Stormshield Data Security

Connector C# sample

|  |  |
| --- | --- |
| **Version** | 0.1 |
| **Reference** | STORMSHIELD/SDS/Connector Sample |
| **Date** | 09/03/2017 |
| **Keywords** |  |
| **Summary** | This document describes the sample that show how to create automatically a Virtual Disk on a USB drive using Connector APIs. |

Change log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Authors** | **Proofreaders** | **Description** |
| 0.1 | 08/03/2017 | JPC |  | Initial version. |

Authors / Proofreaders

JPC: Jean-Philippe Ciman

Table of content

[1 Use case scenario 4](#_Toc476826174)

[2 Prerequisites 4](#_Toc476826175)

[3 Program structure 4](#_Toc476826176)

[4 Be notified by Windows when a device is plugged 5](#_Toc476826177)

[5 Interacting with SDS 7](#_Toc476826178)

[5.1 Using Connector 7](#_Toc476826179)

[5.2 Checking if a user is connected 7](#_Toc476826180)

[5.3 Checking if the device is a Virtual Disk 7](#_Toc476826181)

[5.4 Creating the Virtual Disk 8](#_Toc476826182)

[5.5 Mounting the Virtual Disk 8](#_Toc476826183)

[5.6 Formatting the Virtual Disk 8](#_Toc476826184)

[6 References 9](#_Toc476826185)

# Use case scenario

The use case that the provided sample accomplish is what follows:

I want a Virtual Disk to be automatically created when a new USB device is plugged into a machine. The Virtual Disk will have a size calculated proportionally to the available disk space and will be immediately available for usage in Windows Explorer.

If the Virtual Disk already exists on the plugged device, it will simply be mounted to be used right away.

In this use case scenario, the USB device detection has been covered by someone on [Codeguru website](http://www.codeguru.com/columns/dotnet/detecting-usb-devices-using-c.html). As this sample is based on that, the reading of this article is greatly recommended.

# Prerequisites

To build and run the sample provided you will need:

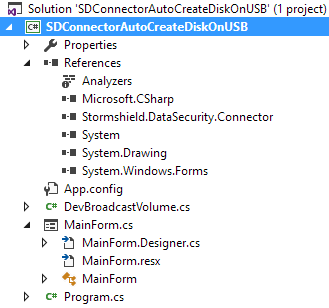
* Visual Studio 2015 or 2017 installed (the [Community Edition](https://www.visualstudio.com/fr/vs/community) is sufficient)
* SDS 9.1.3 installed with Connector module
* A USB thumb drive to be plugged into the computer

# Program structure

The sample is written in C# language and is going to be interpreted by and executed through [.NET runtime](https://msdn.microsoft.com/en-us/library/ff361664.aspx).

[Window Forms](https://msdn.microsoft.com/en-us/library/dd30h2yb.aspx) are used for the user interface. [Windows Presentation Foundation](https://msdn.microsoft.com/en-us/library/ms754130.aspx) could also be used, Connector APIs are agnostic to the user interface libraries.

The Visual Studio solution structure is the following:



Link to Connector APIs

Main program file responsible for creating user interface

Code for user interface, also contains code of sample

Utility class to interface with Win32 APIs

This structure gives you a bird’s eye view of the project.

References

The Stormshield.DataSecurity.Connector reference is what links us with Connector APIs. Concretely speaking, it’s gives Visual Studio the path to the Stormshield.DataSecurity.Connector.dll file (more information in the documentation). This link is required to call Connector APIs and interact with SDS.

Program.cs

This is the main file that Visual Studio created for us (by using a C# project template provided by Visual Studio itself). This file is of no use to fully understand the sample.

MainForm.cs

This is where all happens. This class represents the user interface. For the sake of simplicity, all the code that is not related to the user interface has been put inside this file (creating, formatting and mounting Virtual Disk for example).

DevBroadcastVolume.cs

This file is mandatory to be able to interact with Win32 APIs. It represents the data structure that Win32 APIs will send to our program when the event DBT\_DEVICEARRIVAL is raised (so when device is plugged into the computer).

# Be notified by Windows when a device is plugged

Windows is an event driven programming environment. It is capable of sending events (sort of notifications) to programs that registered to receive it.

In this sample, the event “device arrival” is useful because our program will then know when a user has plugged a device into his computer.

This can be done by overriding the [Window Procedure](https://msdn.microsoft.com/en-us/library/windows/desktop/ms633569.aspx) that each program has. More information about Window Procedure can be found here:

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms633573.aspx>

In C#, we override the Window Procedure with the Control.WndProc method, as seen is the MainForm.cs file:

protected override void WndProc(ref System.Windows.Forms.Message m)

{

base.WndProc(ref m);

...

}

The System.Windows.Forms.Message object then contains all the information we need to intercepts device arrival:

switch (m.Msg)

{

case WM\_DEVICECHANGE:

switch ((int)m.WParam)

{

case DBT\_DEVICEARRIVAL:

...

break;

}

}

We can then use the LParam property of the System.Windows.Forms.Message to have the device information (that is where the DevBroadcastVolume class comes into play):

int devType = Marshal.ReadInt32(m.LParam, 4);

if (devType == DBT\_DEVTYP\_VOLUME)

{

DevBroadcastVolume volume =

(DevBroadcastVolume)Marshal.PtrToStructure(m.LParam, typeof(DevBroadcastVolume));

...

}

Now we have enough information to interact with SDS:

* An USB device has been plugged into the computer
* We know the driver letter (in the DevBroadcast class)

# Interacting with SDS

The method CreateAndMountDiskVolumeIfNecessary is call Connector APIs to interact with SDS.

As a reminder, all what follows in this paragraph is already documented in Connector documentation and code samples. Feel free to dive into it to better understand what is going on.

## Using Connector

The first and mandatory thing to do when you want to call some Connector APIs is to create an API object:

using (API api = new API())

{

...

}

The api variable will then be used to call APIs.

## Checking if a user is connected

We first need to check if a user is connected to its SDS account, otherwise we won’t be able to do anything:

object[] objects = api.Execute("Get-SDSUser");

if (objects == null)

{

AddLog("No user connected to SDS");

}

else

{

...

}

## Checking if the device is a Virtual Disk

It is better to check if the device is not a Virtual Disk one to prevent the creation of Virtual Disks in a loop:

1. Plug a USB device into the computer
2. Device arrival event is triggered
3. Create a Virtual Disk and mount it in Windows Explorer
4. Device arrival event is triggered
5. And so on…

The easiest way to check that is to call Get-SDSDisk and be prepared to catch an exception, indicating that this is not a Virtual Disk (otherwise Virtual Disk information would have been returned):

bool isVirtualDisk = true;

try

{

objects = api.Execute(string.Format("Get-SDSDisk {0}:", driveLetter));

}

catch

{

isVirtualDisk = false;

}

If the device is a Virtual Disk, we stop the treatment right away.

## Creating the Virtual Disk

int vboxSize =

(int)((double)(driveInfo.AvailableFreeSpace / 1024 / 1024) \* vboxSizePercent);

objects =

api.Execute(string.Format("New-SDSDisk '{0}' -Size {1}", vboxPath, vboxSize));

We calculate the size of the Virtual Disk to be a percentage of the available free space on the device (or drive).

The trick here is that the AvailableFreeSpace property is in [bytes](https://msdn.microsoft.com/en-us/library/system.io.driveinfo.availablefreespace.aspx) and we want a value in megabytes because the New-SDSDisk cmdlet is only dealing with that unit (this is why the value is divided twice by 1024).

## Mounting the Virtual Disk

As easy as calling:

objects = api.Execute(string.Format("Mount-SDSDisk '{0}'", vboxPath));

## Formatting the Virtual Disk

To format the newly created drive, we are simply using the old DOS batch command (still available in Windows):

format X: /FS:NTFS /q /y

The command is written to a file with the BAT extension and the batch file is executed (as it would be by just double-clicking on it in Windows Explorer).

# References

C# reference

<https://msdn.microsoft.com/en-us/library/618ayhy6.aspx>

C# tutorial samples

<https://msdn.microsoft.com/en-us/library/aa664425.aspx>

A good C# tutorial (not on MSDN website)

<http://csharp.net-tutorials.com/>

Detecting USB devices in C#

<http://www.codeguru.com/columns/dotnet/detecting-usb-devices-using-c.html>

Visual Studio Community Edition

<https://www.visualstudio.com/vs/community>

Overview of .NET development

<https://msdn.microsoft.com/en-us/library/ff361664.aspx>

Overview of Windows Forms library

<https://msdn.microsoft.com/en-us/library/dd30h2yb.aspx>

Getting started with C# and Visual Studio

<https://msdn.microsoft.com/en-us/library/dd492171.aspx>

About Window Procedures

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms633569.aspx>

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms633573.aspx>

WM\_DEVICECHANGE message

<https://msdn.microsoft.com/fr-fr/library/windows/desktop/aa363480.aspx>

DEV\_BROADCAST\_HDR structure

<https://msdn.microsoft.com/en-us/library/aa363246.aspx>

AvailableFreeSpace property

<https://msdn.microsoft.com/en-us/library/system.io.driveinfo.availablefreespace.aspx>

Batch format command

<https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/format.mspx>