Содержание

[1 Краткая характеристика программы 4](#_Toc177920540)

[2 Обращение к программе 8](#_Toc177920541)

[2.1 Подключение библиотеки 8](#_Toc177920542)

[2.1.1 Заголовочный файл «DataAccessLib.h» 8](#_Toc177920543)

[2.1.2 Файл исходного кода «DataAccessLib.cpp» 9](#_Toc177920544)

[2.2 Операции с хранилищами 12](#_Toc177920545)

[2.3 Операции с файловыми системами, файлами и каталогами 16](#_Toc177920546)

[3 Входные и выходные данные 22](#_Toc177920547)

[3.1 Основные типы, псевдонимы и константы 22](#_Toc177920548)

[3.1.1 StorageType 22](#_Toc177920549)

[3.1.2 FileSystemTypeEnum 22](#_Toc177920550)

[3.1.3 OriginEnum 22](#_Toc177920551)

[3.1.4 PhysicalDriveInfoStruct 22](#_Toc177920552)

[3.1.5 PartitionInfoStruct 23](#_Toc177920553)

[3.1.6 Псевдонимы 24](#_Toc177920554)

[3.1.7 Константы 24](#_Toc177920555)

[3.2 Операции с хранилищами 25](#_Toc177920556)

[3.2.1 CreateStorageHandle 25](#_Toc177920557)

[3.2.2 GetStorageType 25](#_Toc177920558)

[3.2.3 GetPhysicalDriveInfo 26](#_Toc177920559)

[3.2.4 GetNumberOfPartitions 27](#_Toc177920560)

[3.2.5 GetPartitionInfoByNumber 27](#_Toc177920561)

[3.2.6 GetStorageBlockSize 28](#_Toc177920562)

[3.2.7 SetStorageBlockSize 29](#_Toc177920563)

[3.2.8 GetStorageIterator 30](#_Toc177920564)

[3.2.9 StorageBlockFirst 30](#_Toc177920565)

[3.2.10 StorageBlockNext 31](#_Toc177920566)

[3.2.11 StorageBlockIsDone 31](#_Toc177920567)

[3.2.12 CloseStorageIterator 32](#_Toc177920568)

[3.2.13 SetStoragePosition 32](#_Toc177920569)

[3.2.14 ReadStorageData 33](#_Toc177920570)

[3.2.15 RecognizeFileSystem 34](#_Toc177920571)

[3.2.16 CloseStorageHandle 35](#_Toc177920572)

[3.3 Операции с файловой системой 35](#_Toc177920573)

[3.3.1 CreateFileSystemHandle 35](#_Toc177920574)

[3.3.2 FastCreateFileSystemHandle 36](#_Toc177920575)

[3.3.3 GetFileSystemType 37](#_Toc177920576)

[3.3.4 GetFileSystemTypeString, GetFileSystemTypeStringW 37](#_Toc177920577)

[3.3.5 GetNumberOfClusters 38](#_Toc177920578)

[3.3.6 GetFileSystemSize 39](#_Toc177920579)

[3.3.7 GetFileSystemClusterSize 39](#_Toc177920580)

[3.3.8 GetClusterIterator 40](#_Toc177920581)

[3.3.9 ClusterFirst 40](#_Toc177920582)

[3.3.10 ClusterNext 41](#_Toc177920583)

[3.3.11 ClusterIsDone 41](#_Toc177920584)

[3.3.12 ClusterIsUsed 42](#_Toc177920585)

[3.3.13 GetCurrentCluster 42](#_Toc177920586)

[3.3.14 CloseClusterIterator 43](#_Toc177920587)

[3.3.15 SetFileSystemPosition 44](#_Toc177920588)

[3.3.16 ReadFileSystemData 44](#_Toc177920589)

[3.3.17 ReadClustersByNumber 45](#_Toc177920590)

[3.3.18 CloseFileSystemHandle 46](#_Toc177920591)

[3.4 Операции с отдельными файлами и каталогами 47](#_Toc177920592)

[3.4.1 CreateFileHandle 47](#_Toc177920593)

[3.4.2 CreateFileHandleByRecordId 47](#_Toc177920594)

[3.4.3 FileIsDir 48](#_Toc177920595)

[3.4.4 GetNumberOfStreams 49](#_Toc177920596)

[3.4.5 GetNumberOfFiles 49](#_Toc177920597)

[3.4.6 GetFileDataSize 50](#_Toc177920598)

[3.4.7 GetTimeFile[Create/Modify/Access/RecordModify] 51](#_Toc177920599)

[3.4.8 GetFileStdAttributes 52](#_Toc177920600)

[3.4.9 GetFileHash 53](#_Toc177920601)

[3.4.10 SaveFile 54](#_Toc177920602)

[3.4.11 SaveFileWithPath 55](#_Toc177920603)

[3.4.12 SetFilePosition 56](#_Toc177920604)

[3.4.13 ReadFileData 57](#_Toc177920605)

[3.4.14 CloseFileHandle 57](#_Toc177920606)

[3.5 Последовательный обход каталогов 58](#_Toc177920607)

[3.5.1 CreateDirectoryIteratorHandle 58](#_Toc177920608)

[3.5.2 DirFirst 58](#_Toc177920609)

[3.5.3 DirNext 59](#_Toc177920610)

[3.5.4 DirIsDone 59](#_Toc177920611)

[3.5.5 DirFileIsDir 60](#_Toc177920612)

[3.5.6 GetFileNameW 60](#_Toc177920613)

[3.5.7 GetFilePathW 61](#_Toc177920614)

[3.5.8 GetFullPathW 62](#_Toc177920615)

[3.5.9 CloseDirectoryIterator 63](#_Toc177920616)

[3.6 Обработка ошибок 63](#_Toc177920617)

[3.6.1 HasError 63](#_Toc177920618)

[3.6.2 GetLastErrorString, GetLastErrorStringW 63](#_Toc177920619)

[3.6.3 PrintDebug 64](#_Toc177920620)

[4 Сообщения 65](#_Toc177920621)

[5 Глоссарий 66](#_Toc177920622)

1. Краткая характеристика программы

Для работы с данными в библиотеке *DataAccessLib* используются следующие уровни доступа к данным: уровень хранилища (англ. storage), уровень файловой системы (англ. filesystem) и уровень объекта файловой системы (англ. file object).

Функционирование библиотеки проверено на операционных системах семейства Microsoft Windows от Windows XP SP3 до Windows  10.

При описании функционала библиотеки *DataAccessLib* используются следующие понятия. Под *основной файловой системой* подразумевается файловая система, смонтированная для чтения и записи в операционной системе, из которой вызываются функции библиотеки. *Гостевой файловой системой* называется файловая система, которая расположена на анализируемом носителе информации. *Виртуальной файловой системой* называется имитируемая программными средствами библиотеки файловая система, используемая для доступа к содержимому произвольных каталогов.

Работа с хранилищами данных, файловыми системами, объектами файловой системы в библиотеке реализована при помощи механизма дескрипторов. Каждый дескриптор идентифицирует отдельный объект библиотеки. Жизненный цикл дескриптора состоит из этапов: создания, использования и закрытия. Создание дескриптора реализовано соответствующими функциями, в результате работы которых во внутренних переменных библиотеки формируется описание связанного с ним объекта. С использованием механизма дескрипторов решаются задачи получения характеристик объекта, чтения данных по смещению. Перебор однотипных объектов реализован при помощи дескрипторов последовательного обхода.

Для дескрипторов последовательного обхода объектов некоторого множества предполагается работа по следующим этапам. На первом этапе происходит создание дескриптора путем вызова функции вида GetIterator(…)с передачей в нее дескриптора перебираемого множества объектов (например, файла-каталога). После создания дескриптор последовательного обхода следует инициализировать функцией вида First(…) для перехода к первому элементу перебираемого множества. На втором этапе (обычно в цикле) путем вызова соответствующих функций с передачей в них дескриптора последовательного обхода происходит получение характеристик текущего объекта, на который настроен дескриптор. Переход к следующему объекту перебираемого множества выполняется вызовом функции вида Next(…). Определение факта окончания обхода всех элементов множества происходит при помощи функции вида IsDone(…). Перебор объектов осуществляется только в одном направлении, доступ к предыдущему объекту множества получить нельзя. В случае необходимости осуществления повторного обхода следует использовать функцию First(…). Окончание работы с дескриптором последовательного обхода происходит после вызова функции вида CloseIterator(…) Для каждого конкретного типа дескриптора последовательного обхода (кластеров, файлов и др.) используется индивидуальный набор функций.

Дескриптор хранилища (StorageHandle) идентифицирует источник данных для анализа. Библиотека позволяет создавать (п. 3.2.1) дескрипторы хранилища для физического диска, логического раздела, файла-образа или каталога основной файловой системы по их текстовому идентификатору.

Для дескриптора хранилища определены функции, которые позволяют получать описание соответствующих им физических устройств (п. 3.2.3) и логических разделов (п. 3.2.4–3.2.5). Кроме того, библиотека позволяет организовать чтение данных из хранилища путем последовательного считывания блоков установленного размера (п. 3.2.6–3.2.12) или по заданному в байтах смещению (п. 3.2.13–3.2.14). Доступ к данным хранилища организован в режиме «только чтение», без возможности их модификации.

Доступ к данным уровня файловых систем осуществляется при помощи дескриптора файловой системы (FileSystemHandle). Операции для работы со всеми поддерживаемыми типами файловых систем унифицированы.

Создание дескриптора файловой системы реализовано двумя способами. В первом способе (п. 3.3.1) дескриптор файловой системы создается с использованием дескриптора хранилища. Данный метод позволяет указать тип создаваемой файловой системы, ее смещение относительно начала хранилища, а также размер выделенной для нее на области. Большую часть этой информации можно получить в результате анализа таблицы разделов хранилища (п. 3.2.4–3.2.6). Во втором («быстром») способе (п. 3.3.2) дескриптор хранилища пользователем не создается, а все параметры, кроме полного пути к хранилищу и смещения файловой системы относительно его начала, определяются автоматически. Получение характеристик файловой системы реализовано отдельными функциями (п. 3.3.3–3.3.7).

Библиотекой поддерживается доступ к кластерам (блокам) файловой системы в режиме их последовательного обхода (п. 3.3.8–3.3.14) или по номеру кластера (п. 3.3.17). При последовательном обходе реализована функция определения, занят кластер данными объекта файловой системы или помечен как свободный (п. 3.3.12).

Получение характеристик файлов и каталогов, чтение данных из файлов осуществляется при помощи дескриптора файлового объекта. Создание дескриптора файлового объекта реализовано по полному пути в гостевой файловой системе (п. 3.4.1) или по числовому идентификатору (п. 3.4.2). Функции описания объектов файловой системы позволяют определить, является этот объект каталогом или файлом (п. 3.4.3), его временные отметки (п. 3.4.7–3.4.7), стандартные атрибуты (п. 3.4.8), количество потоков данных (п. 3.4.4) и размер каждого из них (п. 3.4.6). Также реализованы функции сохранения файла из гостевой файловой системы в основную (п. 3.4.11, 3.4.12).

Последовательный обход объектов файловой системы (файлов и каталогов) *на стороне пользователя* может быть реализован рекурсивно с использованием дескрипторов последовательного обхода каталогов (п. 3.5.1–3.5.9).

В библиотеке *DataAccessLib* реализованы функции для проведения «пользовательского» сигнатурного анализа на всех уровнях организации доступа к данным. «Пользовательский» сигнатурный анализ может применяться для выявления специальных видов хранилищ и их разделов (п. 3.2.16), определения форматов отдельных файлов (п. 3.4.9) и массивов данных (п. 3.6.5).

Для функционирования «пользовательского» сигнатурного анализа обязательной является конфигурация используемого множества сигнатур, которая происходит в два этапа: на первом этапе данные о сигнатурах загружаются из файла основной файловой системы (п. 3.6.1), на втором этапе выбираются сигнатуры, который будут использоваться при анализе (п. 3.6.2–3.6.4).

В библиотеке реализован унифицированный механизм получения данных об ошибках, возникающих во время выполнения программы. В случае появления такой ошибки во внутренней переменной библиотеки сохраняется ее текстовое описание, которые можно получить вызовами соответствующих функций (п. 3.7.1, 3.7.2).

1. Обращение к программе
   1. Подключение библиотеки

Подключение библиотеки к проекту выполняется динамически на этапе выполнения программы. Пример кода, который необходимо добавить в файл основного проекта для подключения функций работы с хранилищами, приведен ниже. Файлы, содержащие полный код, необходимый для подключения библиотеки к проекту на языке C/C++, поставляются вместе с библиотекой.

* + 1. Заголовочный файл «DataAccessLib.h»

**//-----------------------------------------**

**#ifndef DataAccessLibH**

**#define DataAccessLibH**

**//-----------------------------------------**

**typedef** StorageHandle **(**\_stdcall **\*** PCreateStorageHandle**)**

**(const** WCHAR **\***fileName**);**

**typedef** **void** **(**\_stdcall **\*** PCloseStorageHandle**)(**StorageHandle storageHandle**);**

**typedef** StorageType **(**\_stdcall **\*** PGetStorageType**)(**StorageHandle storageHandle**);**

**typedef** PhysicalDriveInfoStruct **(**\_stdcall **\*** PGetPhysicalDriveInfo**)**

**(**StorageHandle storageHandle**);**

**typedef** **int** **(**\_stdcall **\*** PGetNumberOfPartitions**)(**StorageHandle storageHandle**);**

**typedef** **bool** **(**\_stdcall **\*** PGetPartitionInfoByNumber**)**

**(**StorageHandle storageHandle**,unsigned** **int** selectedNumber**,**

PartitionInfoStruct **\***outPartitionInfo**);**

**typedef** DWORD **(**\_stdcall **\*** PGetStorageBlockSize**)(**StorageHandle storageHandle**);**

**typedef** **bool** **(**\_stdcall **\*** PSetStorageBlockSize**)**

**(**StorageHandle storageHandle**,** DWORD blockSize**);**

**typedef** StorageIteratorHandle **(**\_stdcall **\*** PGetStorageIterator**)**

**(**StorageHandle storageHandle**);**

**typedef** **void** **(**\_stdcall **\*** PStorageBlockFirst**)**

**(**StorageIteratorHandle storageIteratorHandle**);**

**typedef** **void** **(**\_stdcall **\*** PStorageBlockNext**)**

**(**StorageIteratorHandle storageIteratorHandle**);**

**typedef** **bool** **(**\_stdcall **\*** PStorageBlockIsDone**)**

**(**StorageIteratorHandle storageIteratorHandle**);**

**typedef** **void** **(**\_stdcall **\*** PCloseStorageIterator**)**

**(**StorageIteratorHandle storageIteratorHandle**);**

**typedef** **bool** **(**\_stdcall **\*** PSetStoragePosition**)**

**(**StorageHandle storageHandle**,**LONGLONG newPosition**,**OriginEnum newOrigin**);**

**typedef** LONGLONG **(**\_stdcall **\*** PReadStorageData**)**

**(**StorageHandle storageHandle**,**LONGLONG bytesToRead**,**BYTE **\***dataBuffer**);**

**typedef** FileSystemTypeEnum **(**\_stdcall **\*** PRecognizeFileSystem**)**

**(**StorageHandle storageHandle**,** ULONGLONG startOffset**);**

**//------------------------------------**

**bool** \_\_fastcall DataAccessLibInit**();**

**void** \_\_fastcall DataAccessLibCleanup**();**

**//------------------------------------**

**#endif**

* + 1. Файл исходного кода «DataAccessLib.cpp»

**//-----------------------------------------**

**#include "DataAccessLib.h"**

**//-----------------------------------------**

**// Инициализировать указатели на функции**

**//-----------------------------------------**

PCreateStorageHandle CreateStorageHandle **=** **NULL;**

PCloseStorageHandle CloseStorageHandle **=** **NULL;**

PGetStorageType GetStorageType **=** **NULL;**

PGetPhysicalDriveInfo GetPhysicalDriveInfo **=** **NULL;**

PGetNumberOfPartitions GetNumberOfPartitions **=** **NULL;**

PGetPartitionInfoByNumber GetPartitionInfoByNumber **=** **NULL;**

PGetStorageBlockSize GetStorageBlockSize **=** **NULL;**

PSetStorageBlockSize SetStorageBlockSize **=** **NULL;**

PGetStorageIterator GetStorageIterator **=** **NULL;**

PStorageBlockFirst StorageBlockFirst **=** **NULL;**

PStorageBlockNext StorageBlockNext **=** **NULL;**

PStorageBlockIsDone StorageBlockIsDone **=** **NULL;**

PCloseStorageIterator CloseStorageIterator **=** **NULL;**

PSetStoragePosition SetStoragePosition **=** **NULL;**

PReadStorageData ReadStorageData **=** **NULL;**

PRecognizeFileSystem RecognizeFileSystem **=** **NULL;**

**//-----------------------------------------**

HINSTANCE DataAccessLibDll **=** **NULL;**

**//-----------------------------------------**

**void** InitFunctionsFromLib**()**

**{**

CreateStorageHandle **=** **(**PCreateStorageHandle**)**

GetProcAddress**(**DataAccessLibDll**,** "CreateStorageHandle"**);**

CloseStorageHandle **=** **(**PCloseStorageHandle**)**

GetProcAddress**(**DataAccessLibDll**,** "CloseStorageHandle"**);**

GetStorageType **=** **(**PGetStorageType**)**

GetProcAddress**(**DataAccessLibDll**,** "GetStorageType"**);**

GetPhysicalDriveInfo **=** **(**PGetPhysicalDriveInfo**)**

GetProcAddress**(**DataAccessLibDll**,** "GetPhysicalDriveInfo"**);**

GetNumberOfPartitions **=** **(**PGetNumberOfPartitions**)**

GetProcAddress**(**DataAccessLibDll**,** "GetNumberOfPartitions"**);**

GetPartitionInfoByNumber **=** **(**PGetPartitionInfoByNumber**)**

GetProcAddress**(**DataAccessLibDll**,** "GetPartitionInfoByNumber"**);**

GetStorageBlockSize **=** **(**PGetStorageBlockSize**)**

GetProcAddress**(**DataAccessLibDll**,** "GetStorageBlockSize"**);**

SetStorageBlockSize **=** **(**PSetStorageBlockSize**)**

GetProcAddress**(**DataAccessLibDll**,** "SetStorageBlockSize"**);**

GetStorageIterator **=** **(**PGetStorageIterator**)**

GetProcAddress**(**DataAccessLibDll**,** "GetStorageIterator"**);**

StorageBlockFirst **=** **(**PStorageBlockFirst**)**

GetProcAddress**(**DataAccessLibDll**,** "StorageBlockFirst"**);**

StorageBlockNext **=** **(**PStorageBlockNext**)**

GetProcAddress**(**DataAccessLibDll**,** "StorageBlockNext"**);**

StorageBlockIsDone **=** **(**PStorageBlockIsDone**)**

GetProcAddress**(**DataAccessLibDll**,** "StorageBlockIsDone"**);**

CloseStorageIterator **=** **(**PCloseStorageIterator**)**

GetProcAddress**(**DataAccessLibDll**,** "CloseStorageIterator"**);**

SetStoragePosition **=** **(**PSetStoragePosition**)**

GetProcAddress**(**DataAccessLibDll**,** "SetStoragePosition"**);**

ReadStorageData **=** **(**PReadStorageData**)**

GetProcAddress**(**DataAccessLibDll**,** "ReadStorageData"**);**

RecognizeFileSystem **=** **(**PRecognizeFileSystem**)**

GetProcAddress**(**DataAccessLibDll**,** "RecognizeFileSystem"**);**

**}**

**// Проверка подключенных функций**

**bool** HasNullFunctions**()**

**{**

**return** (

CreateStorageHandle **==** **NULL** **||** CloseStorageHandle **==** **NULL** **||**

GetStorageType **==** **NULL** **||** GetPhysicalDriveInfo **==** **NULL** **||**

GetNumberOfPartitions **==** **NULL** **||** GetPartitionInfoByNumber **==** **NULL** **||**

GetStorageBlockSize **==** **NULL** **||** SetStorageBlockSize **==** **NULL** **||**

GetStorageIterator **==** **NULL** **||** StorageBlockFirst **==** **NULL** **||**

StorageBlockNext **==** **NULL** **||** StorageBlockIsDone **==** **NULL** **||**

CloseStorageIterator **==** **NULL** **||** SetStoragePosition **==** **NULL** **||**

ReadStorageData **==** **NULL** **||** RecognizeFileSystem **==** **NULL**

);

**}**

**// Функция общей инициализации библиотеки**

**bool** \_\_fastcall DataAccessLibInit**()**

**{**

**// Загрузить файл библиотеки**

DataAccessLibDll **=** LoadLibraryW**(**L"DataAccessLib.dll"**);**

**// Присвоить указателям на функции значения**

InitFunctionsFromLib**();**

**// Проверить, все ли функции успешно подключены**

**if(** HasNullFunctions**()** **)**

**{**

FreeLibrary **(**DataAccessLibDll**);**

**return** **false;**

**}**

**else**

**{**

**return** **true;**

**}**

**}**

**// Инициализировать указатели на функции**

**void** SetFunctionsAsNull**()**

**{**

CreateStorageHandle **=** **NULL;**

CloseStorageHandle **=** **NULL;**

GetStorageType **=** **NULL;**

GetPhysicalDriveInfo **=** **NULL;**

GetNumberOfPartitions **=** **NULL;**

GetPartitionInfoByNumber **=** **NULL;**

GetStorageBlockSize **=** **NULL;**

SetStorageBlockSize **=** **NULL;**

GetStorageIterator **=** **NULL;**

StorageBlockFirst **=** **NULL;**

StorageBlockNext **=** **NULL;**

StorageBlockIsDone **=** **NULL;**

CloseStorageIterator **=** **NULL;**

SetStoragePosition **=** **NULL;**

ReadStorageData **=** **NULL;**

RecognizeFileSystem **=** **NULL;**

**}**

**// Функция отключения библиотеки**

**void** \_\_fastcall DataAccessLibCleanup**()**

**{**

**if(**DataAccessLibDll **!=** **NULL)**

FreeLibrary**(**DataAccessLibDll**);**

SetFunctionsAsNull**();**

**}**

* 1. Операции с хранилищами

**//-----------------------------------------**

**#include <iostream>**

**using namespace std;**

**//-----------------------------------------**

**#include "FileSystemExport.h"**

**//-----------------------------------------**

**void** StorageTest**()**

**{**

**// Открытие физического диска**

wstring physicalStoragePath**(**L"\\\\.\\PhysicalDrive0"**);**

StorageHandle physicalStorage **=** CreateStorageHandle**(**physicalStoragePath**.**c\_str**());**

**if(**physicalStorage **==** STORAGE\_ERROR**)** **{** wcout **<<** L"Ошибка открытия физического диска!" **<<** endl**;** **return;** **}**

**// Проверка типа хранилища**

StorageType storageType **=** GetStorageType**(**physicalStorage**);**

**if(**storageType **==** StorageType**::**PhysicalDrive**)**

**{**

**// Вывод информации об устройстве**

PhysicalDriveInfoStruct driveInfo **=** GetPhysicalDriveInfo**(**physicalStorage**);**

wcout **<<** L"Устройство: " **<<** driveInfo**.**Device **<<** endl**;**

wcout **<<** L"Модель: " **<<** driveInfo**.**ProductId **<<** endl**;**

wcout **<<** L"Серийный номер: " **<<** driveInfo**.**Serial **<<** endl**;**

wcout **<<** L"Размер: " **<<** driveInfo**.**Size **<<** endl**;**

wcout **<<** endl**;**

**}**

**// Определение размера сектора**

DWORD sectorSize **=** GetStorageBlockSize**(**physicalStorage**);**

wcout **<<** L"Размер сектора: "**<<** sectorSize **<<** L" байт" **<<** endl **<<** endl**;**

**// Вывод таблицы разделов**

**if(**storageType **==** StorageType**::**PhysicalDrive **||** storageType **==** StorageType**::**ImageFile**)**

**{**

**// Определить количество разделов**

**unsigned** **int** numberOfPartitions **=** GetNumberOfPartitions**(**physicalStorage**);**

wcout **<<** L"Всего разделов: " **<<** numberOfPartitions **<<** endl**;**

**for(unsigned** **int** i**=**1**;** i**<=**numberOfPartitions**;** i**++)**

**{**

**// Получить информацию о разделе**

PartitionInfoStruct currentPartition**;**

**if(**GetPartitionInfoByNumber**(**physicalStorage**,** i**,** **&**currentPartition**))**

**{**

FileSystemTypeEnum fsType **=** currentPartition**.**RecognizedType**;**

ULONGLONG startOffset **=** currentPartition**.**FirstSector**\***sectorSize**;**

ULONGLONG partitionSize **=** currentPartition**.**SizeInSectors**\***sectorSize**;**

wcout **<<** L"Порядковый номер: " **<<** i **<<** endl**;**

wcout **<<** L"Идентификатор файловой системы: " **<<** **int(**fsType**)** **<<** endl**;**

wcout **<<** L"Смещение раздела: " **<<** startOffset **<<** L" байт" **<<** endl**;**

wcout **<<** L"Размер раздела: " **<<** partitionSize **<<** L" байт" **<<** endl**;**

wcout **<<** endl**;**

**}**

**}**

**}**

**// Закрытие физического диска**

CloseStorageHandle**(**physicalStorage**);**

**// Открытие логического диска**

wstring logicalStoragePath**(**L"\\\\.\\C:"**);**

StorageHandle logicalStorage **=** CreateStorageHandle**(**logicalStoragePath**.**c\_str**());**

**if(**logicalStorage **==** STORAGE\_ERROR**) {** wcout **<<** L"Ошибка открытия логического диска!" **<<** endl**;** **return;** **}**

**// Произвольное чтение данных из хранилища**

LONGLONG storageOffset **=** 3**;**

**if(**SetStoragePosition**(**logicalStorage**,** storageOffset**))**

**{**

LONGLONG bytesToRead **=** 4**;**

BYTE **\***dataBuffer **=** **new** BYTE**[**bytesToRead**+**1**];**

memset**(**dataBuffer**,** 0**,** bytesToRead**+**1**);**

LONGLONG readSize **=** ReadStorageData**(**logicalStorage**,** bytesToRead**,** dataBuffer**);**

**if(**readSize **==** bytesToRead**)**

**{**

cout **<<** dataBuffer **<<** endl **<<** endl**;**

**}**

**else**

**{**

wcout **<<** L"Ошибка чтения носителя!" **<<** endl**;**

**}**

**delete[]** dataBuffer**;**

**}**

**else**

**{**

wcout **<<** L"Ошибка позиционирования на носителе!" **<<** endl**;**

**}**

**// Закрытие логического диска**

CloseStorageHandle**(**logicalStorage**);**

**// Открытие файла-образа**

wstring imageStoragePath**(**L"S:\\Disk Images\\Demo\\Demo.001"**);**

StorageHandle imageStorage **=** CreateStorageHandle**(**imageStoragePath**.**c\_str**());**

**if(**imageStorage **==** STORAGE\_ERROR**) {** wcout **<<** L"Ошибка открытия файла-образа!" **<<** endl**; return; }**

**// Задание размера блока**

DWORD blockSize **=** 0x100000**;** **// 1 МБ**

BYTE **\***blockBuffer **=** **new** BYTE**[**blockSize**];**

**if(**GetStorageType**(**imageStorage**)** **==** StorageType**::**ImageFile**)**

**{**

**if(!**SetStorageBlockSize**(**imageStorage**,** blockSize**))**

**{**

wcout **<<** L"Ошибка задания размера блока!" **<<** endl**;**

**}**

**}**

**// Последовательный обход блоков хранилища**

ULONGLONG totalReadSize **=** 0**;**

StorageIteratorHandle storageIterator **=** GetStorageIterator**(**imageStorage**);**

**for(**StorageBlockFirst**(**storageIterator**);**

**!**StorageBlockIsDone**(**storageIterator**);**

StorageBlockNext**(**storageIterator**))**

**{**

LONGLONG readSize **=** ReadStorageData**(**storageIterator**,** blockSize**,** blockBuffer**);**

**if(**readSize **==** blockSize**)**

**{**

totalReadSize **+=** readSize**;**

**}**

**else**

**{**

wcout **<<** L"Ошибка чтения!" **<<** endl**;**

**}**

**}**

CloseStorageIterator**(**storageIterator**);**

wcout **<<** L"Всего прочитано: " **<<** totalReadSize **<<** L" байт" **<<** endl**;**

**delete[]** blockBuffer**;**

**// Закрытие файла-образа**

CloseStorageHandle**(**imageStorage**);**

**}**

* 1. Операции с файловыми системами, файлами и каталогами

**void** FileSystemTest**()**

**{**

wstring imageStoragePath**(**L"Z:\\Disk Images\\Demo\\Demo.001"**);**

StorageHandle dataStorage **=** CreateStorageHandle**(**imageStoragePath**.**c\_str**());**

**if(**dataStorage **==** STORAGE\_ERROR**)** **{** wcout **<<** L"Ошибка открытия файла-образа!" **<<** endl**;** **return;** **}**

**unsigned** **int** selectedNumber **=** 2**;**

PartitionInfoStruct selectedPartitionInfo**;**

**if(!**GetPartitionInfoByNumber**(**dataStorage**,** selectedNumber**,** **&**selectedPartitionInfo**))** **{**

wcout **<<** L"Ошибка получения информации о разделе!" **<<** endl**;** **return;**

**}**

**// Установка размера блока чтения равным 1 сектор физического диска (DefaultSectorSize == 512)**

**if(!**SetStorageBlockSize**(**dataStorage**,** DefaultSectorSize**))** **{** wcout **<<** L"Ошибка задания размера блока!" **<<** endl**;** **}**

ULONGLONG startOffset **=** selectedPartitionInfo**.**FirstSector**\***DefaultSectorSize**;**

ULONGLONG partitionSize **=** selectedPartitionInfo**.**SizeInSectors**\***DefaultSectorSize**;**

**// Распознавание типа файловой системы (альтернатива selectedPartitionInfo.RecognizedType)**

FileSystemTypeEnum fsType **=** RecognizeFileSystem**(**dataStorage**,** startOffset**);**

**if(**fsType **==** FileSystemTypeEnum**::**FS\_Error **||** fsType **==** FileSystemTypeEnum**::**FS\_None**)** **{**

wcout **<<** L"Файловая система не распознана!" **<<** endl**;**

CloseStorageHandle**(**dataStorage**);**

**return;**

**}**

**// Открытие файловой системы**

FileSystemHandle fileSystem **=** CreateFileSystemHandle**(**fsType**,** dataStorage**,**

startOffset**,** partitionSize**,** DefaultSectorSize**);**

**if(**fileSystem **==** FILESYSTEM\_ERROR**)** **{**

wcout **<<** L"Ошибка открытия файловой системы!" **<<** endl**;**

CloseStorageHandle**(**dataStorage**);**  **return;**

**}**

**// Определение и вывод внутреннего типа файловой системы**

WCHAR fsTypeStr**[**20**];**

GetFileSystemTypeStringW**(**fileSystem**,** fsTypeStr**);**

wcout **<<** L"Тип файловой системы: " **<<** fsTypeStr **<<** endl**;**

**// Определение размера кластера**

DWORD clusterSize **=** GetFileSystemClusterSize**(**fileSystem**);**

BYTE **\***clusterBuffer **=** **new** BYTE**[**clusterSize**];**

**// Обход файловой системы с использованием итератора**

ULONGLONG totalReadSize1 **=** 0**;**

ULONGLONG usedSpace1 **=** 0**;**

ClusterIteratorHandle clusterIterator **=** GetClusterIterator**(**fileSystem**);**

**if(**clusterIterator **!=** CLUSTER\_ITERATOR\_ERROR**)**

**{**

**for(**ClusterFirst**(**clusterIterator**);** **!**ClusterIsDone**(**clusterIterator**);** ClusterNext**(**clusterIterator**))**

**{**

LONGLONG readSize **=** GetCurrentCluster**(**clusterIterator**,** clusterBuffer**);**

**if(**readSize **!=** clusterSize**)** **{**

wcout **<<** L"Ошибка чтения кластера с использованием итератора!" **<<** endl**;**

**break;**

**}**

totalReadSize1 **+=** readSize**;**

**// В качестве ПРИМЕРА - подсчет количества занятого пространства (НЕОПТИМАЛЬНЫЙ СПОСОБ!)**

**if(**ClusterIsUsed**(**clusterIterator**))**

**{**

usedSpace1 **+=** clusterSize**;**

**}**

**if(**totalReadSize1 **>** 0x8000000**)** **break;** **// 128 МБ**

**}**

wcout **<<** L"Всего прочитано: " **<<** totalReadSize1 **<<** L" байт" **<<** endl**;**

wcout **<<** L"Всего занято: " **<<** usedSpace1 **<<** L" байт" **<<** endl**;**

CloseClusterIterator**(**clusterIterator**);**

**}**

**else {** wcout **<<** L"Ошибка создания итератора по кластерам!" **<<** endl**;** **}**

**// Обход файловой системы в цикле по номерам кластеров**

ULONGLONG totalReadSize2 **=** 0**;**

ULONGLONG usedSpace2 **=** 0**;**

ULONGLONG numberOfClusters **=** GetNumberOfClusters**(**fileSystem**);**

**for(**ULONGLONG i**=**0**;** i**<**numberOfClusters**;** i**++)**

**{**

LONGLONG readSize **=** ReadClustersByNumber**(**fileSystem**,** i**,** 1**,** clusterBuffer**,** clusterSize**);**

**if(**readSize **!=** clusterSize**)**

**{**

wcout **<<** L"Ошибка чтения кластера с использованием цикла!" **<<** endl**;**

**break;**

**}**

totalReadSize2 **+=** readSize**;**

**if(**totalReadSize2 **>** 0x8000000**)** **break;**

**}**

wcout **<<** L"Всего прочитано: " **<<** totalReadSize2 **<<** L" байт" **<<** endl**;**

**delete[]** clusterBuffer**;**

BYTE **\***fileSystemBuffer **=** **new** BYTE**[**0x8000000**];**

**// Чтение первых 128 МБ файловой системы**

SetFileSystemPosition**(**fileSystem**,** 0**);**

LONGLONG fileSystemReadSize **=** ReadFileSystemData**(**fileSystem**,** 0x8000000**,** fileSystemBuffer**);**

**if(**fileSystemReadSize **==** 0x8000000**)**

**{**

**// Запись считанных данных в файл**

WriteDataToFileW**(**fileSystemBuffer**,** fileSystemReadSize**,** L"E:\\Temp\\FileSystemOutData.dat"**);**

wcout **<<** L"Всего прочитано и сохранено: " **<<** fileSystemReadSize **<<** L" байт" **<<** endl**;**

**}**

**else**

**{**

wcout **<<** L"Ошибка чтения кластера с использованием цикла!" **<<** endl**;**

**}**

**delete[]** fileSystemBuffer**;**

**// Открытие файла**

FileHandle fileObject **=** CreateFileHandle**(**fileSystem**,** L"\\.\\Windows\\system32\\config\\software"**);**

**if(**fileObject **!=** FILE\_OBJECT\_ERROR**)**

**{**

**// Вывод временных отметок файла**

**char** timeStampCreate**[**40**],** timeStampModify**[**40**],** timeStampAccess**[**40**];**

ULONGLONG timeCreate **=** GetTimeFileCreate**(**fileObject**,** timeStampCreate**);**

ULONGLONG timeModify **=** GetTimeFileModify**(**fileObject**,** timeStampModify**);**

ULONGLONG timeAccess **=** GetTimeFileAccess**(**fileObject**,** timeStampAccess**);**

wcout **<<** L"Время создания: " **<<** timeStampCreate **<<** endl**;**

wcout **<<** L"Время изменения: " **<<** timeStampModify **<<** endl**;**

wcout **<<** L"Время доступа: " **<<** timeStampAccess **<<** endl**;**

**// Вывод атрибутов файла**

**char** attrStr**[**20**];**

GetFileStdAttributes**(**fileObject**,** attrStr**);**

wcout **<<** L"Атрибуты: " **<<** attrStr **<<** endl**;**

**// Вывод хэш-значения основного (нулевого) потока данных файла**

**unsigned** **int** fileHash**[**5**];**

**if(**GetFileHash**(**fileObject**,** 0**,** fileHash**))**

**{**

WCHAR tempStr**[**80**];**

swprintf**(**tempStr**,** L"%08X %08X %08X %08X %08X"**,**

fileHash**[**0**],** fileHash**[**1**],** fileHash**[**2**],** fileHash**[**3**],** fileHash**[**4**]);**

wcout **<<** L"SHA1: " **<<** wstring**(**tempStr**)** **<<** endl**;**

**}**

**// Определение размера основного (нулевого) потока данных файла**

LONGLONG fileDataSize **=** GetFileDataSize**(**fileObject**,** 0**);**

**// Чтение его содержимое в память**

BYTE **\***dataBuffer **=** **new** BYTE**[**fileDataSize**];**

SetFilePosition**(**fileObject**,** 0**,** 0**,** OriginEnum**::**Begin**);**

ULONGLONG readSize **=** ReadFileData**(**fileObject**,** fileDataSize**,** dataBuffer**);**

**if(**readSize **==** fileDataSize**)** **{**

WriteDataToFileW**(**dataBuffer**,** readSize**,** L"E:\\Temp\\RegistrySoftware1"**);**

wcout **<<** L"Прочитано и сохранено: " **<<** readSize **<<** L" байт" **<<** endl**;**

**}**

**else** **{** wcout **<<** L"Ошибка чтения потока данных файла!" **<<** endl**; }**

**// "Быстрое" сохранение основного (нулевого) потока данных файла**

**if(!**SaveFile**(**fileObject**,** 0**,** L"E:\\Temp\\RegistrySoftware2"**))** **{**

wcout **<<** L"Ошибка \"быстрого\" сохранения файла!" **<<** endl**;**

**}**

CloseFileHandle**(**fileObject**);**

**} else {** wcout **<<** L"Ошибка открытия файла!" **<<** endl**;** **}**

**// Открытие каталога**

wstring dirPath**(**L"\\.\\Windows\\system32\\config"**);**

FileHandle dirObject **=** CreateFileHandle**(**fileSystem**,** dirPath**.**c\_str**());**

**if(**dirObject **!=** FILE\_OBJECT\_ERROR**)**

**{**

**if(**FileIsDir**(**dirObject**))** **// Проверить, является ли открытый объект каталогом**

**{**

wcout **<<** endl **<<** L"Содержимое каталога " **<<** dirPath **<<** L" :" **<<** endl **<<** endl**;**

**// Вывод содержимого каталога**

DirectoryIteratorHandle dirHandle **=** CreateDirectoryIteratorHandle**(**dirObject**);**

**if(**dirHandle **!=** DIRECTORY\_ITERATOR\_ERROR**)**

**{**

**for(**DirFirst**(**dirHandle**);** **!**DirIsDone**(**dirHandle**);** DirNext**(**dirHandle**))**

**{**

**// Получение и вывод имени файла ...**

**int** fileNameLength **=** GetFileNameW**(**dirHandle**,** **NULL);**

WCHAR **\***fileName **=** **new** WCHAR**[**fileNameLength **+** 1**];**

GetFileNameW**(**dirHandle**,** fileName**);**

wcout **<<** fileName **<<** L"\t:\t"**;**

**delete[]** fileName**;**

**if(**DirFileIsDir**(**dirHandle**))**

**{**

**// Текущий объект является каталогом**

wcout **<<** L"<DIR>" **<<** endl**;**

**}**

**else**

**{**

**// Создать файл по полному пути**

**int** fullPathLength **=** GetFullPathW**(**dirHandle**,** **NULL);**

WCHAR **\***fullPath **=** **new** WCHAR**[**fullPathLength **+** 1**];**

GetFullPathW**(**dirHandle**,** fullPath**);**

FileHandle currentFileObject **=** CreateFileHandle**(**fileSystem**,** fullPath**);**

**delete[]** fullPath**;**

**// Получение и вывод размера файла**

**if(**currentFileObject **!=** FILE\_OBJECT\_ERROR**)**

**{**

wcout **<<** GetFileDataSize**(**currentFileObject**,** 0**)** **<<** endl**;**

CloseFileHandle**(**currentFileObject**);**

**}**

**else {** wcout **<<** L"Ошибка!" **<<** endl**; }**

**}**

**}**

CloseDirectoryIterator**(**dirHandle**);**

**}**

**else {** wcout **<<** L"Ошибка создания итератора!" **<<** endl**;** **}**

**// Вывод общего числа объектов в каталоге**

wcout **<<** endl **<<** L"Всего объектов в каталоге: " **<<** GetNumberOfFiles**(**dirObject**)** **<<** endl**;**

**}**

**// "Быстрое" сохранение каталога с восстановлением полных путей файлов**

**if(!**SaveFileWithPath**(**dirObject**,** L"E:\\Temp"**))**

**{**

wcout **<<** L"Ошибка \"быстрого\" сохранения каталога!" **<<** endl**;**

**}**

CloseFileHandle**(**dirObject**);**

**}**

**else** **{** wcout **<<** L"Ошибка создания каталога!" **<<** endl**; }**

**// Закрытие файла-образа**

CloseStorageHandle**(**dataStorage**);**

**}**

1. Входные и выходные данные
   1. Основные типы, псевдонимы и константы
      1. StorageType

Целочисленный идентификатор типа хранилища.

* + 1. FileSystemTypeEnum

Целочисленный идентификатор типа файловой системы (раздела, области хранилища).

* + 1. OriginEnum

Целочисленный идентификатор точки отсчета задаваемой позиции при навигации по файловой системе или потоку данных файла.

| **Тип** | **Значение** | **Описание** |
| --- | --- | --- |
| Begin | 0 | Отсчет ведется от начала области |
| Current | 1 | Отсчет ведется от текущего положения |
| End | 2 | Отсчет ведется от конца области |

* + 1. PhysicalDriveInfoStruct

Структура, содержащая информацию о физическом диске.

struct PhysicalDriveInfoStruct

{

bool Error;

char Device[24];

char ProductId[48];

char Serial[48];

ULONGLONG Size;

};

| **Поле** | **Тип** | **Размер, байт** | **Описание** |
| --- | --- | --- | --- |
| Error | bool | 4 | Признак ошибки |
| Device | char[24] | 24 | Имя устройства в формате: «\\.\PhysicalDriveN» |
| ProductId | char[48] | 48 | Модель устройства |
| Serial | char[48] | 48 | Серийный номер устройства |
| Size | ULONGLONG | 8 | Размер в байтах |

* + 1. PartitionInfoStruct

Структура, содержащая информацию о разделе.

struct PartitionInfoStruct

{

FileSystemTypeEnum RecognizedType;

unsigned char RawType;

unsigned \_\_int64 FirstSector;

unsigned \_\_int64 LastSector;

unsigned \_\_int64 SizeInSectors;

unsigned char TypeGuid[16];

unsigned char PartGuid[16];

unsigned \_\_int64 Flags;

wchar\_t Name[36];

};

| **Поле** | **Тип** | **Размер, байт** | **Описание** |
| --- | --- | --- | --- |
| RecognizedType | FileSystemTypeEnum | 4 | Тип раздела, полученный в результате распознавания (см. п. 3.1.2) |
| RawType | unsigned char | 1 | Идентификатор типа раздела, полученный из таблицы разделов MBR |
| FirstSector | unsigned \_\_int64 | 8 | Начальный сектор раздела |
| LastSector | unsigned \_\_int64 | 8 | Конечный сектор раздела |
| SizeInSectors | unsigned \_\_int64 | 8 | Размер раздела в секторах |
| TypeGuid | unsigned char[16] | 16 | GUID типа раздела, полученный из таблицы разделов GPT |
| PartGuid | unsigned char[16] | 16 | GUID раздела, полученный из таблицы разделов GPT |
| Flags | unsigned \_\_int64 | 8 | Флаги раздела (из GPT или MBR) |
| Name | wchar\_t[36] | 64 | Имя раздела, полученное из таблицы разделов GPT |

* + 1. Псевдонимы

| **Наименование** | **Тип** | **Описание** |
| --- | --- | --- |
| StorageHandle | int | Дескриптор хранилища |
| StorageIteratorHandle | int | Дескриптор последовательного обхода хранилища |
| FileSystemHandle | int | Дескриптор файловой системы |
| ClusterIteratorHandle | int | Дескриптор последовательного обхода кластеров файловой системы |
| FileHandle | int | Дескриптор файлового объекта |
| DirectoryIteratorHandle | int | Дескриптор последовательного обхода каталога |

* + 1. Константы

| **Наименование** | **Тип** | **Значение** |
| --- | --- | --- |
| STORAGE\_ERROR | StorageHandle | –1 |
| FILESYSTEM\_ERROR | FileSystemHandle | –1 |
| CLUSTER\_ITERATOR\_ERROR | ClusterIteratorHandle | –1 |
| FILE\_OBJECT\_ERROR | FileHandle | –1 |
| DIRECTORY\_ITERATOR\_ERROR | DirectoryIteratorHandle | –1 |
| READ\_ERROR | \_\_int64 | –1 |
| DefaultSectorSize | int | 512 |

* 1. Операции с хранилищами
     1. CreateStorageHandle

Создание дескриптора хранилища компьютерной информации.

Прототип

StorageHandle \_\_stdcall CreateStorageHandle(

const WCHAR \*fileName

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileName | const WCHAR \* | [вх] Завершающаяся нулем строка, содержащая имя открываемого хранилища |

Возвращаемое значение

В случае успешного завершения возвращает целочисленный дескриптор, который впоследствии используется в качестве идентификатора открытого хранилища. В случае ошибок возвращает константное значение STORAGE\_ERROR.

Примечание

В качестве имен хранилищ могут использоваться имена логических и физических дисков, файлов-образов, каталогов: «C:», «\\.\PhysicalDrive0», «Z:\Disk Images\SEAGATE\_1000», «E:\Temp».

* + 1. GetStorageType

Получение информации о типе хранилища.

Прототип

StorageType \_\_stdcall GetStorageType(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

В случае успешного завершения возвращает один из целочисленных идентификаторов типа хранилища [StorageType](#StorageType) (п. 3.1.1). В случае ошибки функция возвращает StorageType::Error.

* + 1. GetPhysicalDriveInfo

Получение информации о физическом диске.

Прототип

PhysicalDriveInfoStruct \_\_stdcall GetPhysicalDriveInfo(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища  (типа StorageType::PhysicalDrive) |

Возвращаемое значение

В случае успешного завершения возвращает структуру [PhysicalDriveInfoStruct](#PhysicalDriveInfoStruct) (п. 3.1.4), содержащую информацию об указанном хранилище. В случае ошибки поле Error структуры принимает значение true.

Примечание

Функция корректно работает лишь для физических дисков (хранилищ, имеющих тип StorageType::PhysicalDrive). Идентификатор и серийный номер физического диска считываются стандартными для ОС Windows вызовом функции DeviceIoControl, использующего IOCTL-запрос IOCTL\_STORAGE\_QUERY\_PROPERTY. Такой подход не всегда корректно выводит идентификаторы НЖМД, подключенных через USB-адаптер. Идентификатор модели устройства возвращается «как есть», а серийный номер преобразуется к общепринятому цифробуквенному виду, характерному для таких программ, как *R-Studio*. Размер устройства определяется с использованием IOCTL-запроса IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY только после его *успешного* открытия функцией CreateFile, поэтому для недоступных в данный момент устройств размер принимается равным 0.

* + 1. GetNumberOfPartitions

Определение количества областей в таблице разделов.

Прототип

int \_\_stdcall GetNumberOfPartitions(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

В случае успешного завершения возвращает количество областей, на которые разбито хранилище, включая разделы файловых систем, служебные области таблицы разделов и свободное пространство (в том числе незанятые интервалы между разделами).

Примечание

Функция, как правило, используется для определения верхней границы в цикле по всем разделам (областям) хранилища компьютерной информации.

* + 1. GetPartitionInfoByNumber

Получение информации о разделе (области) хранилища информации по порядковому номеру раздела.

Прототип

bool \_\_stdcall GetPartitionInfoByNumber(

StorageHandle storageHandle,

unsigned int selectedNumber,

PartitionInfoStruct \*outPartitionInfo

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| selectedNumber | unsigned int | [вх] Порядковый номер раздела  (области) хранилища, начиная с 1 |
| outPartitionInfo | PartitionInfoStruct \* | [вых] Указатель на структуру,  в которую необходимо записать  информацию о разделе (области) |

Возвращаемое значение

В случае успешного завершения функция возвращает true, а структура, на которую указывает outPartitionInfo, заполняется необходимой информацией (п. 3.1.5). В случае ошибки возвращается значение false.

Примечание

Значение поля RecognizedType возвращаемой структуры получается в результате использования встроенного в библиотеку алгоритма сигнатурного распознавания типа раздела (области). При необходимости открытия файловой системы для уточнения ее типа рекомендуется дополнительно вызвать функцию [RecognizeFileSystem](#RecognizeFileSystem) (п. 3.2.15).

* + 1. GetStorageBlockSize

Определение размера сектора хранилища.

Прототип

DWORD \_\_stdcall GetStorageBlockSize(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

В случае успешного завершения возвращает размер сектора — минимального блока чтения информации из хранилища. В случае ошибки возвращает 0.

Примечание

Для хранилищ типа StorageType::PhysicalDrive и StorageType::LogicalDrive возвращаемый функцией размер сектора соответствует реальному размеру сектора физического диска, для хранилищ типа StorageType::ImageFile и StorageType::DirectoryFile по умолчанию равен 64 байта, для хранилищ типа StorageType::DataStream  — равен размеру кластера «родительской» файловой системы.

Обычно функция используется для определения размера буфера с целью дальнейшей огранизации последовательного обхода секторов хранилища при помощи функций [GetStorageIterator](#GetStorageIterator), [StorageBlockFirst](#StorageBlockFirst), [StorageBlockNext](#StorageBlockNext), [StorageBlockIsDone](#StorageBlockIsDone), [ReadStorageData](#ReadStorageData) (п. 3.2.8–3.2.12, 3.2.14).

* + 1. SetStorageBlockSize

Задание размера сектора хранилища.

Прототип

bool \_\_stdcall SetStorageBlockSize(

StorageHandle storageHandle, DWORD blockSize

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| blockSize | DWORD | [вх] Новый размер сектора |

Возвращаемое значение

В случае успешной установки размера сектора для указанного хранилища возвращает true, в случае ошибки — false.

Примечание

Для хранилищ типа StorageType::PhysicalDrive и StorageType::LogicalDrive размер сектора должен быть кратен размеру физического сектора, для StorageType::DataStream — кратен размеру кластера «родительской» файловой системы. Для хранилищ типа StorageType::ImageFile и StorageType::DirectoryFile может быть использовано любое положительное значение.

* + 1. GetStorageIterator

Получение дескриптора последовательного обхода секторов хранилища.

Прототип

StorageIteratorHandle \_\_stdcall GetStorageIterator(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор. В случае ошибки возвращает STORAGE\_ERROR.

* + 1. StorageBlockFirst

Переход к первому по счету блоку хранилища (блоку с индексом 0).

Прототип

void \_\_stdcall StorageBlockFirst(

StorageIteratorHandle storageIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

Отсутствует.

* + 1. StorageBlockNext

Переход к следующему блоку хранилища.

Прототип

void \_\_stdcall StorageBlockNext(

StorageIteratorHandle storageIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

Отсутствует.

* + 1. StorageBlockIsDone

Проверка окончания цикла обхода секторов хранилища.

Прототип

bool \_\_stdcall StorageBlockIsDone(

StorageIteratorHandle storageIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

Возвращает true при позиционировании на сектор, следующий *за последним сектором* хранилища, а также в случае, когда хранилище имеет нулевой размер и не содержит данных. В остальных случаях возвращает false.

* + 1. CloseStorageIterator

Закрытие оператора последовательного обхода хранилища.

Прототип

void \_\_stdcall CloseStorageIterator(

StorageIteratorHandle storageIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

Отсутствует.

* + 1. SetStoragePosition

Задание текущей позиции в хранилище.

Прототип

bool \_\_stdcall SetStoragePosition(

StorageHandle storageHandle,

LONGLONG newPosition,

OriginEnum newOrigin = OriginEnum::Begin

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| newPosition | LONGLONG | [вх] Новое значение позиции |
| newOrigin | OriginEnum | [вх] Точка отсчета позиции (п. 3.1.3) |

Возвращаемое значение

Возвращает true при успешном выполнении позиционирования. В остальных случаях возвращает false.

Примечание

Функция используется для произвольного позиционирования в пределах хранилища, обычно с целью последующего чтения данных. Для выполнения чтения функцией ReadStorageData позиция может быть не кратна размеру сектора. Выполняется проверка на возможный выход устанавливаемой позиции за пределы области данных хранилища.

* + 1. ReadStorageData

Чтение данных из хранилища.

Прототип

LONGLONG \_\_stdcall ReadStorageData(

StorageHandle storageHandle,

LONGLONG bytesToRead,

BYTE \*dataBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| bytesToRead | LONGLONG | [вх] Количество байт, подлежащих считыванию |
| dataBuffer | BYTE \* | [вых] Указатель на массив, в который следует производить чтение |

Возвращаемое значение

Возвращает количество считанных байтов. При успешном завершении оно должно быть равно bytesToRead.

Примечание

Функция выполняет буферизованное чтение данных из хранилища по смещению, предварительно заданному функциями StorageBlockFirst, StorageBlockNext или SetStoragePosition. Предполагается, что буфер имеет размер не меньше, чем bytesToRead, при этом дополнительный запас по размеру буфера не требуется.

* + 1. RecognizeFileSystem

Распознавание файловой системы по заданному смещению на хранилище, заданном при помощи дескриптора.

Прототип

FileSystemTypeEnum \_\_stdcall RecognizeFileSystem(

StorageHandle storageHandle,

ULONGLONG startOffset

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| startOffset | ULONGLONG | [вх] Смещение относительно начала хранилища |

Возвращаемое значение

В случае успешного завершения возвращает идентификатор типа файловой системы (п. 3.1.2), в случае ошибки — значение FileSystemTypeEnum:: FS\_Error.

Примечание

Распознавание производится путем считывания первых 2048 байт по смещению startOffset относительно начала контейнера и последовательной проверки с использованием перечня сигнатур известных файловых систем (соответствует множеству возможных значений перечисляемого типа [FileSystemTypeEnum](#FileSystemTypeEnum)).

* + 1. CloseStorageHandle

Закрытие дескриптора хранилища компьютерной информации.

Прототип

void \_\_stdcall CloseStorageHandle(

StorageHandle storageHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |

Возвращаемое значение

Отсутствует.

* 1. Операции с файловой системой
     1. CreateFileSystemHandle

Создание дескриптора файловой системы.

Прототип

FileSystemHandle \_\_stdcall CreateFileSystemHandle(

FileSystemTypeEnum fsType,

StorageHandle storageHandle,

ULONGLONG startOffset,

ULONGLONG diskSize,

WORD sectorSize

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fsType | FileSystemTypeEnum | [вх] Тип файловой системы |
| storageHandle | StorageHandle | [вх] Дескриптор хранилища |
| startOffset | ULONGLONG | [вх] Смещение относительно начала хранилища |
| diskSize | ULONGLONG | [вх] Максимальный размер файловой системы в байтах |
| sectorSize | WORD | [вх] Размер одного сектора |

Возвращаемое значение

В случае успешного завершения возвращает описатель файловой системы. В случае ошибки возвращает FILESYSTEM\_ERROR.

Примечание

Тип создаваемой файловой системы рекомендуется предварительно определить с использованием вызова функции [RecognizeFileSystem](#RecognizeFileSystem) (п. 3.2.15). При создании файловой системы в пределах предварительно идентифицированного раздела значения startOffset и diskSize должны быть получено при помощи функции [GetPartitionInfoByNumber](#GetPartitionInfoByNumber) (размеры в секторах необходимо умножить на размер сектора в байтах, полученный при помощи функции [GetStorageBlockSize](#GetStorageBlockSize)).

При необходимости создания «фиктивной» файловой системы, представляющей собой массив кластеров заданного размера, следует указать тип создаваемой файловой системы FileSystemTypeEnum:: FS\_None.

* + 1. FastCreateFileSystemHandle

«Быстрое» создание дескриптора файловой системы.

Прототип

FileSystemHandle \_\_stdcall FastCreateFileSystemHandle(

const WCHAR \*fileName,

ULONGLONG startOffset = 0

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileName | const WCHAR \* | [вх] Завершающаяся нулем строка, содержащая имя открываемого хранилища |
| startOffset | ULONGLONG | [вх] Смещение относительно начала хранилища |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор файловой системы. В случае ошибки возвращает FILESYSTEM\_ERROR.

Примечание

Функция используется для создания дескриптора файловой системы без предварительного создания дескриптора хранилища. Определение типа файловой системы производится автоматически.

В качестве имен хранилищ могут использоваться имена логических и физических дисков, файлов-образов, каталогов: «C:», «\\.\PhysicalDrive0», «Z:\Disk Images\SEAGATE\_1000», «E:\Temp».

* + 1. GetFileSystemType

Определение типа файловой системы.

Прототип

FileSystemTypeEnum \_\_stdcall GetFileSystemType(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает один из целочисленных идентификаторов типа файловой системы [FileSystemTypeEnum](#FileSystemTypeEnum) (п. 3.1.2). В случае ошибки функция возвращает FileSystemTypeEnum::FS\_Error.

* + 1. GetFileSystemTypeString, GetFileSystemTypeStringW

Получение наименования файловой системы.

Прототипы

int \_\_stdcall GetFileSystemTypeString(

FileSystemHandle fileSystemHandle,

char \*outType = NULL );

int \_\_stdcall GetFileSystemTypeStringW(

FileSystemHandle fileSystemHandle,

WCHAR \*outType = NULL );

| **Параметр** | **Тип** | **Описание** |
| --- | --- | --- |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| outType | char \*  WCHAR \* | [вых] Указатель на буфер, в который будет записано наименование файловой системы в кодировке ANSI или UTF-16 |

Возвращаемое значение

В случае успешного завершения возвращает количество символов в наименовании файловой системы. В случае ошибки возвращает текстовое значение «Error».

Примечание

Если параметр outType имеет значение NULL, то запись данных в буфер не производится, однако возвращается длина наименования файловой системы в символах. Длина наименования не превышает 20 символов.

* + 1. GetNumberOfClusters

Получение количества кластеров файловой системы.

Прототип

ULONGLONG \_\_stdcall GetNumberOfClusters(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает количество кластеров файловой системы. В случае ошибки возвращает значение 0.

Примечание

Функция предназначена для работы с «настоящими» (не виртуальными) файловыми системами (п. 3.1.2), позволяющими напрямую работать с кластерами данных.

* + 1. GetFileSystemSize

Получение размера файловой системы в байтах.

Прототип

ULONGLONG \_\_stdcall GetFileSystemSize(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает размер в байтах области, которая размечена указанной файловой системой. В случае ошибки возвращает значение 0.

Примечание

Сведения о размере файловой системы извлекаются из служебных областей файловой системы (как правило, загрузочной записи или заголовка раздела).

* + 1. GetFileSystemClusterSize

Получение размера в байтах одного кластера файловой системы.

Прототип

DWORD \_\_stdcall GetFileSystemClusterSize(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает размер в байтах одного кластера файловой системы. В случае ошибки возвращает значение 0.

* + 1. GetClusterIterator

Создание дескриптора последовательного обхода кластеров файловой системы.

Прототип

ClusterIteratorHandle \_\_stdcall GetClusterIterator(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор последовательного обхода кластеров файловой системы. В случае ошибки возвращает CLUSTER\_ITERATOR\_ERROR.

Примечание

Обход кластеров файловой системы происходит по порядку возрастания номера. Для файловых систем FAT и exFAT начальным является кластер с индексом 2, в остальных файловых системах — кластер с индексом 0.

* + 1. ClusterFirst

Переход к начальному кластеру файловой системы.

Прототип

void \_\_stdcall ClusterFirst(

ClusterIteratorHandle clusterIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |

Возвращаемое значение

Отсутствует.

Примечание

Для файловых систем FAT и exFAT начальным является кластер с индексом 2, в остальных файловых системах — кластер с индексом 0.

* + 1. ClusterNext

Переход к следующему кластеру файловой системы.

Прототип

void \_\_stdcall ClusterNext(

ClusterIteratorHandle clusterIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |

Возвращаемое значение

Отсутствует.

* + 1. ClusterIsDone

Проверка окончания цикла обхода кластеров файловой системы.

Прототип

bool \_\_stdcall ClusterIsDone(

ClusterIteratorHandle clusterIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |

Возвращаемое значение

Возвращает true при позиционировании на кластер, следующий *за последним кластером* раздела, а также в случае, когда файловая система не позволяет напрямую обращаться к кластерам. В остальных случаях возвращает false.

* + 1. ClusterIsUsed

Определение того, является ли текущий кластер частью файла или помечен как свободный.

Прототип

bool \_\_stdcall ClusterIsUsed(

ClusterIteratorHandle clusterIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |

Возвращаемое значение

Возвращает true, если текущий кластер помечен в файловой системе как «занятый» (являющийся частью существующего файла). В остальных случаях возвращает false.

* + 1. GetCurrentCluster

Чтение данных из текущего кластера.

Прототип

LONGLONG \_\_stdcall GetCurrentCluster(

ClusterIteratorHandle clusterIteratorHandle,

BYTE \*dataBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |
| dataBuffer | BYTE \* | [вых] Указатель на область памяти, в которую будут записаны данные из кластера |

Возвращаемое значение

В случае успешного завершения возвращает количество считанных байтов. В случае ошибки возвращает значение –1.

Примечание

Размер выделенного буфера должен иметь размер, больший или равный значению, возвращаемому функцией [GetFileSystemClusterSize](#GetFileSystemClusterSize) (п. 3.3.7).

* + 1. CloseClusterIterator

Закрытие дескриптора последовательного обхода кластеров.

Прототип

void \_\_stdcall CloseClusterIterator(

ClusterIteratorHandle clusterIteratorHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| clusterIteratorHandle | ClusterIteratorHandle | [вх] Дескриптор последовательного обхода кластеров |

Возвращаемое значение

Отсутствует.

* + 1. SetFileSystemPosition

Задание текущей позиции в области памяти, занимаемой файловой системой.

Прототип

bool \_\_stdcall SetFileSystemPosition(

FileSystemHandle fileSystemHandle,

LONGLONG newPosition,

OriginEnum newOrigin = OriginEnum::Begin

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| newPosition | LONGLONG | [вх] Новое значение позиции |
| newOrigin | OriginEnum | [вх] Точка отсчета позиции (п. 3.1.3) |

Возвращаемое значение

Возвращает true при успешном выполнении позиционирования. В остальных случаях возвращает false.

Примечание

Функция используется для произвольного позиционирования в пределах файловой системы, обычно с целью последующего чтения данных. Для выполнения чтения функцией [ReadFileSystemData](#ReadFileSystemData) (п. 3.3.16) позиция может быть не кратна размеру кластера. Выполняется проверка на возможный выход устанавливаемой позиции за пределы области данных файловой системы.

* + 1. ReadFileSystemData

Чтение данных из области памяти, занимаемой файловой системой.

Прототип

LONGLONG \_\_stdcall ReadFileSystemData(

FileSystemHandle fileSystemHandle,

ULONGLONG bytesToRead,

BYTE \*dataBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| bytesToRead | ULONGLONG | [вх] Количество считываемых байт |
| dataBuffer | BYTE \* | [вых] Указатель на буфер, в который будут считаны данные из файловой системы |

Возвращаемое значение

В случае успешного завершения возвращает количество считанных и записанных в буфер данных (в байтах). В случае ошибки возвращает READ\_ERROR. В случае отсутствия данных, подлежащих считыванию (например, при достижении конца файловой системы), возвращает 0.

Примечание

Функция выполняет буферизованное чтение данных из файловой системы по смещению, предварительно заданному функцией [SetFileSystemPosition](#SetFileSystemPosition) (п. 3.3.15). Предполагается, что буфер имеет размер не меньше, чем bytesToRead, при этом дополнительный запас по размеру буфера не требуется. Для проверки корректности выполнения операции чтения следует в обязательном порядке сравнивать возвращаемое функцией значение со значением bytesToRead.

* + 1. ReadClustersByNumber

Чтение заданного количества кластеров файловой системы, начиная с указанного.

Прототип

LONGLONG \_\_stdcall ReadClustersByNumber(

FileSystemHandle fileSystemHandle,

ULONGLONG clusterId,

DWORD numberOfClusters,

BYTE \*dataBuffer,

const DWORD bufferSize );

| **Параметр** | **Тип** | **Описание** |
| --- | --- | --- |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| clusterId | ULONGLONG | [вх] Номер начального кластера |
| numberOfClusters | DWORD | [вх] Количество кластеров для чтения |
| dataBuffer | BYTE \* | [вых] Указатель на буфер, в который будут записаны данные |
| bufferSize | const DWORD | [вх] Размер буфера |

Возвращаемое значение

В случае успешного завершения возвращает количество считанных и записанных в буфер данных (в байтах). В случае ошибки возвращает READ\_ERROR. В случае отсутствия данных, подлежащих считыванию (например, при достижении конца файловой системы), возвращает 0.

Примечание

В буфер записывается максимум bufferSize байт.

* + 1. CloseFileSystemHandle

Закрытие дескриптора файловой системы.

Прототип

void \_\_stdcall CloseFileSystemHandle(

FileSystemHandle fileSystemHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |

Возвращаемое значение

Отсутствует.

* 1. Операции с отдельными файлами и каталогами
     1. CreateFileHandle

Создание дескриптора файлового объекта по его полному пути в гостевой файловой системе.

Прототип

FileHandle \_\_stdcall CreateFileHandle(

FileSystemHandle fileSystemHandle,

const WCHAR \*fullPath

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| fullPath | const WCHAR \* | [вх] Полный путь до объекта в гостевой файловой системе |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор файлового объекта. В случае ошибки возвращает FILE\_OBJECT\_ERROR.

Примечание

Полный путь до объекта должен начинаться с символов «\.», обозначающих корневой каталог: «\.\Windows\System32\config\software» (регистр значения не имеет).

* + 1. CreateFileHandleByRecordId

Создание дескриптора файлового объекта по его числовому идентификатору (работает не для всех файловых систем).

Прототип

FileHandle \_\_stdcall CreateFileHandleByRecordId(

FileSystemHandle fileSystemHandle,

ULONGLONG recordId

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileSystemHandle | FileSystemHandle | [вх] Дескриптор файловой системы |
| recordId | ULONGLONG | [вх] Числовой идентификатор файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор файлового объекта. В случае ошибки возвращает FILE\_OBJECT\_ERROR.

Примечание

Функция работает в файловых системах, позволяющих находить файловые объекты по их идентификаторам. В файловой системе NTFS с использованием этой функции может быть получен полноценный файловый объект, в файловой системе Ext\* файловый объект будет создан, однако его полный путь не будет известен. В файловых системах HFS+, FAT и exFAT, виртуальной файловой системе DFS (каталог) и при работе с архивами функция не работает.

* + 1. FileIsDir

Определение того, является ли файловый объект каталогом.

Прототип

bool \_\_stdcall FileIsDir(

FileHandle fileHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |

Возвращаемое значение

Если заданный файловый объект является каталогом, возвращает true. В противном случае возвращает false. В случае ошибки возвращает false.

* + 1. GetNumberOfStreams

Получение количества потоков данных файлового объекта.

Прототип

int \_\_stdcall GetNumberOfStreams(

FileHandle fileHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает количество потоков данных файлового объекта. В случае ошибки возвращает значение –1.

Примечание

Несколько потоков данных может быть у одного файлового объекта файловой системы NTFS, а также в виртуальных файловых системах, описывающих базы данных клиентских почтовых программ (где потоки данных соответствуют различным полям сообщений электронной почты).

* + 1. GetNumberOfFiles

Получение количества объектов файловой системы (файлов и каталогов) в каталоге.

Прототип

int \_\_stdcall GetNumberOfFiles(

FileHandle fileHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает количество объектов файловой системы в каталоге. Если объект не является каталогом и в случае других ошибок возвращает значение –1.

Примечание

Функция корректно работает только для файловых объектов, являющихся каталогами.

* + 1. GetFileDataSize

Получение размера заданного потока данных файла.

Прототип

LONGLONG \_\_stdcall GetFileDataSize(

FileHandle fileHandle,

int streamIndex

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| streamIndex | int | [вх] Индекс потока данных |

Возвращаемое значение

В случае успешного завершения возвращает размер в байтах потока данных файла с заданным. В случае ошибки возвращает значение –1.

Примечание

Основной поток данных файла имеет индекс 0. Во всех файловых системах, кроме NTFS и виртуальных файловых систем баз данных клиентских почтовых программ, файловый объект имеет не более одного потока данных.

* + 1. GetTimeFile[Create/Modify/Access/RecordModify]

Получение временных отметок создания, последней модификации, последнего доступа и последней модификации файловой записи файлового объекта соответственно.

Прототипы

ULONGLONG \_\_stdcall GetTimeFileCreate(

FileHandle fileHandle,

char \*outBuffer = NULL,

bool localTime = true,

long utcBias = 0 );

ULONGLONG \_\_stdcall GetTimeFileModify(

FileHandle fileHandle,

char \*outBuffer = NULL,

bool localTime = true,

long utcBias = 0 );

ULONGLONG \_\_stdcall GetTimeFileAccess(

FileHandle fileHandle,

char \*outBuffer = NULL,

bool localTime = true,

long utcBias = 0 );

ULONGLONG \_\_stdcall GetTimeFileRecordModify(

FileHandle fileHandle,

char \*outBuffer = NULL,

bool localTime = true,

long utcBias = 0 );

| **Параметр** | **Тип** | **Описание** |
| --- | --- | --- |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| outBuffer | char \* | [вых] Указатель на буфер, в который будет записано текстовое представление временной отметки |
| localTime | bool | [вх] Флаг, указывающий на необходимость вывода текстового представления временной отметки в локальном времени (с учетом временной зоны, установленной в основной ОС) |
| utcBias | long | [вх] В случае, если флаг localTime не установлен — значение (в часах) сдвига времени относительно UTC, с учетом которого необходимо формировать текстовое представление отметки |

Возвращаемое значение

В случае успешного завершения возвращает 8-байтовое беззнаковое значение временной отметки в количестве 100-наносекундных интервалов, прошедших с 1 января 1601 года в UTC (стандартное время в формате ОС Windows). В случае ошибки возвращает значение 0.

Примечание

Установка флага localTime и смещения utcBias влияет только на текстовое представление временной отметки. Возвращаемое функцией числовое значение всегда представлено в UTC. Для преобразования его к локальному времени следует использовать функцию FileTimeToLocalFileTime Windows API.

* + 1. GetFileStdAttributes

Получение флагов атрибутов файла в виде текстовой строки.

Прототип

bool \_\_stdcall GetFileStdAttributes(

FileHandle fileHandle,

char \*outBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| outBuffer | char \* | [вых] Указатель на буфер, в который будет записана текстовая строка с описанием атрибутов файлового объекта |

Возвращаемое значение

В случае успеха возвращает true. В случае ошибки возвращает false, а по адресу outBuffer записывает пустую строку.

Примечание

Возвращаемая строка представляет собой последовательность дефисов, на определенных позициях которой в виде определенных символов отображаются флаги файлового объекта, например «---a---------» («архивный файл») или «rhs----------» («только для чтения», «скрытый», «системный»). Аналогичным образом отображаются флаги файлов в ПО *Total Commander*. Длина возвращаемой текстовой строки и количество представленных в ней флагов зависит от гостевой файловой системы. Во избежание ошибок следует использовать буфер outBuffer размером не менее 20 символов.

* + 1. GetFileHash

Получение хэш-значения заданного потока данных файла.

Прототип

bool \_\_stdcall GetFileHash(

FileHandle fileHandle,

int streamIndex,

unsigned int \*outBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| streamIndex | int | [вх] Индекс потока данных |
| outBuffer | unsigned int \* | [вых] Указатель на буфер, в который будет записано хэш-значение |

Возвращаемое значение

В случае успеха возвращает true. В случае ошибки возвращает false.

Примечание

Расчет хэш-значения производится с использованием алгоритма SHA‑1, поэтому длина выходного буфера должна быть не менее 160 бит (рекомендуется использовать указатель на массив unsigned int[5]). Порядок следования байтов хэш-значения совместим с СПО *AutoExpert*, что позволяет формировать базы данных хэш-значений для этой программы с использованием описываемой библиотеки.

* + 1. SaveFile

Сохранение файлового объекта из гостевой файловой системы в основную файловую систему.

Прототип

bool \_\_stdcall SaveFile(

FileHandle fileHandle,

int streamIndex,

const WCHAR \*outFilePath

);

| **Параметр** | **Тип** | **Описание** |
| --- | --- | --- |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| streamIndex | int | [вх] Индекс потока данных |
| outFilePath | const WCHAR \* | [вх] Путь в основной файловой файловой системе, по которому будут сохранены данные из файлового потока |

Возвращаемое значение

В случае успешного завершения возвращает true. В случае ошибки возвращает false.

Примечание

Для сохранения *всех* потоков данных файла в качестве индекса потока streamIndex следует указать –1. В этом случае каждый поток будет сохранен в виде отдельного файла с добавлением имени потока данных в конце строки: «dbgview.chm.*Zone.Identifier*» (курсивом выделено имя потока). Основной поток будет сохранен с указанным в outFilePath именем файла. Временные отметки создания, последней модификации и последнего доступа создаваемых файлов копируют временные отметки сохраняемого файлового объекта в гостевой файловой системе.

* + 1. SaveFileWithPath

Сохранение файлового объекта из гостевой файловой системы в основную файловую систему с воспроизведением исходной структуры каталогов.

Прототип

bool \_\_stdcall SaveFileWithPath(

FileHandle fileHandle,

const WCHAR \*outPath

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| outPath | const WCHAR \* | [вх] Путь в основной файловой системе, относительно которого будет воспроизведен полный путь к сохраняемому файловому объекту |

Возвращаемое значение

В случае успешного завершения возвращает true. В случае ошибки возвращает false.

Примечание

Временные отметки создания, последней модификации и последнего доступа создаваемых файлов копируют временные отметки сохраняемого файлового объекта в гостевой файловой системе. Временные отметки создаваемых каталогов не модифицируются и соответствуют текущему времени.

* + 1. SetFilePosition

Задание текущего потока данных файла и позиции в нем.

Прототип

bool \_\_stdcall SetFilePosition(

FileHandle fileHandle,

int newStreamIndex,

LONGLONG newPosition,

OriginEnum newOrigin

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| newStreamIndex | int | [вх] Индекс потока данных |
| newPosition | LONGLONG | [вх] Новое значение позиции |
| newOrigin | OriginEnum | [вх] Точка отсчета позиции (п. 3.1.3) |

Возвращаемое значение

Возвращает true при успешном выполнении позиционирования. В остальных случаях возвращает false.

Примечание

Функция используется для произвольного позиционирования в пределах заданного потока данных файла, обычно с целью последующего чтения данных. Для выполнения чтения функцией [ReadFileData](#ReadFileData) (п. 3.4.14) позиция может быть не кратна размеру кластера гостевой файловой системы. Выполняется проверка на возможный выход устанавливаемой позиции за пределы потока данных. С файловым объектом ассоциируется *единственный* управляемый функцией [SetFilePosition](#SetFilePosition) (п. 3.4.13) курсор, который в каждый момент указывает на *один* поток данных и позицию в нем.

* + 1. ReadFileData

Чтение данных из файла.

Прототип

LONGLONG \_\_stdcall ReadFileData(

FileHandle fileHandle,

LONGLONG bytesToRead,

BYTE \*dataBuffer

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |
| bytesToRead | LONGLONG | [вх] Количество байт, подлежащих считыванию |
| dataBuffer | BYTE \* | [вых] Указатель на массив, в который следует производить чтение |

Возвращаемое значение

В случае успешного завершения возвращает количество считанных и записанных в буфер данных (в байтах). В случае ошибки возвращает READ\_ERROR.

Примечание

Чтение начинается с текущей позиции курсора, которую можно установить функцией [SetFilePosition](#SetFilePosition) (п. 3.4.13). В случае успешного завершения чтения позиция курсора сдвигается вперед на bytesToRead байт. Текущий поток данных не изменяется.

* + 1. CloseFileHandle

Закрытие дескриптора файлового объекта.

Прототип

void \_\_stdcall CloseFileHandle(

FileHandle fileHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор объекта файловой  системы |

Возвращаемое значение

Отсутствует.

* 1. Последовательный обход каталогов
     1. CreateDirectoryIteratorHandle

Создание дескриптора последовательного обхода объектов в каталоге.

Прототип

DirectoryIteratorHandle \_\_stdcall CreateDirectoryIteratorHandle(

FileHandle fileHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| fileHandle | FileHandle | [вх] Дескриптор файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает дескриптор последовательного обхода файловых объектов в каталоге. В случае ошибки возвращает DIRECTORY\_ITERATOR\_ERROR.

Примечание

Функция корректно работает только для файловых объектов типа «каталог», поэтому перед ее вызовом рекомендуется проверить тип файлового объекта функцией [FileIsDir](#FileIsDir) (п. 3.4.3).

* + 1. DirFirst

Переход к первому файловому объекту в каталоге.

Прототип

void \_\_stdcall DirFirst(

DirectoryIteratorHandle dirHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |

Возвращаемое значение

Отсутствует.

* + 1. DirNext

Переход к следующему файловому объекту в каталоге.

Прототип

void \_\_stdcall DirNext(

DirectoryIteratorHandle dirHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |

Возвращаемое значение

Отсутствует.

* + 1. DirIsDone

Проверка окончания цикла обхода файловых объектов в каталоге.

Прототип

bool \_\_stdcall DirIsDone(

DirectoryIteratorHandle dirHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |

Возвращаемое значение

Возвращает true при позиционировании на файловый объект, следующий *за последним файловым объектом* каталога, а также в случае, когда каталог является пустым (не содержит объектов). В остальных случаях возвращает false.

* + 1. DirFileIsDir

Определение того, является ли текущий файловый объект каталогом.

Прототип

bool \_\_stdcall DirFileIsDir(

DirectoryIteratorHandle dirHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |

Возвращаемое значение

Если текущий файловый объект является каталогом, возвращает true. В противном случае возвращает false. В случае ошибки возвращает false.

* + 1. GetFileNameW

Получение имени текущего файлового объекта в кодировке UTF‑16.

Прототип

int \_\_stdcall GetFileNameW(

DirectoryIteratorHandle dirHandle,

WCHAR \*outFileName = NULL

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |
| outFileName | WCHAR \* | [вых] Указатель на буфер, в который будет записано имя файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает количество символов в имени файла. В случае ошибки возвращает значение –1.

Примечание

Максимальная длина имени файла в большинстве «настоящих» (не виртуальных) файловых систем не превышает 255 символов. Вместе с тем, во избежание ошибок переполнения следует выделять память для буфера лишь после определения реальной длины имени вызовом функции с пустым (NULL) значением указателя outFileName с дополнительным запасом в один символ (будет использован функцией для записи терминального нуля).

* + 1. GetFilePathW

Получение пути к текущему файловому объекту (не включает имя самого файлового объекта) в кодировке UTF‑16.

Прототип

int \_\_stdcall GetFilePathW(

DirectoryIteratorHandle dirHandle,

WCHAR \*outFilePath = NULL

);

| **Параметр** | **Тип** | **Описание** |
| --- | --- | --- |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |
| outFilePath | WCHAR \* | [вых] Указатель на буфер, в который будет записан путь до файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает количество символов в пути к файловому объекту. В случае ошибки возвращает значение -1.

Примечание

Во избежание ошибок переполнения следует выделять память для буфера лишь после определения реальной длины пути вызовом функции с пустым (NULL) значением указателя outFileName с дополнительным запасом в один символ (будет использован функцией для записи терминального нуля).

* + 1. GetFullPathW

Получение полного пути к текущему файловому объекту в кодировке UTF-16.

Прототип

int \_\_stdcall GetFullPathW(

DirectoryIteratorHandle dirHandle,

WCHAR \*outFullPath = NULL

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |
| outFullPath | WCHAR \* | [вых] Указатель на буфер, в который будет записан полный путь до файлового объекта |

Возвращаемое значение

В случае успешного завершения возвращает количество символов в записи текстового представления полного пути к файлу. В случае ошибки возвращает значение –1.

Примечание

Во избежание ошибок переполнения следует выделять память для буфера лишь после определения реальной длины пути вызовом функции с пустым (NULL) значением указателя outFullPath с дополнительным запасом в один символ (будет использован функцией для записи терминального нуля).

* + 1. CloseDirectoryIterator

Закрытие дескриптора последовательного обхода файловых объектов в каталоге.

Прототип

void \_\_stdcall CloseDirectoryIterator(

DirectoryIteratorHandle dirHandle

);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| dirHandle | DirectoryIteratorHandle | [вх] Дескриптор последовательного обхода каталога |

Возвращаемое значение

Отсутствует.

* 1. Обработка ошибок
     1. HasError

Проверка на наличие ошибок.

Прототип

bool \_\_stdcall HasError();

Возвращаемое значение

В случае наличия ошибок возвращает true. В случае отсутствия ошибок возвращает false.

* + 1. GetLastErrorString, GetLastErrorStringW

Получение текстовой строки, описывающей последнюю произошедшую ошибку.

Прототипы

bool \_\_stdcall GetLastErrorString(char \*outStr);

bool \_\_stdcall GetLastErrorStringW(WCHAR \*outStr);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| outStr | char \*  WCHAR \* | [вых] Указатель на строку, в которую будет записана информация об ошибке в кодировке ANSI (CP-1251) или UTF‑16 |

Возвращаемое значение

В случае наличия ошибок и успешной записи информации о них в буфер возвращает true. В случае отсутствия ошибок возвращает false.

* + 1. PrintDebug

Установка режима вывода отладочной информации.

Прототип

void \_\_stdcall PrintDebug(bool newValue);

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Тип** | **Описание** |
| newValue | bool | [вх] Новое значение параметра |

Возвращаемое значение

Отсутствует.

Примечание

При установке параметра в значение true в случае возникновения ошибок информация о них сразу будет выведена в стандартный поток ошибок.

1. Сообщения

В случае возникновения ошибок (нестандартных ситуаций), которые предусмотрены алгоритмами работы библиотеки и могут быть корректно обработаны вызывающей подпрограммой, функции библиотеки возвращают специальные значения (п. 3.1.7). При включении опции отладочного вывода (функция [PrintDebug](#PrintDebug), п. 3.7.3) краткая информация об ошибках дополнительно выводится в стандартный поток ошибок в виде текстовых сообщений на русском языке.

1. Глоссарий

|  |  |  |
| --- | --- | --- |
| **Краткая характеристика функции** | **Прототип функции** | **Раздел** |
| 1. **Операции с хранилищами** | | |
| * 1. **Начало работы с хранилищем** | | |
| Создание дескриптора хранилища | StorageHandle \_\_stdcall [**CreateStorageHandle**](#CreateStorageHandle)(  const WCHAR \*fileName); | 3.2.1 |
| * 1. **Получение харакеристик хранилища** | |  |
| Получение информации о типе хранилища | StorageType\_\_stdcall [**GetStorageType**](#GetStorageType)(  StorageHandle storageHandle); | 3.2.2 |
| Получение информации о физическом диске | PhysicalDriveInfoStruct \_\_stdcall  [**GetPhysicalDriveInfo**](#GetPhysicalDriveInfo)(  StorageHandle storageHandle); | 3.2.3 |
| Определение количества областей в таблице разделов | int \_\_stdcall [**GetNumberOfPartitions**](#GetNumberOfPartitions)(  StorageHandle storageHandle); | 3.2.4 |
| Получение информации о разделе (области) хранилища информации по порядковому номеру раздела | bool \_\_stdcall [**GetPartitionInfoByNumber**](#GetPartitionInfoByNumber)(  StorageHandle storageHandle,  unsigned int selectedNumber,  PartitionInfoStruct \*outPartitionInfo); | 3.2.5 |
| Определение размера сектора хранилища | DWORD \_\_stdcall [**GetStorageBlockSize**](#GetStorageBlockSize)(  StorageHandle storageHandle); | 3.2.6 |
| Распознавание файловой системы по заданному смещению | FileSystemTypeEnum \_\_stdcall  [**RecognizeFileSystem**](#RecognizeFileSystem)(  StorageHandle storageHandle,  ULONGLONG startOffset); | 3.2.15 |
| * 1. **Последовательный обход блоков данных хранилища** | | |
| Задание размера сектора хранилища | bool \_\_stdcall [**SetStorageBlockSize**](#SetStorageBlockSize)(  StorageHandle storageHandle,  DWORD blockSize); | 3.2.7 |
| Получение дескиптора последовательного обхода | StorageIteratorHandle \_\_stdcall  [**GetStorageIterator**](#GetStorageIterator)(  StorageHandle storageHandle); | 3.2.8 |
| Переход к первому блоку данных | void \_\_stdcall [**StorageBlockFirst**](#StorageBlockFirst)(  StorageIteratorHandle storageIteratorHandle); | 3.2.9 |
| Переход к следующему блоку данных | void \_\_stdcall [**StorageBlockNext**](#StorageBlockNext)(  StorageIteratorHandle storageIteratorHandle); | 3.2.10 |
| Проверка окончания обхода | bool \_\_stdcall [**StorageBlockIsDone**](#StorageBlockIsDone)(  StorageIteratorHandle storageIteratorHandle); | 3.2.11 |
| Закрытие дескриптора последовательного обхода блоков данных | void \_\_stdcall [**CloseStorageIterator**](#CloseStorageIterator)(  StorageIteratorHandle storageIteratorHandle); | 3.2.12 |
| * 1. **Чтение данных хранилища по заданному смещения** | | |
| Задание текущей позиции | bool \_\_stdcall [**SetStoragePosition**](#SetStoragePosition)(  StorageHandle storageHandle,  LONGLONG newPosition,  OriginEnum newOrigin = OriginEnum::Begin); | 3.2.13 |
| Чтение данных из хранилища | LONGLONG \_\_stdcall [**ReadStorageData**](#ReadStorageData)(  StorageHandle storageHandle,  LONGLONG bytesToRead,  BYTE \*dataBuffer); | 3.2.14 |
| * 1. **Завершение работы с хранилищем** | | |
| Закрытие дескриптора хранилища | void \_\_stdcall [**CloseStorageHandle**](#CloseStorageHandle)(  StorageHandle storageHandle); | 3.2.17 |
| 1. **Операции с файловыми системами** | | |
| * 1. **Начало работы с файловой системой** | | |
| Создание дескриптора | FileSystemHandle \_\_stdcall  [**CreateFileSystemHandle**](#CreateFileSystemHandle)(  FileSystemTypeEnum fsType,  StorageHandle storageHandle,  ULONGLONG startOffset,  ULONGLONG diskSize,  WORD sectorSize); | 3.3.1 |
| «Быстрое» создание дескриптора | FileSystemHandle \_\_stdcall  [**FastCreateFileSystemHandle**](#FastCreateFileSystemHandle)(  const WCHAR \*fileName,  ULONGLONG startOffset = 0); | 3.3.2 |
| * 1. **Получение характеристик файловой системы** | | |
| Определение типа файловой системы | FileSystemTypeEnum \_\_stdcall [**GetFileSystemType**](#GetFileSystemType)(  FileSystemHandle fileSystemHandle); | 3.3.3 |
| Получение наименования файловой системы | int \_\_stdcall [**GetFileSystemTypeString**](#GetFileSystemTypeStringW)(  FileSystemHandle fileSystemHandle,  char \*outType); | 3.3.4 |
| int \_\_stdcall [**GetFileSystemTypeStringW**](#GetFileSystemTypeStringW)(  FileSystemHandle fileSystemHandle,  WCHAR \*outType); |
| Получение количества кластеров файловой системы | ULONGLONG \_\_stdcall [**GetNumberOfClusters**](#GetNumberOfClusters)(  FileSystemHandle fileSystemHandle); | 3.3.5 |
| Получение размера файловой системы в байтах | ULONGLONG \_\_stdcall [**GetFileSystemSize**](#GetFileSystemSize)(  FileSystemHandle fileSystemHandle); | 3.3.6 |
| Получение размера в байтах одного кластера файловой системы | DWORD \_\_stdcall [**GetFileSystemClusterSize**](#GetFileSystemClusterSize)(  FileSystemHandle fileSystemHandle); | 3.3.7 |
| * 1. **Чтение кластеров файловой системы методом последовательного обхода** | | |
| Создание дескриптора последовательного обхода кластеров | ClusterIteratorHandle \_\_stdcall  [**GetClusterIterator**](#GetClusterIterator)(  FileSystemHandle fileSystemHandle); | 3.3.8 |
| Переход к первому кластеру | void \_\_stdcall [**ClusterFirst**](#ClusterFirst)(  ClusterIteratorHandle clusterIteratorHandle); | 3.3.9 |
| Переход к следующему кластеру | void \_\_stdcall [**ClusterNext**](#ClusterNext)(  ClusterIteratorHandle clusterIteratorHandle); | 3.3.10 |
| Проверка окончания обхода кластеров | bool \_\_stdcall [**ClusterIsDone**](#ClusterIsDone)(  ClusterIteratorHandle clusterIteratorHandle); | 3.3.11 |
| Определение того, является ли текущий кластер используемым или свободным | bool \_\_stdcall [**ClusterIsUsed**](#ClusterIsUsed)(  ClusterIteratorHandle clusterIteratorHandle); | 3.3.12 |
| Чтение данных из текущего кластера | LONGLONG \_\_stdcall [**GetCurrentCluster**](#GetCurrentCluster)(  ClusterIteratorHandle clusterIteratorHandle,  BYTE \*dataBuffer); | 3.3.13 |
| Закрытие дескриптора последовательного обхода кластеров | void \_\_stdcall [**CloseClusterIterator**](#CloseClusterIterator)(  ClusterIteratorHandle clusterIteratorHandle); | 3.3.14 |
| * 1. **Чтение данных файловой системы по заданному смещению** | | |
| Задание текущей позиции в области памяти, занимаемой файловой системой | bool \_\_stdcall [**SetFileSystemPosition**](#SetFileSystemPosition)(  FileSystemHandle fileSystemHandle,  LONGLONG newPosition,  OriginEnum newOrigin = OriginEnum::Begin); | 3.3.15 |
| Чтение данных из области памяти, занимаемой файловой системой | LONGLONG \_\_stdcall [**ReadFileSystemData**](#ReadFileSystemData)(  FileSystemHandle fileSystemHandle,  ULONGLONG bytesToRead,  BYTE \*dataBuffer); | 3.3.16 |
| Получение данных диапазона кластеров, заданного номером первого кластера и количеством кластеров | LONGLONG \_\_stdcall [**ReadClustersByNumber**](#ReadClustersByNumber)(  FileSystemHandle fileSystemHandle,  ULONGLONG clusterId,  DWORD numberOfClusters,  BYTE \*dataBuffer,  const DWORD bufferSize); | 3.3.17 |
| * 1. **Завершение работы с файловой системой** | | |
| Закрытие дескриптора файловой системы | void \_\_stdcall [**CloseFileSystemHandle**](#CloseFileSystemHandle)(  FileSystemHandle fileSystemHandle); | 3.3.18 |
| 1. **Операции с отдельными файлами и каталогами** | | |
| * 1. **Начало работы с отдельным объектом файловой системы** | | |
| Создание дескриптора файлового объекта по его полному пути в гостевой файловой системе | FileHandle \_\_stdcall [**CreateFileHandle**](#CreateFileHandle)(  FileSystemHandle fileSystemHandle,  const WCHAR \*fullPath); | 3.4.1 |
| Создание дескриптора файлового объекта по его числовому идентификатору | FileHandle \_\_stdcall [**CreateFileHandleByRecordId**](#CreateFileHandleByRecordId)(  FileSystemHandle fileSystemHandle,  ULONGLONG recordId); | 3.4.2 |
| * 1. **Получение характеристик файлового объекта** | | |
| Определение того, является ли файловый объект каталогом | bool \_\_stdcall [**FileIsDir**](#FileIsDir)(  FileHandle fileHandle); | 3.4.3 |
| Получение количества потоков данных файлового объекта | int \_\_stdcall [**GetNumberOfStreams**](#GetNumberOfStreams)(  FileHandle fileHandle); | 3.4.4 |
| Получение количества объектов файловой системы (файлов и каталогов) в каталоге | int \_\_stdcall [**GetNumberOfFiles**](#GetNumberOfFiles)(  FileHandle fileHandle); | 3.4.5 |
| Получение размера заданного потока данных файла | LONGLONG \_\_stdcall [**GetFileDataSize**](#GetFileDataSize)(  FileHandle fileHandle,  int streamIndex); | 3.4.6 |
| Получение временных отметок файлового объекта | ULONGLONG \_\_stdcall [**GetTimeFileCreate**](#GetTimeFileCreate)(  FileHandle fileHandle,  char \*outBuffer = NULL,  bool localTime = true,  long utcBias = 0); | 3.4.7 |
| ULONGLONG \_\_stdcall [**GetTimeFileModify**](#GetTimeFileCreate)(  FileHandle fileHandle,  char \*outBuffer = NULL,  bool localTime = true,  long utcBias = 0); |
| ULONGLONG \_\_stdcall [**GetTimeFileAccess**](#GetTimeFileCreate)(  FileHandle fileHandle,  char \*outBuffer = NULL,  bool localTime = true,  long utcBias = 0); |
| ULONGLONG \_\_stdcall [**GetTimeFileRecordModify**](#GetTimeFileCreate)(  FileHandle fileHandle,  char \*outBuffer = NULL,  bool localTime = true,  long utcBias = 0); |
| Получение флагов атрибутов файла в виде текстовой строки | void \_\_stdcall [**GetFileStdAttributes**](#GetFileStdAttributes)(  FileHandle fileHandle,  char \*outBuffer); | 3.4.8 |
| Получение хэш-значения заданного потока данных файла | bool \_\_stdcall [**GetFileHash**](#GetFileHash)(  FileHandle fileHandle,  int streamIndex,  char \*outBuffer); | 3.4.9 |
| * 1. **Сохранение файлового объекта в основную файловую систему** | | |
| Сохранение файлового объекта из гостевой файловой системы в основную файловую систему | bool \_\_stdcall [**SaveFile**](#SaveFile)(  FileHandle fileHandle,  int streamIndex,  const WCHAR \*outFilePath); | 3.4.11 |
| Сохранение файлового объекта из гостевой файловой системы в основную файловую систему с воспроизведением исходной структуры каталогов | bool \_\_stdcall [**SaveFileWithPath**](#SaveFileWithPath)(  FileHandle fileHandle,  const WCHAR \*outPath); | 3.4.12 |
| * 1. **Чтение данных файла по заданному смещению** | | |
| Задание текущего потока данных файла и позиции в нем | bool \_\_stdcall [**SetFilePosition**](#SetFilePosition)(  FileHandle fileHandle,  int newStreamIndex,  LONGLONG newPosition,  OriginEnum newOrigin); | 3.4.13 |
| Чтение данных из файла | LONGLONG \_\_stdcall [**ReadFileData**](#ReadFileData)(  FileHandle fileHandle,  LONGLONG bytesToRead,  BYTE \*dataBuffer); | 3.4.14 |
| * 1. **Завершение работы с отдельным объектом файловой системы** | | |
| Закрытие дескриптора файлового объекта | void \_\_stdcall [**CloseFileHandle**](#CloseFileHandle)(  FileHandle fileHandle); | 3.4.15 |
| 1. **Последовательный обход каталогов** | | |
| Создание дескриптора последовательного обхода объектов в каталоге | DirectoryIteratorHandle \_\_stdcall  [**CreateDirectoryIteratorHandle**](#CreateDirectoryIteratorHandle)(  FileHandle fileHandle); | 3.5.1 |
| Переход к первому файловому объекту в каталоге | void \_\_stdcall [**DirFirst**](#DirFirst)(  DirectoryIteratorHandle dirHandle); | 3.5.2 |
| Переход к следующему файловому объекту в каталоге | void \_\_stdcall [**DirNext**](#DirNext)(  DirectoryIteratorHandle dirHandle); | 3.5.3 |
| Проверка окончания цикла обхода файловых объектов в каталоге | bool \_\_stdcall [**DirIsDone**](#DirIsDone)(  DirectoryIteratorHandle dirHandle); | 3.5.4 |
| Определение того, является ли текущий файловый объект каталогом | bool \_\_stdcall [**DirFileIsDir**](#DirFileIsDir)(  DirectoryIteratorHandle dirHandle); | 3.5.5 |
| Получение имени текущего файлового объекта | int \_\_stdcall [**GetFileNameW**](#GetFileNameW)(  DirectoryIteratorHandle dirHandle,  WCHAR \*outFileName = NULL); | 3.5.6 |
| Получение пути к текущему файловому объекту | int \_\_stdcall [**GetFilePathW**](#GetFilePathW)(  DirectoryIteratorHandle dirHandle,  WCHAR \*outFilePath = NULL); | 3.5.7 |
| Получение полного пути к текущему файловому объекту | int \_\_stdcall [**GetFullPathW**](#GetFullPathW)(  DirectoryIteratorHandle dirHandle,  WCHAR \*outFullPath = NULL); | 3.5.8 |
| Закрытие дескриптора последовательного обхода файловых объектов в каталоге | void \_\_stdcall [**CloseDirectoryIterator**](#CloseDirectoryIterator)(  DirectoryIteratorHandle dirHandle); | 3.5.9 |
| 1. **Обработка ошибок** | | |
| Проверка на наличие ошибок | bool \_\_stdcall [**HasError**](#HasError)(); | 3.7.1 |
| Получение текстовой строки, описывающей последнюю произошедшую ошибку | bool \_\_stdcall [**GetLastErrorString**](#GetLastErrorString)(  char \*outStr); | 3.7.2 |
| bool \_\_stdcall [**GetLastErrorStringW**](#GetLastErrorString)(  WCHAR \*outStr); |
| Установка режима вывода отладочной информации | void \_\_stdcall [**PrintDebug**](#PrintDebug)(  bool newValue); | 3.7.3 |