I / O file in C

A file is a collection of data stored on the drive with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.

A file is a collection of data stored on the drive with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream** .

Basically, stream is a sequence of bytes passed through the path. There are two important **streams** : **Input stream** and **Output stream** . Input stream is used to read data from file (read action) and Output stream is used to write to file (write action).

I / O classes in C

System.IO Namespace has many different layers, used to perform file operations, such as creating and deleting files, reading and writing a file, closing a file .

The following table shows some commonly used **non-abstract** classes in the System.IO namespace in C #:

I / O Class Description BinaryReader Read primitive data from a binary stream. BinaryWriter Write the original data in binary format. BufferedStream Temporary memory for the bytes of a stream. Directory Help control directory structure. DirectoryInfo Used to perform operations on directories. DriveInfo Provides information for drives. File Helps in manipulating files. FileInfo Used to perform operations on files. FileStream Use to read and write any location in a file. MemoryStream Used for random access to streams stored in memory. Path Performs operations on path information. StreamReader Used to read characters from a stream of bytes. StreamWriter Used to write characters to a stream. StringReader Used to read from a string buffer. StringWriter Used to write to a string buffer.

FileStream class in C

FileStream class in the System.IO namespace in C # helps read, write, and close files. This class derives from the abstract class, Stream.

You need to create a FileStream object to create a new file or open an existing file. The syntax for creating a FileStream object in C # is as follows:

```
FileStream  t ê n_ ?? i_t ?? ng > = new FileStream(  t ê n_file > , ,
, );
```

For example : Create a FileStream object F to read a file with the name vidu.txt, as follows:

```
FileStream F = new FileStream(" vidu.txt ", FileMode.Open, FileAccess.Read,
FileShare.Read);
```

ParameterDescriptionFileMode
The FileMode enumerator defines different methods to open the file. The members of the FileMode enumerator are:

1. **Append** : Open an **existing** file and place the cursor at the end of the file, or create a File, if the file doesn't already exist.
2. **Create** : Create a new file.
3. **CreateNew** : Determine with the operating system that it will create a new file.
4. **Open** : Open an existing file.
5. **OpenOrCreate** : Determine with the operating system that it will open a file if it exists, otherwise it will create a new file.
6. **Truncate** : Open an existing file and truncate its size to 0 bytes.

FileAccess
FileAccess enumerators have members: **Read** , **ReadWrite** and **Write** .

FileShare
FileShare enumerators have the following members:

1. **Inheritable** : Allows a traditional file to inherit to child processes.
2. **None** : Reject the current file sharing.
3. **Read** : Allow to open the file to read.
4. **ReadWrite** : Allows opening files to read and write.
5. **Write** : Allows you to open files for recording.

For example:

Here is an example to illustrate how to use the **FileStream** class in C #:

```
using System ; using System . IO ; namespace VdIO { class Program { static void
```

Use the **Console.ReadKey** command (); to see the results more clearly. Compiling and running the above C # program will produce the following results:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

Advanced file operations in C

The previous example illustrates simple operations on files in C #. However, to take full advantage of the System.IO classes in C #, you need to know the commonly used properties and methods for these classes.

1. **Reading and writing text files:** StreamReader and StreamWriter classes help to read and write text files.
2. **Read and write binary files:** The BinaryReader and BinaryWriter classes help to read and write binary files.
3. **Manipulating Windows file system:** It provides C # programmers the ability to browse and locate files and folders in Windows.

Follow tutorialspoint

Previous article: Handling exceptions (Try / Catch / Finally) in C #

Next lesson: Attribute in C #