

ADSDotNET

Programming guide

ADSDotNET version: 0.1.0
Document version: 1.0

Introduction

ADSDotNET is a library which makes it possible for .NET-based programming languages like C# or Visual Basic .NET to access alternate data streams (ADS). Without this DLL it would be necessary to use P-Invoke or C++/CLI to work with ADS and to benefit from .net-based languages at the same time. ADSdotNET was developed in C++/CLI by the way.

System requirements

- Windows with installed .NET Framework 2.0

Using ADSdotNET

After including the DLL as reference in your Visual Studio project (you can do that in the Solution Explorer) you can see a new entry inside the folder References named ADSdotNET.

From now on you can use a new namespace called `AdsDotNet`. It contains a class called `AdsHandler`. First of all you have to create an object of this class. Every object of type `AdsHandler` points to one file or directory. The methods of these objects can be used for adding/deleting/modifying the alternate data streams of the corresponding file or directory (will be called "root object" in the future). To root objects alternate data streams can be attached.

Next the methods of the class `AdsHandler` will be described. Examples will help to understand the correct usage.

Constructor

Creates a new object of type `AdsHandler` which represents a file or a folder on which the ADS operations can be executed.

C++/CLI declaration: `AdsHandler(String ^filename);`
Parameter: `filename:` Name of an existing file or folder (= name of the root object)

Example 1: `AdsManager adsman = new AdsManager("C:\\texts\\textfile.txt");`
Example 2: `AdsManager adsman = new AdsManager("C:\\texts");`

Note: By passing an empty string - `new AdsManager("");` - a `CommandNotAllowedException` will be thrown.
If the root object does not exist, a `FileNotFoundException` will be thrown.

AddAds

Adds an empty ADS to the root object.

C++/CLI declaration: `void AddAds(String ^adsname);`
Parameter: `adsname:` The name of the ADS which should be created.

Example: `adsman.AddAds("my_stream");`

Note: By passing an empty string - `adsman.AddAds("")` - a `CommandNotAllowedException` will be thrown.
If an ADS with the same name already exists nothing happens.

DeleteAds

Removes (deletes) an ADS from the root object.

C++/CLI declaration: `void DeleteAds(String ^adsname);`

Parameter: `adsname:` The name of the ADS which should be deleted.

Example: `adsman.AddAds("my_stream");`

Note: By passing an empty string - `adsman.AddAds("")` - the root object and all its ADS will be deleted if the root object was a file. Otherwise a `WinAPIException` will be thrown.

AdsExists

Returns `true` if a certain ADS exists, `false` otherwise.

C++/CLI declaration: `bool AdsExists(String ^adsname);`

Parameter: `adsname:` The existence of an ADS with this name will be checked.

Example: `if(adsman.AdsExists("stream334")) { ... }`

Note: By passing an empty string - `adsman.AdsExists("")` -, the method checks the existence of the root object .

GetAdsList

Returns a list of names of all ADS attached to the root object.

C++/CLI declaration: `System::Collections::Generic::List<System::String^> ^GetAdsList();`

Example: `List<String> adslist = adsman.GetAdsList();`

OpenAds

Opens an ADS for reading/writing. A `FileStream` object will be returned so that you can work with ADS in the same manner as you can work with normal files.

C++/CLI declaration: `System::IO::FileStream ^OpenAds(String ^adsname);`

Parameter: `adsname:` Name of the ADS which should be opened.

Example:

```
AdsHandler adsman_folder = new AdsHandler("testfolder");
adsman_folder.AddAds("asd");
System.IO.FileStream fs = adsman_folder.OpenAds("asd");
System.IO.StreamWriter sw = new System.IO.StreamWriter(fs);
sw.WriteLine("Hello ADS!");
sw.Close();
adsman_folder.CloseAds("asd");
```

Note: If the stream does not exist, a `StreamNotFoundException` will be thrown.
By passing an empty string a `FileStream` which points to the root file will be returned.
If the root object is not a file a `WinAPIException` will be thrown.
The `FileStream` will be closed inside the method `CloseAds`. So you do not have to call `fs.Close()`.

AdsIsOpened

Returns `true` if an ADS with the name specified is opened at the moment, `false` otherwise. The stream must have been opened with the method `OpenAds`. So `AdsIsOpened` only returns `true` between the calls `OpenAds` and `CloseAds` (even if the `FileStream` was closed manually with `fs.Close()` before calling the method `CloseAds`).

C++/CLI declaration: `bool AdsIsOpened(String ^adsname);`
Parameter: `adsname:` A stream name.

Example: `if (adsman_folder.AdsIsOpened("stream")) { ... }`

CloseAds

Closes an ADS which was opened by `OpenAds` before. The `FileStream` which was provided by `OpenAds` will be closed by calling this method.

C++/CLI declaration: `void CloseAds(String ^adsname);`
Parameter: `adsname:` The name of an open ADS.

Example: (see `OpenAds`)

GetCompleteFilename

Builds a name like `X:\path\file.txt:streamname` or `X:\path:streamname` out of the filename and a stream name.

C++/CLI declaration: `System::String ^GetCompleteFilename(String ^adsname);`
Parameter: `adsname:` A name which should be used as stream name.

Example: `string filename = adsman.GetCompleteFilename("stream");`

Note: A stream with the name specified does not have to exist.