The `Log` object TCC

The purpose of a `Log` object is to manage streams of error messages. Design goals for this object include:

- All error messages are always written to the standard error stream (`&errout`).
- A standardized prefix is attached to all error message lines so they will stand out if mingled with other streams.
- If desired, all messages will also be written to a log file.
- The `Log` object can be passed to any routine that might need to generate an error message, but all such messages go to the same places.
- The object can be queried to find out how many error messages have been sent to it. These messages can be of multiple classes (error, warning, informational, and so on) and counts of each error class are maintained separately.

Methods

`Log_Open`

To create a `Log` object, use this method:

```
Log_Open ( fileName, prefix )
```

where the arguments have these definitions:

`fileName`	The name of the log file to be written, as a string. This argument can be omitted or `&null`, in which case the messages are written only to the standard error stream.
`prefix`	A string that will be prefixed to all error messages. If omitted or `&null`, the error prefix will be `"*** "`.

This method returns a `Log` object that must be passed to the remaining methods.

`Log_Close`

To close a `Log` object:

```
Log_Close ( log )
```

where `log` is a `Log` object.

`Log_Error_Kind`

To send an error message to a `Log` object, use this method:

```
Log_Error_Kind ( log, errKind, text, ... )
```

where:

`log`	The `Log` object.
`errKind`	A string indicating what kind of error this is. The string will be included in the error message just after the prefix. This argument is optional and defaults to `"Error"`.
`text`	One or more strings that will be concatenated to form the final error message.

Here is an example:

```
Log_Error_Kind ( log, "Warning",
                      "We do not recommend more than ",
                      MAX_TURKEYS,
                      " turkeys." );
```

`Log_Error`

This method is similar to `Log_Error_Kind()`, but the `kind` argument defaults to `"Error"`:

```
Log_Error ( log, text, ... )
```

`Log_Error_Count`

To find out how many errors of a given kind have been sent to a `Log` object:

```
Log_Error_Count( log, kind )
```

where `kind` is an optional string, as passed to `Log_Error_Kind()`, defaulting to `"Error"`.

`Log_Fatal`

To write a message and then stop execution:

```
Log_Fatal ( log, text, ... )
```

This method is like `Log_Error()` except that it halts execution with status code 1.

`Log_Message`

To write a multi-line error message, use `Log_Error()` or `Log_Error_Kind()` for the first line, then use this method for the remaining lines:

```
Log_Message ( log, text, ... )
```

The arguments are concatenated to form the actual error message, but the method does not increment the count of errors of any kind.

`Log_Write`

To write messages on a log without the standard error prefix:

```
Log_Write ( log, text, ... )
```

As with `Log_Message()`, this method does not increment any error counts.

Intended functions

More information about my particular variant of the Cleanroom software development methodology can be found at:

`http://www.nmt.edu/~john/soft/clean`

Here are the Cleanroom intended functions for the above methods.

```
#--
# Log_Open ( fileName, prefix )
#   [ if fileName is a string or & null, and
#     prefix is & null or a nonempty string ->
#       if fileName is & null or the name of a file that can be
#       opened for writing ->
#         return a new Log object, logging to & errout, and also to
#         fileName if given, and using eff-prefix(prefix) as the
#         error message prefix
#       else ->
#         & errout  :=  a message about how we can't open (fileName)
#         return a new Log object, logging to & errout, and using
#         eff-prefix(prefix) as the error message prefix
#   ]
#--
# Log_Error ( self, text, ... )
#   [ if the (text) arguments are all strings ->
#       self  ||:=  an error of the default kind, prefixed by
#                         self's prefix, whose text is made from all
#                         the (text) arguments
#       fail
#   ]
#--
# Log_Error_Count ( self, errKind )
#   [ returns the number of times Log_Error() has been called on self
#     with kind (errKind, defaulting to DEFAULT_ERR_KIND), as an integer
#   ]
#--
# Log_Error_Kind ( self, errKind, text, ... )
#   [ if (errKind) is & null or a string, and all the (text) arguments
#     are strings ->
#       self  ||:=  an error of kind (errKind, defaulting to
#                   DEFAULT_ERR_KIND), prefixed by self's prefix,
#                   whose message is the concatenation of all the
#                   (text) arguments
#       fail
#   ]
#--
```

```
#--
# Log_Fatal ( self, text, ... )
#   [ if the (text) arguments are all strings ->
#       self  ||:=  an error message with self's prefix, made from
#                   the concatenation of the (text) arguments
#       stop execution with error code 1
#   ]
#--
# Log_Message ( log, text, ... )
#   [ if the (text) arguments are all strings ->
#       self  ||:=  a message (not considered an error), prefixed
#                   with self's prefix, made from the concatenation
#                   of the (text) arguments
#   ]
# Note: For multi-line error messages, use Log_Error[_Kind]() for
# the first line and Log_Message() for the remaining lines, so it
# will be counted as only one error.
#--
# Log_Write ( self, text, ... )
#   [ if the (text) arguments are all strings ->
#       self  ||:=  a message (not considered an error), with no
#                   prefix, and made from the concatenation of the
#                   (text) arguments
#   ]
# Note: Log_Message() starts the line with the standard prefix,
# while Log_Write() sends the text out unadorned.
#--
# Log_Close ( self )
#   [ if Log_Error_Count(self) is nonzero ->
#       fails
#     else ->
#       returns & null
#     In any case ->
#       self  :=  self with the log file (if any) closed
#   ]
#--
```

Written by John W. Shipman (`john@nmt.edu`). This version printed 1997-09-24.