File Handling in CPP

# **File Stream V/s File**

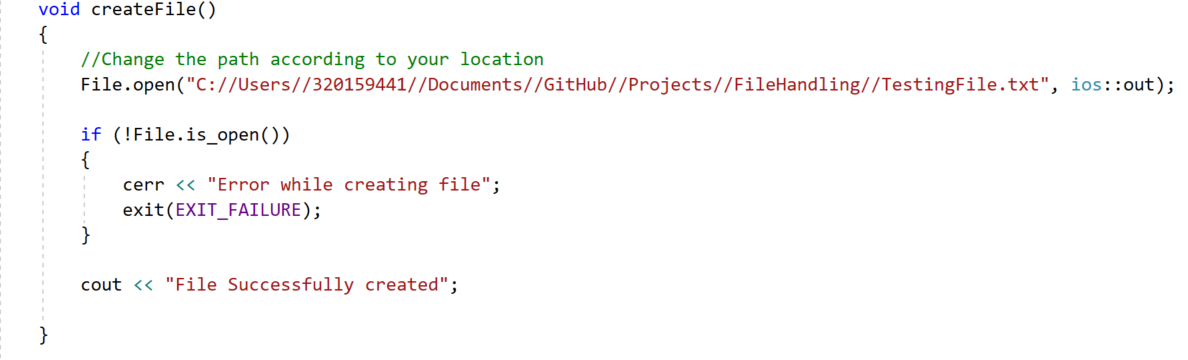
In C++ and many other programming languages, **fstream** and **FILE** are mechanisms to handle file input and output operations, but they are based on different libraries and have different characteristics. Here are the primary differences between the two:

1. **Library Origin**:
   * **Fstream** : **Belongs to the C++ Standard Library**. It is object-oriented **and part of the iostream header**.
   * **FILE**: Belongs to the C Standard Library. It's a structure used in conjunction with functions like **fopen**, **fclose**, **fread**, **fwrite**, etc., from the **cstdio** (or **stdio.h** in C) header.
2. **Object-Oriented vs. Procedural**:
   * **fstream**: Provides an object-oriented approach. File operations are performed using classes and their member functions.
   * **FILE**: Uses procedural functions for file operations. You need to use functions like **fopen**, **fread**, etc., to perform operations on **FILE** pointers.
3. **Ease of Use**:
   * **fstream**: Often considered more user-friendly and intuitive, especially for those familiar with C++ and object-oriented programming.
   * **FILE**: Might be a bit more cumbersome to use because of the need to handle pointers and use procedural functions.
4. **Error Handling**:
   * **fstream**: Utilizes exceptions for error handling. This means you can use try-catch blocks to handle file-related exceptions.
   * **FILE**: Uses return values (like **NULL** for file opening failures) or **errno** to indicate errors. You typically check these return values after every file operation.
5. **Portability**:
   * **fstream**: Being a part of the C++ Standard Library, it is more portable across different platforms and compilers that support the C++ standard.
   * **FILE**: Also quite portable since it's a part of the C Standard Library, but you might encounter slight differences or nuances across different platforms or compilers.
6. **Flexibility**:
   * **fstream**: Provides more flexibility in terms of file operations, especially when combined with other C++ features.
   * **FILE**: While it offers a wide range of file-related functions, its procedural nature might be less intuitive for those who prefer an object-oriented approach.

In summary, while both **fstream** and **FILE** serve the purpose of file I/O in C++ and C, respectively, **fstream** offers a more modern and object-oriented approach, whereas **FILE** provides a traditional procedural mechanism for file operations. Your choice between the two might depend on your familiarity with the programming paradigm, the specific requirements of your project, and personal preference.

# **Creation of File**

* **Use the fstream Library**: This library is essential for file operations in C++.
* **Instantiate an fstream Object**: Create an object from the **fstream** class and name it **File**.
* **Open the File for Writing**: Using the **open()** method on the **File** object, you can initialize a new file in write mode (**out**).
* **Check File Creation**: Utilize the **is\_open()** method to confirm if the file creation was successful.



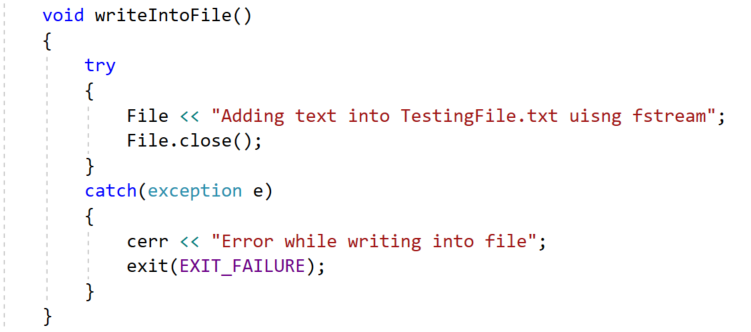
Different modes available in **open()** func are :

|  |  |
| --- | --- |
| Modes | Description |
| ios::out | Opens the file to write(default for ofstream) |
| ios::in | Opens the file to read(default for ifstream) |
| ios::binary | Opens the file in binary mode |
| ios::app | Opens the file and appends all the outputs at the end |
| ios::ate | Opens the file and moves the control to the end of the file |
| ios::trunc | Removes the data in the existing file |

**Note:** We can combine the different modes using or symbol **|**

# **Writing into File**

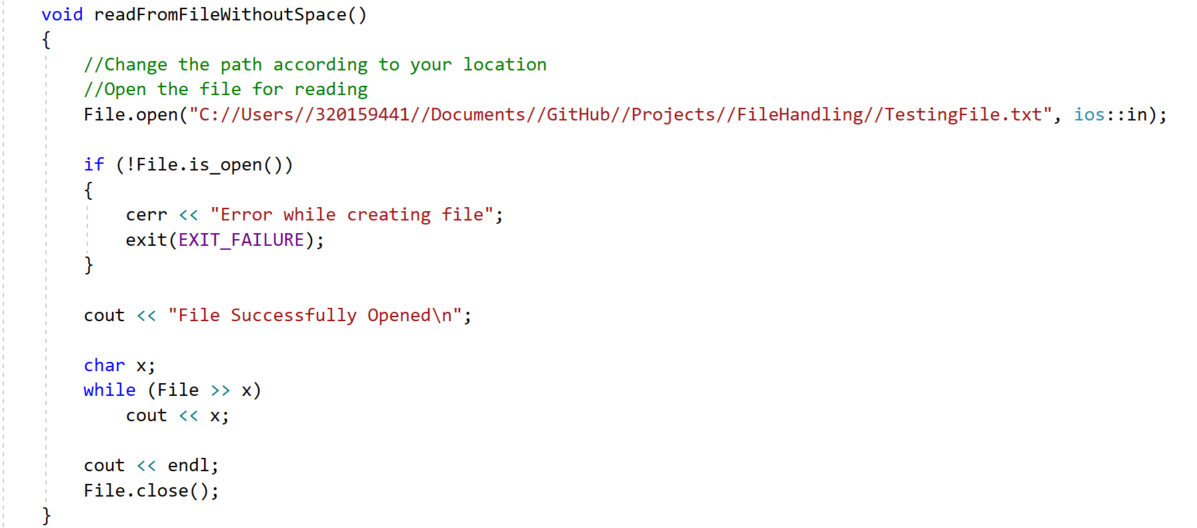
* After Successful creation of file, you can write into file.
* **Syntax** : FileName << "Insert the text here";
* We use the **close()** function on the object to close the file.



# **Reading from File**

* **Open the File for Reading**: Using the **open()** method on the **File** object, you can open a new file in read mode using **ios::in**.

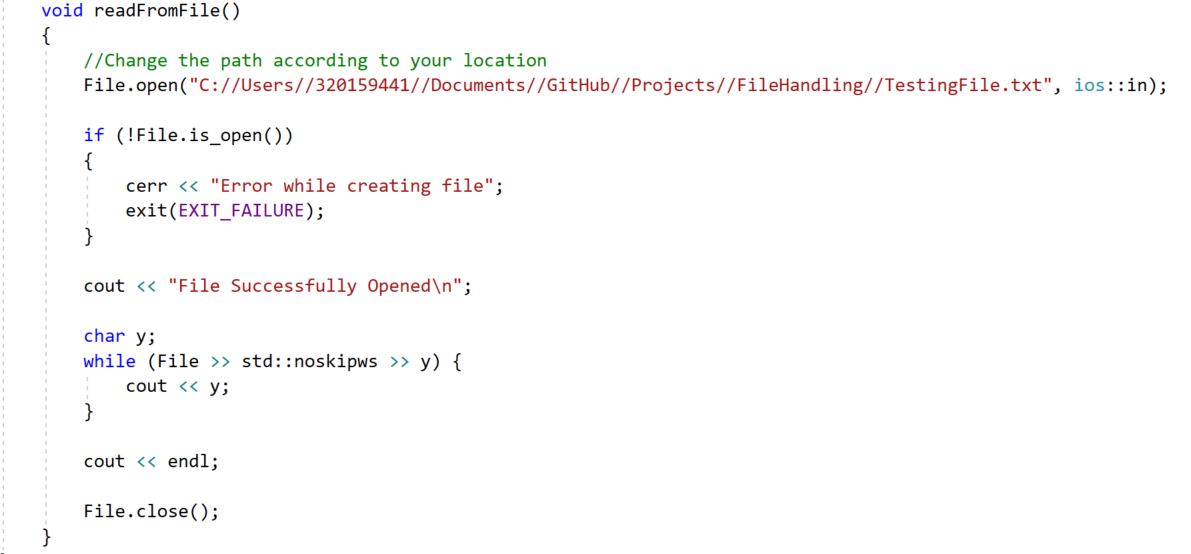
**Method 1:**



Content Present in File : *Adding text into TestingFile.txt using fstream*

Output: *AddingtextintoTestingFile.txtusingfstream*

**Method 2:**



Content Present in File : *Adding text into TestingFile.txt using fstream*

Output: *Adding text into TestingFile.txt using fstream*

* **std::noskipws**: This manipulator ensures that the whitespace characters (like spaces) are not skipped. By default, the **>>** operator of the stream skips whitespace characters.

**Method 3:**



# **Adding New Line**

|  |  |
| --- | --- |
| ios::app | Opens the file and appends all the outputs at the end |

