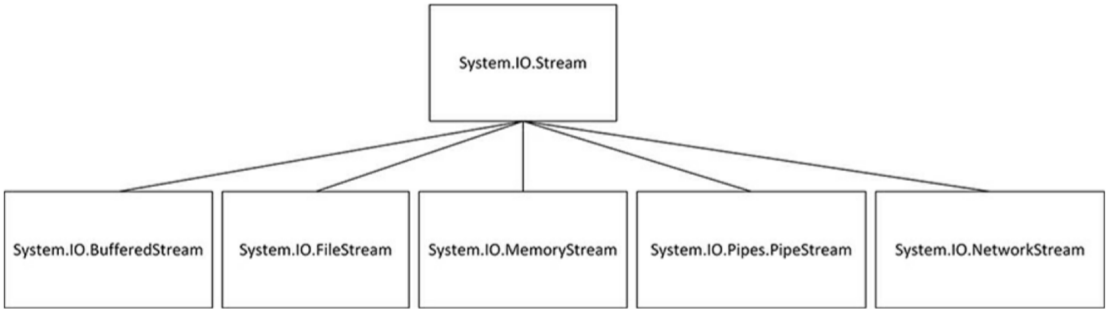
Skill 4.1: Perform I/O operations

Any object that can work with a Stream can work with any of the objects that behave like a stream.



FileStream (use async method perfered)

新建了一个FileStream, 输出bytes, 再新建一个FileStream读取刚刚的输出。

```
using System;
using System.IO;
using System.Text;

namespace LISTING_4_1_Using_a_FileStream
{
    class Program
    {
        static void Main(string[] args)
        {
            // Writing to a file
            FileStream outputStream = new FileStream("OutputText.txt", FileMode.
            OpenOrCreate, FileAccess.Write);
            string outputMessageString = "Hello world";
            byte[] outputMessageBytes = Encoding.UTF8.GetBytes(outputMessageString);
            outputStream.Write(outputMessageBytes, 0, outputMessageBytes.Length);
            outputStream.Close();

            FileStream inputStream = new FileStream("OutputText.txt", FileMode.Open,
            FileAccess.Read)
            long fileLength = inputStream.Length;
            byte[] readBytes = new byte[fileLength];
            inputStream.Read(readBytes, 0, (int)fileLength);
            string readString = Encoding.UTF8.GetString(readBytes);
            inputStream.Close();
            Console.WriteLine("Read message: {0}", readString);
            Console.ReadKey();
        }
    }
}
```

Control file use with FileMode and FileAccess

> The base Stream class provides properties that a program can use to determine the abilities of a given stream instance (whether a program can read, write, or seek on this stream).

- The FileMode enumeration is used in the constructor of a FileStream to indicate how the file is to be opened.

- **FileMode.Append** Open a file for appending to the end. If the file exists, move the seek position to the end of this file. If the file does not exist; create it. This mode can only be used if the file is being opened for writing.
- **FileMode.Create** Create a file for writing. If the file already exists, it is overwritten. Note that this means the existing contents of the file are lost.
- **FileMode.CreateNew** Create a file for writing. If the file already exists, an exception is thrown.
- **FileMode.Open** Open an existing file. An exception is thrown if the file does not exist. This mode can be used for reading or writing.
- **FileMode.OpenOrCreate** Open a file for reading or writing. If the file does not exist, an empty file is created. This mode can be used for reading or writing.
- **FileMode.Truncate** Open a file for writing and remove any existing contents.

- The FileAccess enumeration is used to indicate how the file is to be used. The following access types are available:

  - **FileAccess.Read** Open a file for reading.
  - **FileAccess.ReadWrite** Open a file for reading or writing.
  - **FileAccess.Write** Open a file for writing.

IDispose and FileStream objects

```
using (FileStream outputStream = new FileStream("OutputText.txt", FileMode.OpenOrCreate,
FileAccess.Write))
{
    string outputMessageString = "Hello world";
    byte[] outputMessageBytes = Encoding.UTF8.GetBytes(outputMessageString);
    outputStream.Write(outputMessageBytes, 0, outputMessageBytes.Length);
}
```

Work with text files

> 读取**text file**的**short cut method.**

```
using (StreamWriter writeStream = new StreamWriter("OutputText.txt"))
{
    writeStream.Write("Hello world");
}

using (StreamReader readStream = new StreamReader("OutputText.txt"))
{
    string readSTring = readStream.ReadToEnd();
    Console.WriteLine("Text read: {0}", readSTring);
}
```

Chain streams together

> **The Stream class has a constructor that will accept another stream as a parameter, allowing the creation of chains of streams.**

> e.g.保存,读取压缩文件

```
using (FileStream writeFile = new FileStream("CompText.zip", FileMode.OpenOrCreate,
 FileAccess.Write))
{
    using (GZipStream writeFileZip = new GZipStream(writeFile,
    CompressionMode.Compress))
    {
        using (StreamWriter writeFileText = new StreamWriter(writeFileZip))
        {
            writeFileText.Write("Hello world");
        }
    }
}

using (FileStream readFile = new FileStream("CompText.zip", FileMode.Open,
 FileAccess.Read))
{
    using (GZipStream readFileZip = new GZipStream(readFile,
    CompressionMode.Decompress))
    {
        using (StreamReader readFileText = new StreamReader(readFileZip))
        {
            string message = readFileText.ReadToEnd();
            Console.WriteLine("Read text: {0}", message);
        }
    }
}
```

File class

一个Helper class,包含静态方法copy a file, create a file, delete a file, move a file, open a file, read a file, and manage file security.

```
File.WriteAllText(path: "TextFile.txt",   contents: "This text goes in the file");

File.AppendAllText(path: "TextFile.txt",   contents: " - This goes on the end");

if (File.Exists("TextFile.txt"))
    Console.WriteLine("Text File exists");

string contents = File.ReadAllText(path:"TextFile.txt");
Console.WriteLine("File contents: {0}", contents);
File.Copy(sourceFileName: "TextFile.txt",  destFileName: "CopyTextFile.txt");

using (TextReader reader = File.OpenText(path: "CopyTextFile.txt"))
{
    string text = reader.ReadToEnd();
    Console.WriteLine("Copied text: {0}", text);
}
```

Handle stream exceptions

```
try
{
    string contents = File.ReadAllText(path: "Testfile.txt");
    Console.WriteLine(contents);
}
catch(FileNotFoundException notFoundEx)
{
    // File not found
    Console.WriteLine(notFoundEx.Message);
}
catch(Exception ex)
{
    // Any other exception
    Console.WriteLine(ex.Message);
}
```

Files storage

- DriveInfo :获取磁盘信息

```
DriveInfo[] drives = DriveInfo.GetDrives();
foreach (DriveInfo drive in drives)
{
    Console.Write("Name:{0} ", drive.Name);
    if (drive.IsReady)
    {
        Console.Write("  Type:{0}", drive.DriveType);
        Console.Write("  Format:{0}", drive.DriveFormat);
        Console.Write("  Free space:{0}", drive.TotalFreeSpace);
    }
    else
    {
        Console.Write("  Drive not ready");
    }
    Console.WriteLine();
}
```

输出示例:

```
Name:C:\   Type:Fixed  Format:NTFS  Free space:69709230080
Name:D:\   Type:Fixed  Format:NTFS  Free space:133386022912
Name:E:\   Drive not ready
Name:F:\   Type:Removable  Format:exFAT  Free space:419379281
Name:G:\   Type:Fixed  Format:NTFS  Free space:44650496
Name:K:\   Drive not ready
Name:L:\   Drive not ready
```

- **FileInfo class**

> 可以用FileInfo 类打开文件，读取写入移动重命名等，有些功能和File类重复，File类一般用来操作单个文件，FileInfo操作多个文件。

tributes to a file. The available attributes are as follows:

- **FileAttributes.Archive** The file has not been backed up yet. The attribute will be cleared when/if the file is backed up.
- **FileAttributes.Compressed** The file is compressed. This is not something that our program should change.
- **FileAttributes.**Directory The file is a directory. This is not something our program should change.
- **FileAttributes.Hidden** The file will not appear in an ordinary directory listing.
- **FileAttributes.Normal** This is a normal file with no special attributes. This attribute is only valid when there are no other attributes assigned to the file.
- **FileAttributes.ReadOnly** The file cannot be written.
- **FileAttributes.System** The file is part of the operating system and is used by it.
- **FileAttributes.Temporary** The file is a temporary file that will not be required when the application has finished. The file system will attempt to keep this file in memory to improve performance.

使用:

```
string filePath = "TextFile.txt";

File.WriteAllText(path: filePath, contents: "This text goes in the file");
FileInfo info = new FileInfo(filePath);
Console.WriteLine("Name: {0}", info.Name);
Console.WriteLine("Full Path: {0}", info.FullName);
Console.WriteLine("Last Access: {0}", info.LastAccessTime);
Console.WriteLine("Length: {0}", info.Length);
Console.WriteLine("Attributes: {0}", info.Attributes);
Console.WriteLine("Make the file read only");
info.Attributes |= FileAttributes.ReadOnly;
Console.WriteLine("Attributes: {0}", info.Attributes);
Console.WriteLine("Remove the read only attribute");
info.Attributes &= ~FileAttributes.ReadOnly;
Console.WriteLine("Attributes: {0}", info.Attributes);
```

- 用FileInfo类GetFile(string searchPattern)搜索文件

> searchPattern可以接受*号代表个数字符,?号代表任意单个字符, 如下使用递归查找目录里文件

```
static void FindFiles(DirectoryInfo dir, string searchPattern)
{
    foreach (DirectoryInfo directory in dir.GetDirectories())
    {
        FindFiles(directory, searchPattern);
    }

    FileInfo[] matchingFiles = dir.GetFiles(searchPattern);
    foreach(FileInfo fileInfo in matchingFiles)
    {
        Console.WriteLine(fileInfo.FullName);
    }
}
static void Main(string[] args)
{
    DirectoryInfo startDir = new DirectoryInfo(@"..\..\..\..");
    string searchString = "*.cs";

    FindFiles(startDir, searchString);

    Console.ReadKey();
}
```

- **Directory and DirectoryInfo classes**

> **As with files, there are two ways to work with directories: the Directory class and the DirectoryInfo class.**

> The **Directory** class is like the File class. It is a static class that provides methods that can enumerate the contents of directories and create and manipulate directories.

```
Directory.CreateDirectory("TestDir");

if (Directory.Exists("TestDir"))
    Console.WriteLine("Directory created successfully");

Directory.Delete("TestDir");

Console.WriteLine("Directory deleted successfully");
```

> An instance of the **DirectoryInfo** class describes the contents of one directory. The class also provides methods that can be used to cerate and manipulate directories. 下面的代码和上面的功能一样。

```
DirectoryInfo localDir = new DirectoryInfo("TestDir");

localDir.Create();
if(localDir.Exists)
    Console.WriteLine("Directory created successfully");

localDir.Delete();

Console.WriteLine("Directory deleted successfully");
```

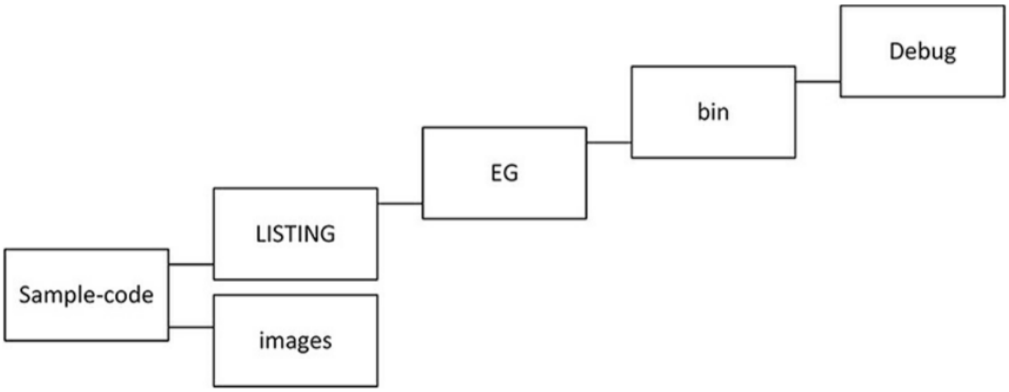
Files paths

- **relative path**

> A relative path specifies the location of a file relative to the folder in which the program is presently running.
>
> **When expressing paths, the character "." (period) has a special significance. A single period "." means the current directory. A double period ".." means the directory above the present one.**The **@** character at the start of the string literal marks the string as a verbatim string. This means that any escape characters in the string will be ignored (backslash).

```
string imagePath =  @"..\..\..\..\images");
```

程序运行的时候再Debug文件夹下，上面的代码路径如下



- **Absolute Path**

```
string absPath = @"c:\users\rob\Documents\test.txt"
```

- Path 类

> 可以用来执行如下操作

```
string fullName = @"c:\users\rob\Documents\test.txt";

string dirName = Path.GetDirectoryName(fullName);
string fileName = Path.GetFileName(fullName);
string fileExtension = Path.GetExtension(fullName);
string lisName = Path.ChangeExtension(fullName, ".lis");
string newTest = Path.Combine(dirName, "newtest.txt");

Console.WriteLine("Full name: {0}", fullName);
Console.WriteLine("File directory: {0}", dirName);
Console.WriteLine("File name: {0}", fileName);
Console.WriteLine("File extension: {0}", fileExtension);
Console.WriteLine("File with lis extension: {0}", lisName);
Console.WriteLine("New test: {0}", newTest);
```

可能的输出结果

```
Full name: c:\users\rob\Documents\test.txt
File directory: c:\users\rob\Documents
File name: test.txt
File extension: .txt
File with lis extension: c:\users\rob\Documents\test.lis
New test: c:\users\rob\Documents\newtest.txt
```

Skill 4.2: Consume data

Query a Database

e.g.

make a connection to a database, create an SQL query, and then execute this query on the database to read and print out information from the MusicTrack table.

```
using System;
using System.Data.SqlClient;

namespace LISTING_4_19_Read_with_SQL
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = "Server=(localdb)\\mssqllocaldb;" +
                "Database=MusicTracksContext-e0f0cd0d-38fe-44a4-add2-359310ff8b5d;" +
                "Trusted_Connection=True;MultipleActiveResultSets=true";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                SqlCommand command = new SqlCommand("SELECT * FROM MusicTrack",
                connection);

                SqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
                    string artist = reader["Artist"].ToString();
                    string title = reader["Title"].ToString();

                    Console.WriteLine("Artist: {0} Title: {1}", artist, title);
                }
            }
            Console.ReadKey();
        }
    }
}
```

Update data in a database

e.g.

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    dumpDatabase(connection);
    SqlCommand command = new SqlCommand(
       "UPDATE MusicTrack SET Artist='Robert Miles' WHERE ID='1'", connection);
    int result = command.ExecuteNonQuery();
    Console.WriteLine("Number of entries updated: {0}", result);
    dumpDatabase(connection);
}
```

SQL injection

you should never construct SQL commands directly from user input. You must use parameterized SQL statements instead.

```
string SqlCommandText = "UPDATE MusicTrack SET Artist=@artist WHERE Title=@searchTitle";

SqlCommand command = new SqlCommand(SqlCommandText, connection);
command.Parameters.AddWithValue("@artist", newArtist);
command.Parameters.AddWithValue("@searchTitle", searchTitle);
```

Asynchronous database access(perfered)

```
async Task<string> dumpDatabase(SqlConnection connection)
{
    SqlCommand command = new SqlCommand("SELECT * FROM MusicTrack", connection);
    SqlDataReader reader = await command.ExecuteReaderAsync();
    StringBuilder databaseList = new StringBuilder();
    while (await reader.ReadAsync())
    {
        string id = reader["ID"].ToString();
        string artist = reader["Artist"].ToString();
        string title = reader["Title"].ToString();

        string row = string.Format("ID: {0} Artist: {1} Title: {2}", id, artist, title);
        databaseList.AppendLine(row);
    }
    return databaseList.ToString();
}
```

Skill 4.3: LINQ

- **Basic**

```
IEnumerable<MusicTrack> selectedTracks = from track in musicTracks where track.Artist.
Name == "Rob Miles" select track;

foreach (MusicTrack track in selectedTracks)
{
    Console.WriteLine("Artist:{0} Title:{1}", track.Artist.Name, track.Title);
}
```

- **LINQ projection(select operation)**

```
var selectedTracks = from track in musicTracks
                     where track.Artist.Name == "Rob Miles"
                     select new TrackDetails
                     {
                         ArtistName = track.Artist.Name,
                         Title = track.Title
                     };
```

- **Anonymous types**

```
var selectedTracks = from track in musicTracks
                     where track.Artist.Name == "Rob Miles"
                     select new // projection type name missing from here
                     {
                         ArtistName = track.Artist.Name,
                         track.Title
                     };
```

- **LINQ join**

  e.g.

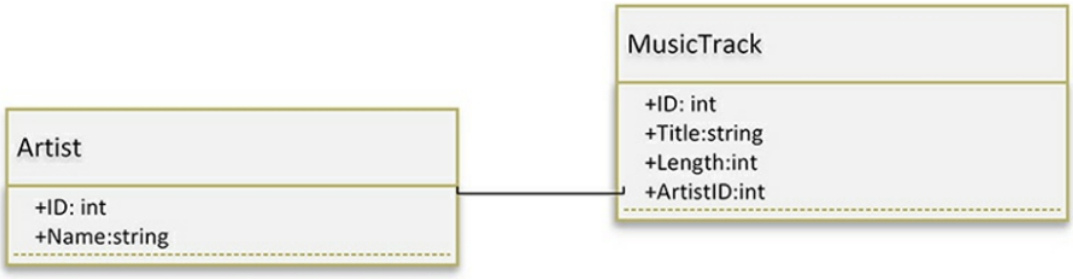


FIGURE 4-16 Music tracks and Artist ID

```
var artistTracks = from artist in artists where artist.Name == "Rob Miles"
                   join track in musicTracks on artist.ID equals track.ArtistID
                   select new
{
    ArtistName = artist.Name,
    track.Title
};
```

> **The first query selects the artist with the name "Rob Miles." The results of that query are joined to the second query that searches the musicTracks collection for tracks with an ArtistID property that matches that of the artist found by the first query.**

- **LINQ group**

> **group the results of a query to create a summary output. For example, you may want to create a query to tell how many tracks there are by each artist in the music collection.**

```
var artistSummary = from track in musicTracks
                    group track by track.ArtistID
                    into artistTrackSummary
                    select new
                    {
                        ID = artistTrackSummary.Key,
                        Count = artistTrackSummary.Count()
                    };
```

```
foreach (var item in artistSummary)
{
    Console.WriteLine("Artist:{0} Tracks recorded:{1}",
        item.ID, item.Count);
}
```

输出:

```
Artist:0 Tracks recorded:5
Artist:6 Tracks recorded:5
Artist:12 Tracks recorded:5
Artist:18 Tracks recorded:5
```

再用join

```
var artistSummaryName = from track in musicTracks
                join artist in artists on track.ArtistID equals artist.ID
                group track by artist.Name
                into artistTrackSummary
                select new
                {
                    ID = artistTrackSummary.Key,
                    Count = artistTrackSummary.Count()
                };
```

The output from this query is shown here:

```
Artist:Rob Miles Tracks recorded:5
Artist:Fred Bloggs Tracks recorded:5
Artist:The Bloggs Singers Tracks recorded:5
Artist:Immy Brown Tracks recorded:5
```

- **LINQ Take and skip**

```
int pageNo = 0;
int pageSize = 10;

while(true)
{
    // Get the track information
    var trackList = from musicTrack in musicTracks.Skip(pageNo*pageSize).Take(pageSize)
                    join artist in artists on musicTrack.ArtistID equals artist.ID
                    select new
                    {
                        ArtistName = artist.Name,
                        musicTrack.Title
                    };

    // Quit if we reached the end of the data
    if (trackList.Count() == 0)
        break;

    // Display the query result
    foreach (var item in trackList)
    {
        Console.WriteLine("Artist:{0} Title:{1}",
            item.ArtistName, item.Title);
    }

    Console.Write("Press any key to continue");
    Console.ReadKey();

    // move on to the next page
    pageNo++;
}
```

- **LINQ aggregate**

上面group中的Count()也是aggregate方法之一，其他例如Sum():

```
var artistSummary = from track in musicTracks
           join artist in artists on track.ArtistID equals artist.ID
           group track by artist.Name
           into artistTrackSummary
           select new
           {
                   ID = artistTrackSummary.Key,
                   Length = artistTrackSummary.Sum(x => x.Length)
           };
```

x是group中的单个item.

```
Name:Rob Miles  Total length:1406
Name:Fred Bloggs  Total length:1533
Name:The Bloggs Singers  Total length:1413
Name:Immy Brown  Total length:1813
```

- **Select data by using anonymous types**

> **sample programs have shown the use of anonymous types moving from creating values that summarize the contents of a source data object (for example extracting just the artist and title information from a MusicTrack value), to creating completely new types that contain data from the database and the results of aggregate operators.**

```
var artistSummary = MusicTracks.Join(
                        Artists,
                        track => track.ArtistID,
                        artist => artist.ID,
                        (track, artist) =>
                            new
                            {
                                track = track,
                                artist = artist
                            }
                    )
                    .GroupBy(
                        temp0 => temp0.artist.Name,
                        temp0 => temp0.track
                    )
                    .Select(
                        artistTrackSummary =>
                            new
                            {
                                ID = artistTrackSummary.Key,
                                Length = artistTrackSummary.Sum(x => x.Length)
                            }
                    );
```

- **method-based LINQ queries**

```
IEnumerable<MusicTrack> selectedTracks =
     from track in musicTracks where track.Artist.Name == "Rob Miles" select track;
```

相当于

```
IEnumerable<MusicTrack> selectedTracks =
        musicTracks.Where(track => track.Artist.Name == "Rob Miles");
```

Force execution of a query

> The actual evaluation of a LINQ query normally only takes place when a program starts to extract results from the query. This is called deferred execution. If you want to force the execution of a query you can use the ToArray() method

```
var artistTracksQuery = from artist in artists
                    where artist.Name == "Rob Miles"
                    join track in musicTracks on artist.ID equals track.ArtistID
                    select new
                    {
                        ArtistName = artist.Name,
                        track.Title
                    };

var artistTrackResult = artistTracksQuery.ToArray();

foreach (var item in artistTrackResult)
{
    Console.WriteLine("Artist:{0} Title:{1}",
        item.ArtistName, item.Title);
}
```

Note that in the case of this example the result will be an array of anonymous class instances.

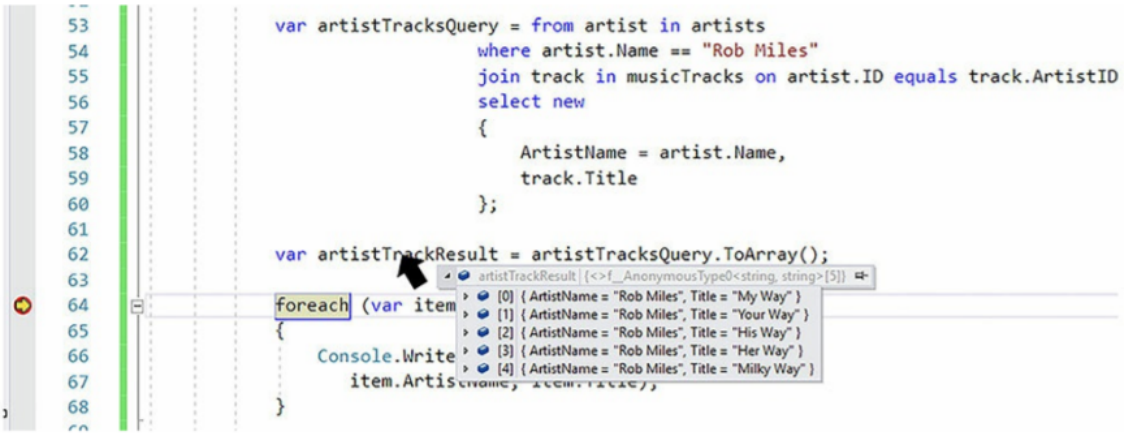


FIGURE 4-17 Immediate query results