Reading Log and Text Files

In the previous section, you discovered that the `error_log` method writes to a log file using just one line of code. It creates the log file if it does not exist. It appends (adds to the end of the file) any message passed to the contents of the file. It then closes the file. If you were to create your own logging process, it would take several lines of code.

```
$logFile = fopen("error_log.log", "a");
$eMessage = $e->getMessage();
fwrite($logFile, $eMessage);
fclose($logFile);
```

The `fopen` method will also create the file if it does not already exist. The “`a`” parameter indicates that anything written to the file should be appended. “`w`” would indicate that any contents in the file would be lost (written over). The `fwrite` method will then place the string located in the second parameter (`$eMessage`) into the file indicated by the first parameter (`$logFile`). `$logFile` is a pointer that points to the location of the text file. The `fclose` method closes the text file.

For more information on writing to text files, visit
Examples: visit w3schools at: http://www.w3schools.com/php/php_file_create.asp
Video: visit “The New Boston” at https://www.thenewboston.com/videos.php?cat=11&video=17063

Since a log file is a text-based file, you can use similar logic to create your own application to open a log file and read its contents.

```
$logFile = fopen("error_log.log", "r");
echo fgets($logFile);
fclose($logFile);
```

This code will open the log file and read the first line (via a `fgets` method) in the file and close the file. However, it is likely that there is more than one line in the file. You must be able to loop through and display each line in the file. You can do this using the `while` loop shown here.

```
$logFile = fopen("error_log.log", "r");
while(!feof($logFile))
{
          echo fgets($logFile) . "<br>";
}
fclose($logFile);
```

The `while` loop will continue to loop as long as the conditional statement is `TRUE`. Once the statement is `FALSE`, the code will exit the loop and jump to the next line of code after the end of the loop. In this example the `error_log` file is open for read only (“`r`”). The `while` loop looks at the end of file indicator (`feof`) of the log file to determine if it has reached the end of the file. If `feof` returns `TRUE`, the end of the file has been reached. The loop must continue while you have not reached the end of the file. To cause the conditional statement to produce a `TRUE`, while there are still records to be read, you must reverse the logic and have `feof` produce `TRUE` if there are records and `FALSE` if there are not records. You can do this by using the `!` operator. The `!` operator is a `NOT` operator and it reverses the result. A `NOT TRUE` is `FALSE` or a `NOT FALSE` is