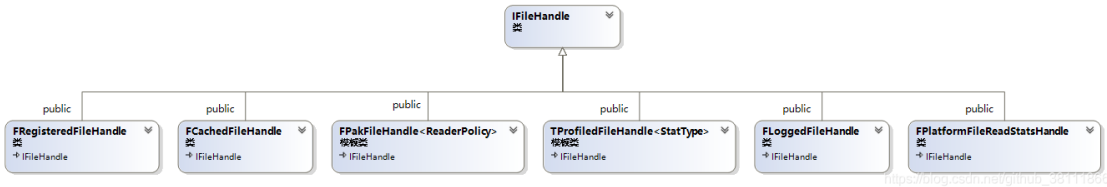
IFileManager

> 通过该类的实例对象，可以获取文件或目录的信息。比如目录、文件是否存在，创建或删除目录或文件，拷贝或移动文件，获取目录下的所有目录或文件等功能。



通过单例方式获取IFileManager对象：

```
static IFileManager& Get();
```

拷贝文件：

```
uint32 Copy(
    const TCHAR* Dest,                                  //目标路径
    const TCHAR* Src,                                   //文件路径
    bool Replace=1,                                     //是否替换
    bool EvenIfReadOnly=0,                              //是否只读
    bool Attributes=0,
    FCopyProgress* Progress = nullptr,
    EFileRead ReadFlags=FILEREAD_None,
    EFileWrite WriteFlags=FILEWRITE_None
    );
```

还有更多其它操作，此处不再赘述，具体见源码。

FFileHelper

> 该类可以加载文件到内存，读取文件内容到程序中并赋值到字符串等类型变量，并且可以保存字符串等内容到文件。该类的方法都为静态方法，可以直接通过类调用成员方法。

把ANSI/Unicode编码的字符数组转换为字符串：

```
static void BufferToString( FString& Result, const uint8* Buffer, int32 Size );
```

加载文件并赋值到字符数组：

```
static bool LoadFileToArray( TArray<uint8>& Result, const TCHAR* Filename,
uint32 Flags = 0 );
```

加载文件并赋值到字符串：

```
static bool LoadFileToString( FString& Result, const TCHAR* Filename,
EHashOptions VerifyFlags = EHashOptions::None );
```

加载文件并赋值到字符串数组，每行文本保存为一个字符串：

```
static bool LoadFileToStringArray( TArray<FString>& Result, const TCHAR*
Filename, EHashOptions VerifyFlags = EHashOptions::None );
```

保存字符数组到文件：

```
static bool SaveArrayToFile(TArrayView<const uint8> Array, const TCHAR*
Filename, IFileManager* FileManager=&IFileManager::Get(), uint32 WriteFlags =
0);
```

保存字符串到文件：

```
static bool SaveStringToFile( const FString& String, const TCHAR* Filename,
EEncodingOptions EncodingOptions = EEncodingOptions::AutoDetect, IFileManager*
FileManager = &IFileManager::Get(), uint32 WriteFlags = 0 );
```

保存字符串数组到文件：

```
static bool SaveStringArrayToFile( const TArray<FString>& Lines, const TCHAR*
Filename, EEncodingOptions EncodingOptions = EEncodingOptions::AutoDetect,
IFileManager* FileManager = &IFileManager::Get(), uint32 WriteFlags = 0 );
```

创建和保存一个24/32位位图文件：

```
static bool CreateBitmap( const TCHAR* Pattern, int32 DataWidth, int32
DataHeight, const struct FColor* Data, struct FIntRect* SubRectangle = NULL,
IFileManager* FileManager = &IFileManager::Get(), FString* OutFilename = NULL,
bool bInWriteAlpha = false );
```

生成一个唯一位图文件名，并使用指定拓展名：

```
static bool GenerateNextBitmapFilename(const FString& Pattern, const FString&
Extension, FString& OutFilename, IFileManager* FileManager =
&IFileManager::Get());
```

将一个ANSI文本文件内容赋值到字符串数组，数组每个元素为一行文本：

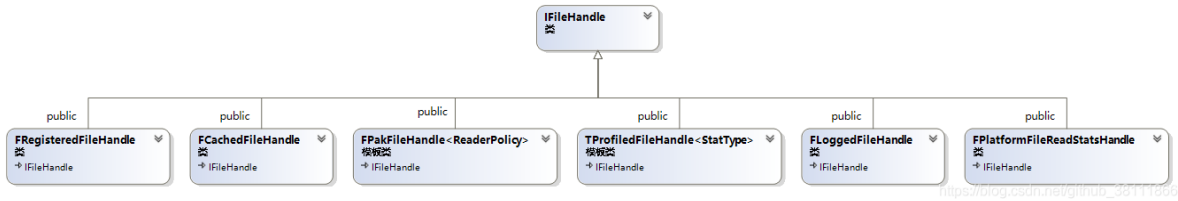
```
static bool LoadANSITextFileToStrings(const TCHAR* InFilename, IFileManager*
InFileManager, TArray<FString>& OutStrings);
```

检查文件名是否合法：

```
static bool IsFilenameValidForSaving(const FString& Filename, FText& OutError);
```

IPlatformFile

> 文件的IO接口



FPlatformFileManager

> FPlatformFileManager类包含一个IPlatformFile链表，该类的主要功能就是维护这个链表，如通过该类成员方法SetPlatformFile设置新的链表结点，该结点将被保存到TopmostPlatformFile变量，然后通过GetPlatformFile获取最近使用的IPlatformFile对象引用，即TopmostPlatformFile变量。该类作为单例使用，可以通过Get方法获取该类单例对象的引用。

获取该类的单例对象引用：

```
static FPlatformFileManager& Get();
```

获取最近使用的IPlatformFile对象，即TopmostPlatformFile：

```
IPlatformFile& GetPlatformFile();
```

添加并设置一个新的IPlatformFile对象到链表中：

```
void SetPlatformFile( IPlatformFile& NewTopmostPlatformFile );
```

通过PlatformFile的名称查找链表中是否存在该IPlatformFile对象并返回；若不存在，返回nullptr：

```
IPlatformFile* FindPlatformFile( const TCHAR* Name );
```

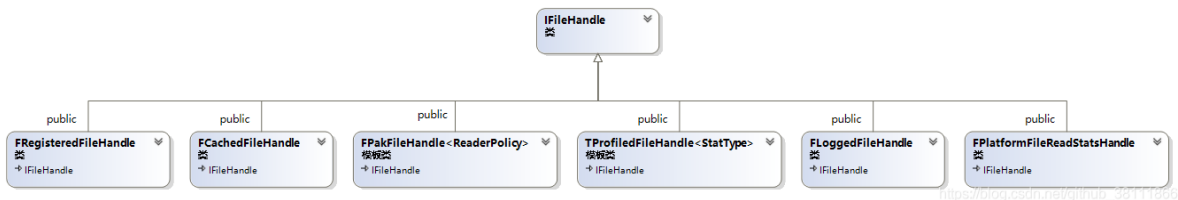
通过PlatformFile的名称创建一个IPlatformFile对象并返回：

```
IPlatformFile* GetPlatformFile( const TCHAR* Name );
```

执行一遍链表中所有IPlatformFile对象的Tick函数：

```
void TickActivePlatformFile();
```

IFileHandle



FPaths

> 当进行文件操作时，通常需要进行路径获取与处理。FPaths这个类就是用来辅助获取相关路径和路径处理的类。通过此类，可以获取项目目录、引擎目录、项目和引擎的插件、Content、Saved等目录路径，还可以进行相对路径和绝对路径的转换，获取通过文件路径获取文件类型等操作。这些操作方法通常通过静态函数提供使用。
