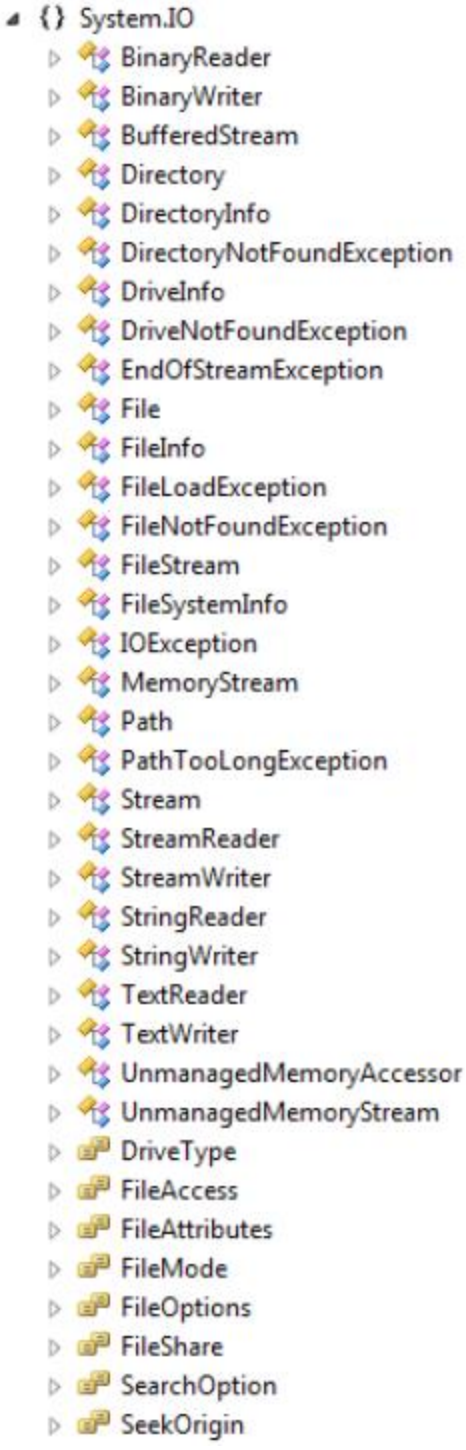
Программирование Ввода –Вывода



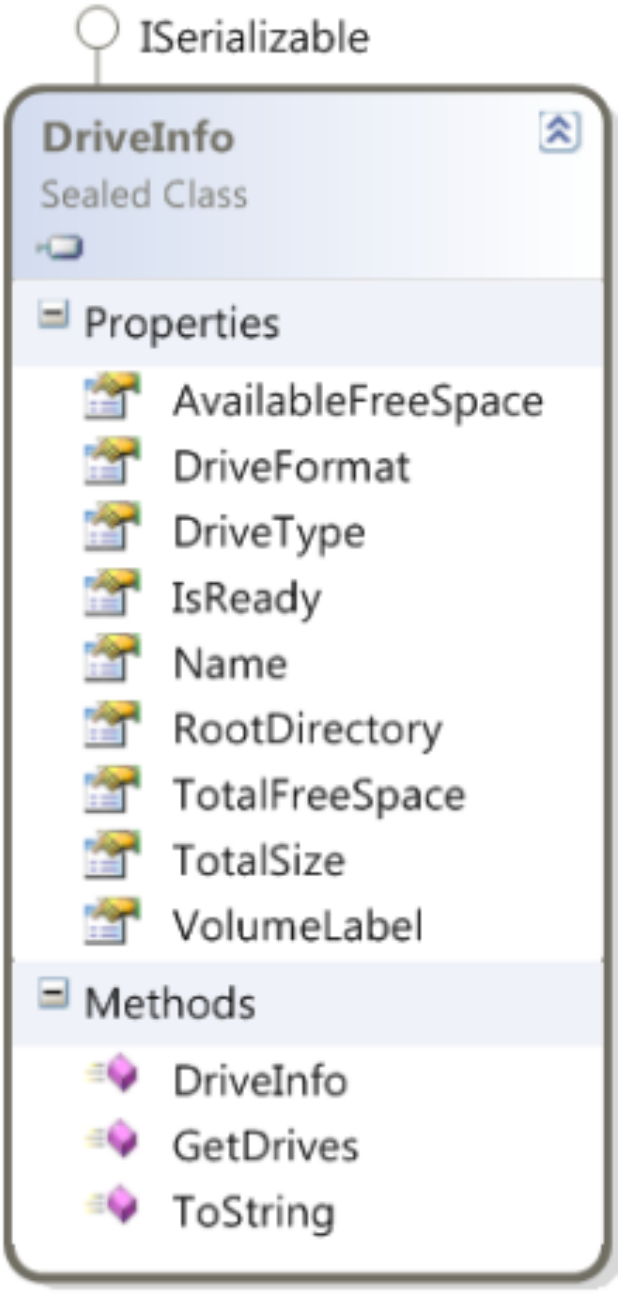
Пространство имен

`System.IO` – пространство имен, в котором определены классы для работы с файловой системой

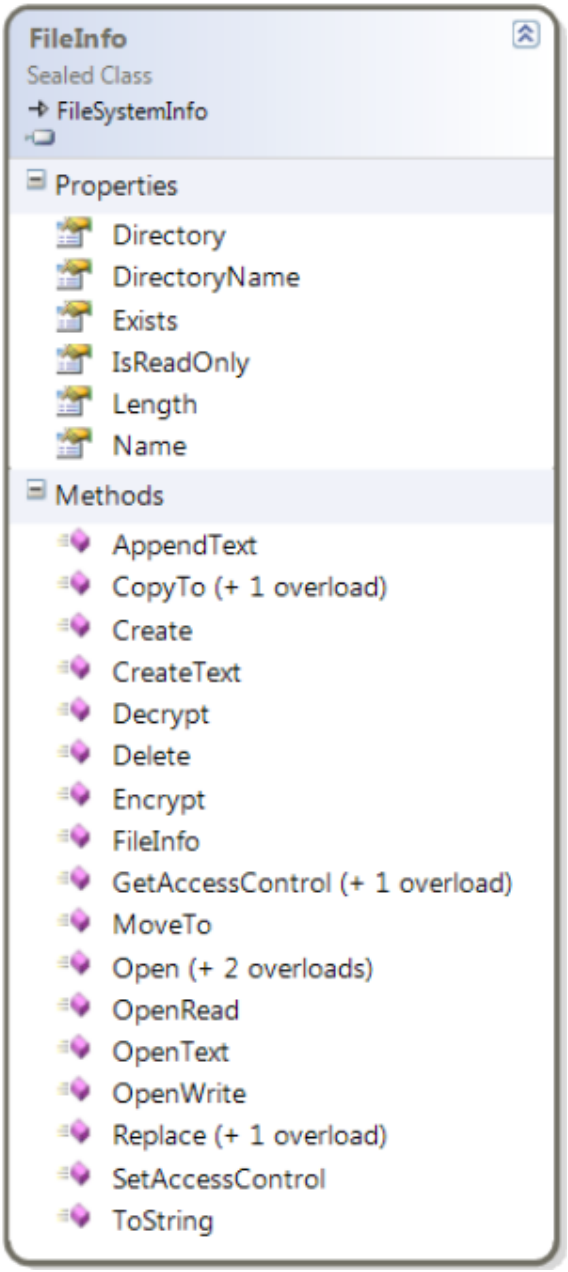
Получение информации

.NET Framework предоставляет доступ к информации о дисках, файлах и директориях посредством методов и свойств классов

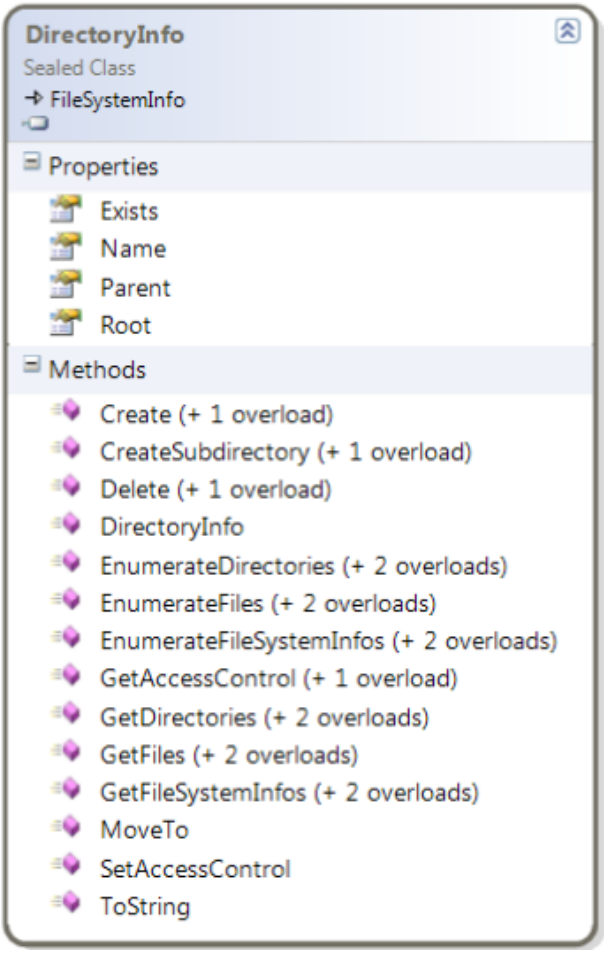
- `DriveInfo`
- `FileInfo`
- `DirectoryInfo`
- `FileSystemWatcher`
- `Path`



Класс `DriveInfo` предоставляет информацию о системных дисках и их состоянии. Является `sealed` и не может выступать в роли базового

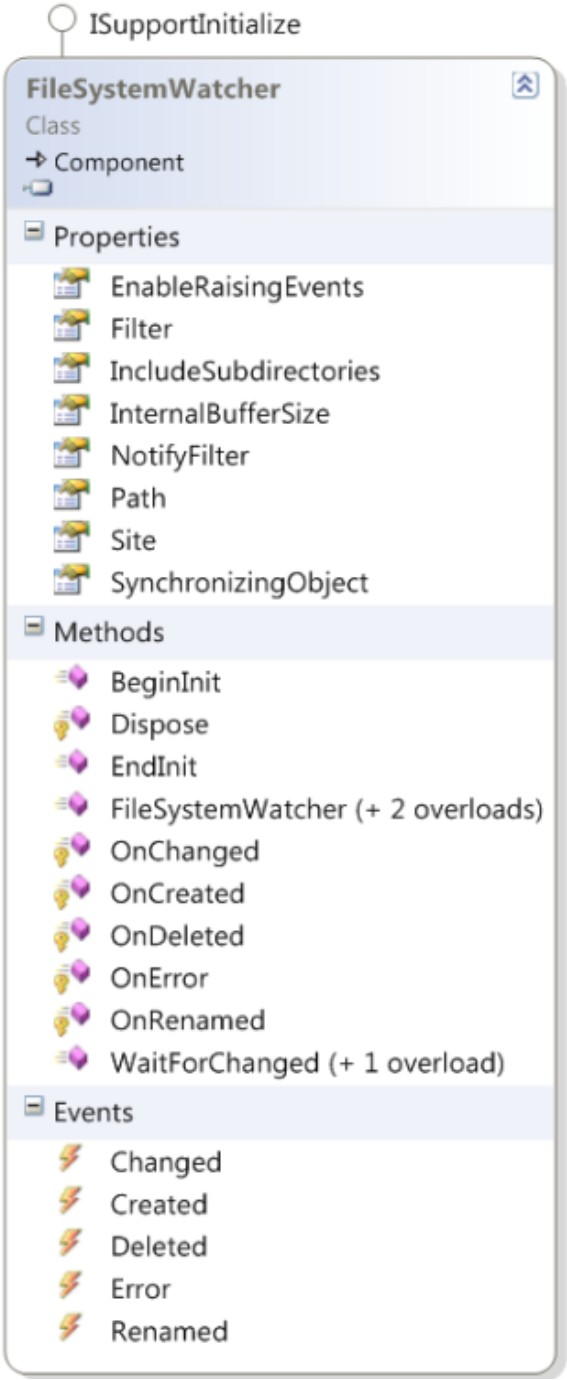


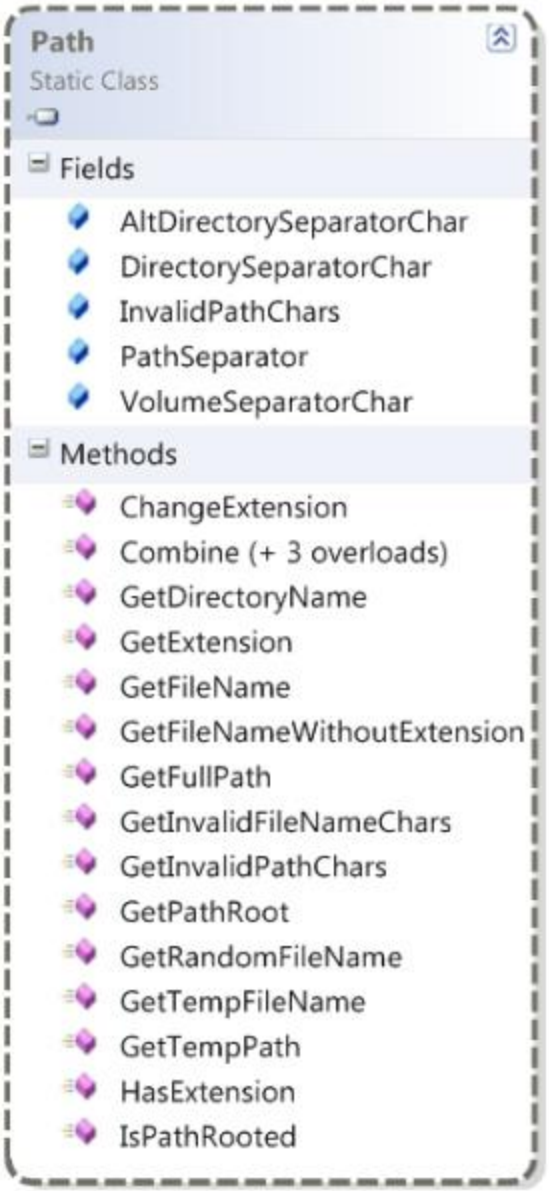
Класс `FileInfo` предназначен для работы с файлами. В нем определены методы для создания, удаления, копирования, перемещения и открытия файлов. Является `sealed` и не может выступать в роли базового



Класс `DirectoryInfo` содержит методы для создания, перемещения и перечисления директорийи / или поддиректорий компьютера. Является `sealed` и не может выступать в роли базового

Класс `FileSystemWatcher` специализируется на отслеживании изменений в системе. Вызывает соответствующие события при создании, удалении или изменении файлов и директорий





Статический класс `Path` предназначен для работы с путями файловой системы. Все операции класса выполняются в кросс-платформенной манере

Виды потоков

Байтовые потоки:

- `BufferedStream`
- `FileStream`
- `MemoryStream`
- `UnmanagedMemoryStream`

Символьные потоки:

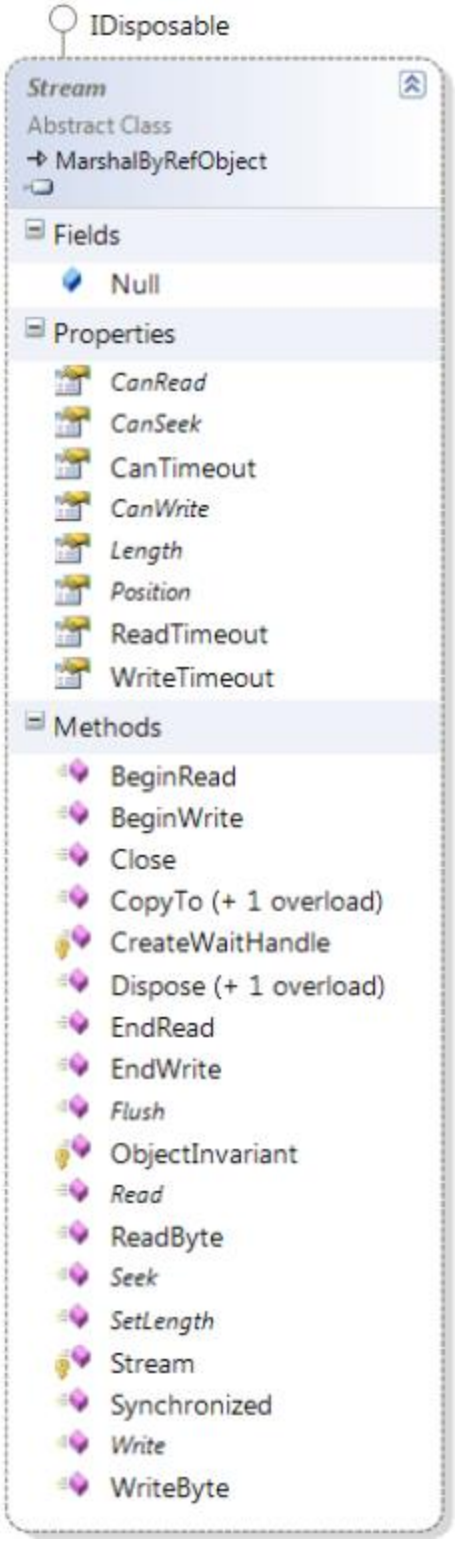
- `StreamReader`
- `StreamWriter`
- `StringReader`
- `StringWriter`

Двоичные потоки:

- `BinaryReader`
- `BinaryWriter`

Абстрактный класс

Абстрактный класс `Stream`, является базовым для всех классов потоков



Байтовые

`BufferedStream` – заключает байтовый поток в оболочку и добавляет буферизацию.

`FileStream` – поток для осуществления файлового ввода/вывода.

`MemoryStream` – поток для работы с памятью.

`UnmanagedMemoryStream` – поток для работы с неуправляемой памятью

Символьные

На вершине иерархии классов символьных потоков находятся абстрактные классы `TextReader` и `TextWriter`.

`StreamReader` / `StreamWriter` – предназначены для ввода/вывода символов из байтового потока, являются оболочкой для потоков байтового ввода/вывода.

`StringReader` / `StringWriter` – предназначены для ввода/вывода символов в символьную строку

Двоичные

Используются для файлового ввода/вывода данных имеющих не строковой тип.

`BinaryReader` / `BinaryWriter` - представляют собой оболочку для байтовых потоков.

Позволяют сохранять информацию непосредственно в двоичном формате, что исключает необходимость предварительного преобразования данных

```
struct State
{
    public string name;
    public string capital;
    public int area;
    public double people;

    public State(string n, string c, int a, double p)
    {
        name = n;
        capital = c;
        people = p;
        area = a;
    }
}
class Program
{
    static void Main(string[] args)
    {
        State[] states = new State[2];
        states[0] = new State("Германия", "Берлин",  357168,  80.8);
        states[1] = new State("Франция", "Париж", 640679, 64.7);

        string path= @"C:\SomeDir\states.dat";

        try
        {
            // создаем объект BinaryWriter
            using (BinaryWriter writer = new BinaryWriter(File.Open(path,
FileMode.OpenOrCreate)))
            {
                // записываем в файл значение каждого поля структуры
                foreach (State s in states)
                {
                    writer.Write(s.name);
                    writer.Write(s.capital);
                    writer.Write(s.area);
                    writer.Write(s.people);
                }
            }
            // создаем объект BinaryReader
            using (BinaryReader reader = new BinaryReader(File.Open(path,
FileMode.Open)))
            {
                // пока не достигнут конец файла
                // считываем каждое значение из файла
                while (reader.PeekChar() > -1)
                {
                    string name = reader.ReadString();
                    string capital = reader.ReadString();
                    int area = reader.ReadInt32();
                    double population = reader.ReadDouble();
```

```
                    Console.WriteLine("Страна: {0}  столица: {1}  площадь {2}
кв. км   численность населения: {3} млн. чел.",
                        name, capital, area, population);
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        Console.ReadLine();
    }
}
```

Compression

`System.IO.Compression` – пространство имен, в котором определены классы для работы со сжатием данных

GZipStream

`GZipStream` реализует стандартный алгоритм сжатия без потерь, включает циклический контроль суммы для обнаружения поврежденных данных

DeflateStream

`DeflateStream` выполняет сжатие/декомпрессию данных по алгоритму Лемпеля-Зива LZ77 с использованием кодирования по методу Хаффмана

IsolatedStorage

Изолированное хранилище рекомендуется использовать для:

- Промежуточного хранения и загрузки элементов управления

•Хранения общих компонентов

•Хранения личных данных пользователей на сервере

•Перемещения личных данных пользователей

Изолированное хранилище не рекомендуется использовать для:

•Хранения конфиденциальных и важных данных

•Хранения кода

•Хранения данных развертывания