlen)
```

DESCRIPTION:

This routine is called by `ESMF_LogErr` (defined in ESMF_LogErr.F90). `C_ESMC_LogErr` calls the C++ method that actually writes the warning.

6.5.13 C_ESMF_LogSetHaltOnErr - program halts on encountering an error

INTERFACE:

```
void FTN(esmf_logsethaltonerr)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that sets a flag to stop execution on reaching an error. This is the default behavior of the `ESMC_Log` class.

6.5.14 C_ESMF_LogSetNotHaltOnErr - program does not halt on an error

INTERFACE:

```
void FTN(c_esmf_logsetnothaltonerr)(
```

RETURN VALUE:

```
none
```

ARGUMENTS:

```
ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that sets a flag to prevent the program from stopping reaching an error.

6.5.15 C_ESMF_LogSetHaltOnWarn - program halts on encountering a warning

INTERFACE:

```
 void FTN(c_esmf_logsethaltonwarn)(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that sets a flag to stop execution on reaching a warning.

6.5.16 C_ESMF_LogSetNotHaltOnWarn - program does not halt on warning

INTERFACE:

```
     void FTN(c_esmf_logsetnothaltonwarn)(
```

RETURN VALUE:

```
     none
```

ARGUMENTS:

```
       ESMC_Log* aLog)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that sets a flag to prevent the program from stopping reaching an error.

6.5.17 C_ESMF_GetVerbose - gets verbose flag

INTERFACE:

```
 void FTN(c_esmf_loggetverbose)(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     ESMC_Log* aLog, ESMC_Logical* verbose)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that gets the verbose flag

6.5.18 C_ESMF_LogGetFlush - gets flush flag

INTERFACE:

```
 void FTN(c_esmf_loggetflush)(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     ESMC_Log* aLog, ESMC_Logical* flush)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that gets the flush flag

6.5.19 C_ESMF_LogGetHaltOnErr - gets HaltOnErr flag

INTERFACE:

```
 void FTN(c_esmf_loggethaltonerr)(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     ESMC_Log* aLog,ESMC_Logical* haltOnErr)
```

DESCRIPTION:

This routine gets the HaltOnErr flag

6.5.20 C_ESMF_LogGetHaltOnWarn - gets HaltOnErr flag

INTERFACE:

```
 void FTN(c_esmf_loggethaltonwarn)(
```

RETURN VALUE:

```
    none
```

ARGUMENTS:

```
     ESMC_Log* aLog,ESMC_Logical* haltOnWarn)
```

DESCRIPTION:

This routine calls a `ESMC_Log` method that gets the HaltOnWarn flag.

7 LogErr Design

8 Review Status

Design Review

Review Date:	<Date>	
Reviewers:	Reviewer	<Institution>
	Reviewer	<Institution>
	Reviewer	<Institution>

item1 <Definition of item1.>

item2 <Definition of item2.>

References