Security Removable Media Manager
(secRMM)

SDK Programmers Guide

Version 9.11.23.0

(October 2023)

Protect your valuable data

If you have any questions regarding your potential use of this material, contact:

Squadra Technologies, LLC
7575 West Washington Ave
Suite 127-252
Las Vegas, NV 89128 USA
www.squadratechnologies.com
email: info@squadratechnologies.com

Refer to our Web site for regional and international office information.

TRADEMARKS

Disclaimer

Squadra Technologies Administrator Guide
Created - March 2011

Contents

Introduction

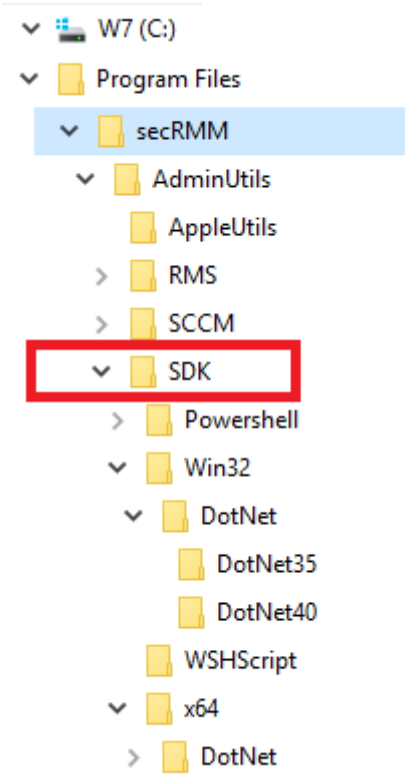
Squadra Technologies *security Removable Media Manager* (**secRMM**) software is Windows security software that runs on your company's workstations and servers. secRMM manages and monitors removable media. In this context, Removable media is defined as external hard disks, USB (flash) drives, smart phones, tablets, SD-Cards, CD-ROM and DVD. Such devices typically use the computers Universal Serial Bus (USB) ports to connect to the computer. Removable media devices are popular because they are very convenient when you want to copy files around or backup data. secRMM allows you to track all write activity to the removable media devices in your computer environment as well as giving you the ability to control (or authorize) who can write to the removable media devices.

The secRMM SDK is intended for software developers and/or IT administrators who want to integrate the secRMM functionality into their software/scripts. The secRMM SDK is comprised of a COM object, a .Net Assembly, a Powershell cmdlet and vbscripts. Which component you use depends on whether you are writing code in the unmanaged or managed (i.e. Microsoft .Net) space or writing scripts (Powershell, vbscript, jscript, etc.). Typically, unmanaged code is written in C++ and managed code is written in C#.

SDK logistics

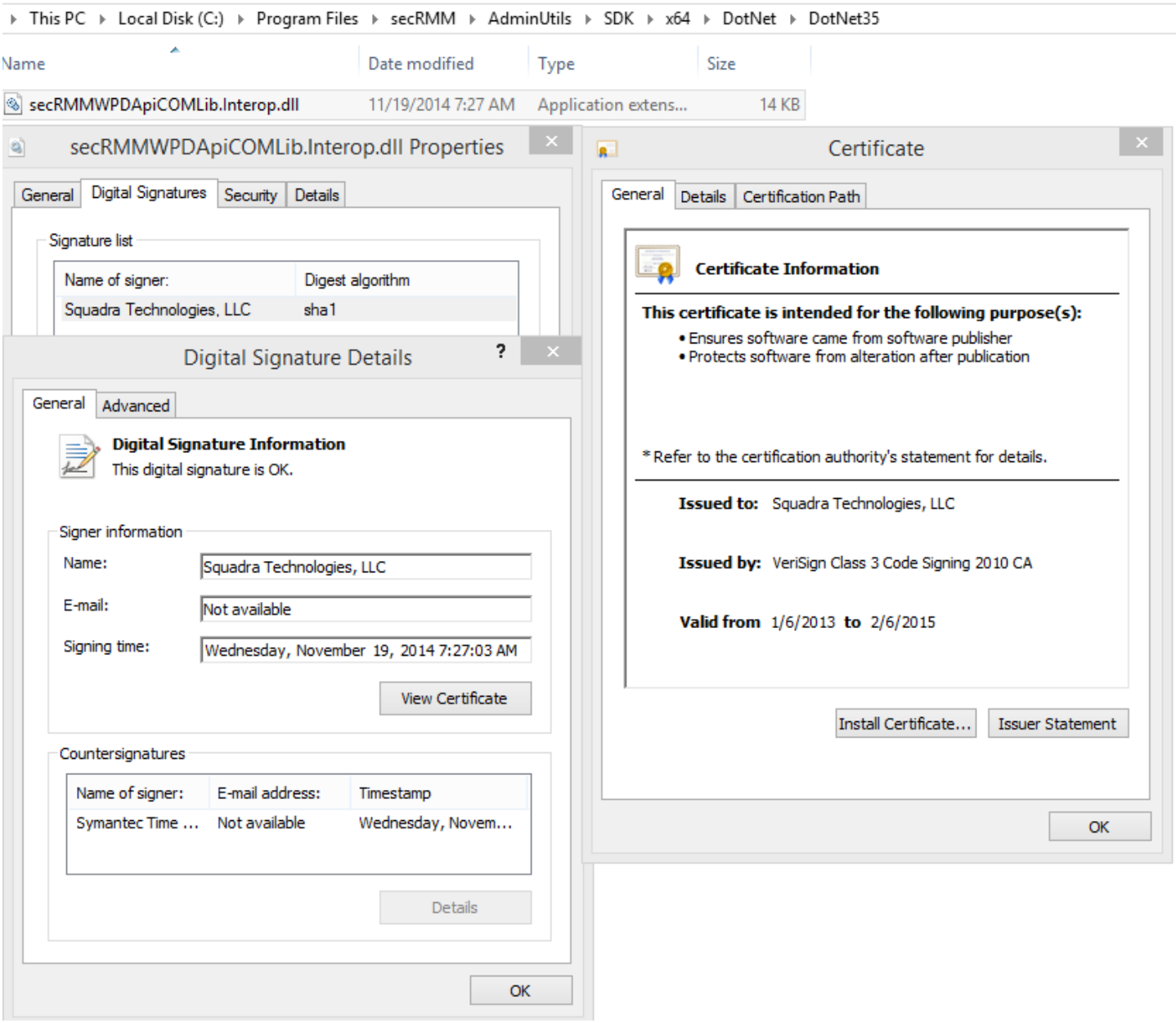
The secRMM SDK is included with the secRMM product. It is not a separate download or installation. When you install the secRMM product onto the Windows computer, a subdirectory will be created under the secRMM installation directory (which is by default at C:\Program Files\secRMM) called AdminUtils\SDK. The SDK comes with 32 and 64 bit components (when required). For the managed (Microsoft .Net framework), there is a component for both the 3.5 and 4.0 versions.

The remainder of this document describles the SDK interface and how to use it from a programming perspective. It is assumed that you have a programming background in Windows managed and/or unmanaged code.



Digital Signature

The secRMM SDK executables are digitally code-signed using the Verisign code-signing certificate assigned to Squadra Technologies as shown in the screen shot below.



Accessing the SDK from Managed code

Runtime considerations

Resolving a .Net assembly location at runtime can be accomplished with different techniques. Squadra recommends using a ResolveEventHandler so that new versions of secRMM can be installed without impacting your code. Please use the following code snippet to accomplish this:

```
using System.Reflection;
using System.IO;
using Microsoft.Win32;
```

```
static void Main(string[] args)
{
AppDomain.CurrentDomain.AssemblyResolve += new System.ResolveEventHandler(ResolveEventHandler);
}

//===================================================================
public static Assembly ResolveEventHandler(object sender, ResolveEventArgs ResolveEventArgs_in)
{
Assembly l_Assembly = null;
string l_DllName = ResolveEventArgs_in.Name.Split(',')[0];
string[] l_ArrayOfDllNames = new string[] { "secRMMWPDApiCOMLib.Interop" };

   if (l_DllName.Equals(l_ArrayOfDllNames[0], StringComparison.CurrentCultureIgnoreCase) == true)
   {
   string l_stringBaseProductInstallPath = GetDllFullPath();
   string l_stringSDKDirectory = l_stringBaseProductInstallPath + "AdminUtils\\SDK\\";

      if (Directory.Exists(l_stringSDKDirectory) == true)
      {
         if (IntPtr.Size == 4)
         {
            l_stringSDKDirectory += "Win32\\DotNet\\DotNet35\\";
         }
         else
         if (IntPtr.Size == 8)
         {
            l_stringSDKDirectory += "x64\\DotNet\\DotNet35\\";
         }
         if (Directory.Exists(l_stringSDKDirectory) == true)
         {
         string l_stringAssemblyName = l_stringSDKDirectory + l_DllName + ".dll";
            try
            {
               l_Assembly = Assembly.LoadFile(l_stringAssemblyName);
            }
            catch (Exception)
            {
               //DebugOut(e.Message);
            }
         }
      }
   }
return (l_Assembly);
}

//===================================================================
private static string GetDllFullPath()
{
string l_stringDllFullPath = null;

   if (l_stringDllFullPath == null)
   {
      try
      {
      string l_stringRegistryKey=@"SOFTWARE\Microsoft\MMC\SnapIns\FX:{4bbd4ebc-d808-4efc-b0a6-
83c62e4ac931}";
      string l_stringRegistryValue = @"ApplicationBase";
```

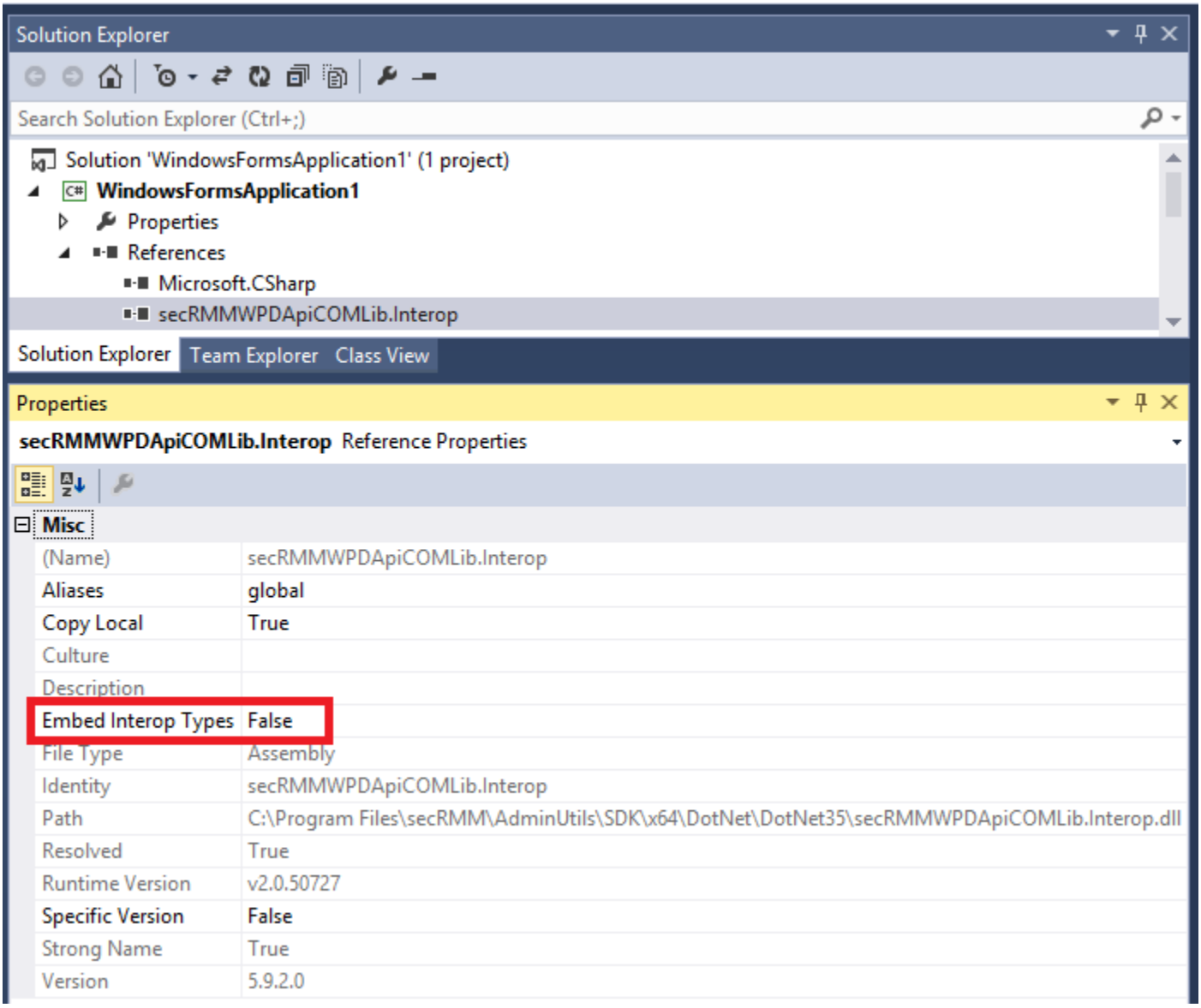
```
      RegistryKey l_RegistryKeyRoot = RegistryKey.OpenRemoteBaseKey(RegistryHive.LocalMachine, "");
      RegistryKey l_RegistryKeyFolder = l_RegistryKeyRoot.OpenSubKey(l_stringRegistryKey);
         l_stringDllFullPath = (string)l_RegistryKeyFolder.GetValue(l_stringRegistryValue);
      }
      catch (Exception)
      {
      }
      finally
      {
      }
   }
return (l_stringDllFullPath);
}
```

Project development considerations

Since the secRMM SDK managed interop assembly is a wrapper around an unmanaged COM component, considerations about the "platform" (i.e. the $(Platform) Visual Studio variable) must be taken into consideration. Microsoft has not really made such a situation convenient for the Visual Studio developer. The common solution seems to be that you need to manually edit the project file (i.e. the *.csproj file) and variable-ize the referenced assembly. You can see the steps here:

http://stackoverflow.com/questions/3832552/conditionally-use-32-64-bit-reference-when-building-in-visual-studio

Also, please note that the reference to the secRMM SDK managed interop assembly must have property "Embedded Interop Types" set to false (as shown in the screen shot below).



Once you have the secRMM SDK assembly reference in your managed project, you can instantiate an instance of the class using the code below:

```
using System;
using System.Runtime.InteropServices;

using secRMMWPDApiCOMLib.Interop;

namespace namespace1
{
    class class1
    {
    secRMMWPDApiCOM2Class m_secRMMWPDApiCOM2Class = new secRMMWPDApiCOM2Class();
```

Now you are ready to call the methods (i.e. the API) associated with the secRMM SDK. The API section below is a reference to each method in the SDK.

API

Reference

The secRMM SDK lets you interact with a mobile device file system.
The following methods are available:

- CopyFileFromDevice
- CopyFileToDevice
- DeleteFileFromDevice
- CreateFolderOnDevice
- EnumerateDevicePath

RequestWriteToken

The RequestWriteToken method generates a unique token for your program/script that you need to pass to the CopyFileToDevice method.

Syntax

```
HRESULT RequestWriteToken([out, retval] VARIANT* pVARIANTWriteToken_out);
```

Parameters

pVARIANTReturnValue_out [out]
This value will be the write token.

CopyFileFromDevice

The CopyFileFromDevice method copies a file from the mobile device to the Windows local file system.

Syntax

```
HRESULT CopyFileFromDevice([in] VARIANT VARIANTObjectDeviceId_in,
                           [in] VARIANT VARIANTObjectStoragePath_in,
                           [in] VARIANT VARIANTLocalFileName_in,
                           [out,retval] VARIANT* pVARIANTReturnValue_out);
```

Parameters

VARIANTObjectDeviceId_in [in]
The mobile device id. The section below titled "Getting the mobile device Id" explains how to retrieve this value.

VARIANTObjectStoragePath_in [in]
The mobile device storage path. The section below titled "Getting the mobile device storage path" explains how to retrieve this value.

VARIANTLocalFileName_in [in]
The complete path of the "Windows file system location" where the mobile device file will be copied to.

pVARIANTReturnValue_out [out]
This value will be a string indicating the success or failure of the copy operation. On success, it will contain a "1", otherwise, it will contain a "0" indicating the operation has failed. If a failure occurs, the calling code will get detailed error information in a com_error object. The section below titled "Getting error information" explains how to retrieve the error information.

CopyFileToDevice

The CopyFileToDevice method copies a file from Windows local file system to the mobile device.

Syntax

```
HRESULT CopyFileToDevice([in] VARIANT VARIANTObjectDeviceId_in,
                         [in] VARIANT VARIANTObjectStoragePath_in,
                         [in] VARIANT VARIANTFileToCopy_in,
                         [in] VARIANT VARIANTWriteToken_in,
                         [out,retval] VARIANT* pVARIANTReturnValue_out);
```

Parameters

VARIANTObjectDeviceId_in [in]
The mobile device id. The section below titled "Getting the mobile device Id" explains how to retrieve this value.

VARIANTObjectStoragePath_in [in]
The mobile device storage path. The section below titled "Getting the mobile device storage path" explains how to retrieve this value.

VARIANTFileToCopy_in [in]
The complete path of the "Windows file system location" of the file you want to copy to the mobile device.

VARIANTWriteToken_in [in]
A token generated by the call to RequestWriteToken method.

pVARIANTReturnValue_out [out]
This value will be a string indicating the success or failure of the copy operation. On success, it will contain a "1", otherwise, it will contain a "0" indicating the operation has failed. If a failure occurs, the calling code will get detailed error information in a com_error object. The section below titled "Getting error information" explains how to retrieve the error information.

DeleteFileFromDevice

The DeleteFileFromDevice method deletes a file or directory from the mobile device. Use caution when calling this method for directories because the delete is recursive and also removes any subdirectories if they exist.

Syntax

```
HRESULT DeleteFileFromDevice([in] VARIANT VARIANTObjectDeviceId_in,
                             [in] VARIANT VARIANTObjectStoragePath_in,
                             [out,retval] VARIANT* pVARIANTReturnValue_out);
```

Parameters

VARIANTObjectDeviceId_in [in]

The mobile device id. The section below titled "Getting the mobile device Id" explains how to retrieve this value.

VARIANTObjectStoragePath_in [in]

The mobile device storage path that is to be deleted. The section below titled "Getting the mobile device storage path" explains how to retrieve this value.

pVARIANTReturnValue_out [out]

This value will be a string indicating the success or failure of the copy operation. On success, it will contain a "1", otherwise, it will contain a "0" indicating the operation has failed. If a failure occurs, the calling code will get detailed error information in a com_error object. The section below titled "Getting error information" explains how to retrieve the error information.

CreateFolderOnDevice

The CreateFolderOnDevice method creates a folder (directory) on the mobile device under the directory specified by VARIANTObjectStoragePath_in.

Syntax

```
HRESULT CreateFolderOnDevice([in] VARIANT VARIANTObjectDeviceId_in,
                             [in] VARIANT VARIANTObjectStoragePath_in,
                             [in] VARIANT VARIANTNewFolderName_in,
                             [out,retval] VARIANT* pVARIANTReturnValue_out);
```

Parameters

VARIANTObjectDeviceId_in [in]

The mobile device id. The section below titled "Getting the mobile device Id" explains how to retrieve this value.

VARIANTObjectStoragePath_in [in]

The mobile device storage path where the new folder is to be created. The section below titled "Getting the mobile device storage path" explains how to retrieve this value.

VARIANTNewFolderName_in [in]

The name of the new folder to be created.

pVARIANTReturnValue_out [out]
This value will be a string indicating the success or failure of the copy operation. On success, it will contain a "1", otherwise, it will contain a "0" indicating the operation has failed. If a failure occurs, the calling code will get detailed error information in a com_error object. The section below titled "Getting error information" explains how to retrieve the error information.

EnumerateDevicePath

The EnumerateDevicePath method returns the contents of a folder (directory) on the mobile device. The term contents here means files and folders.

Syntax

```
HRESULT EnumerateDevicePath([in] VARIANT VARIANTObjectDeviceId_in,
                            [in] VARIANT VARIANTObjectStoragePath_in,
                            [in] VARIANT VARIANTContentType_in,
                            [out,retval] VARIANT* pVARIANTReturnValue_out);
```

Parameters

VARIANTObjectDeviceId_in [in]
The mobile device id. The section below titled "Getting the mobile device Id" explains how to retrieve this value.

VARIANTObjectStoragePath_in [in]
The mobile device storage path where the new folder is to be created. The section below titled "Getting the mobile device storage path" explains how to retrieve this value.

VARIANTContentType_in [in]
An integer value representing:
0=both – return both folders and files
1=folder – return only folders
2=files – return only files

pVARIANTReturnValue_out [out]
This value will be a string containing the data requested. The syntax of the string is:
Name1@InternalId1@[DIR|FILE]~ Name2@InternalId2@[DIR|FILE]...

Below is a C# snippet that will parse the returned string:

```
try
{
string l_stringReturnCode =
   (string)
   m_secRMMWPDApiCOM2Class.EnumerateDevicePath(
      stringDeviceId_in,
      stringStoragePath_in,
      uintContentType_in);

   if (string.IsNullOrEmpty(l_stringReturnCode) == false)
   {
      l_stringArray =
      l_stringReturnCode.Split(new char[] { '~' });
```

```
      Array.Sort(l_stringArray);

      for (int i = 0; i < l_stringArray.Length; i++)
      {
      string[] l_stringArray2 =
         l_stringArray[i].Split(new char[] { '@' });
      string l_stringName = stringArray2[0]; // name of DIR or FILE
      string l_stringId   = stringArray2[1]; // relevant for Android and Windows
      string l_stringType = stringArray2[2]; // "DIR" or "FILE"
         // process data
      }
   }
}
catch (System.Runtime.InteropServices.COMException CE)
{
   ShowCOMException(CE, stringStoragePath_in);
}
```

If a failure occurs, the string will have a length of 0 and the calling code will get detailed error information in a com_error object. The section below titled "Getting error information" explains how to retrieve the error information.

Using the SDK

The sections below give further explaination for some of the parameters used in the secRMM SDK methods.

Getting the mobile device Id

The mobile device Id is a unique identifier to the Windows Operating System represented as a string. The **true** mobile device Id is (although ASCII text characters) a non-friendly readable string. The following examples make this point clear:

Windows:
\\?\USB#VID_0421&PID_0661&MI_00#7&36E73B98&0&0000#{6AC27878-A6FA-4155-BA85-F98F491D4F33}

Android:
\\?\USB#VID_04E8&PID_6860&MS_COMP_MTP&SAMSUNG_ANDROID#7&39000448&0&0000#{6AC27878-A6FA-4155-BA85-F98F491D4F33}

Apple:
\\?\USB#VID_05AC&PID_1294&MI_00#0#{6AC27878-A6FA-4155-BA85-F98F491D4F33}

Blackberry:
\\?\USB#VID_0FCA&PID_8012&MI_02#7&4E1374D&0&0002#{80375827-83B8-4A51-B39B-905FEDD4F118}

While the mobile device string has meaningful data about the device, it is not easy to remember. It is best practice to provide the **true** mobile device Id to the secRMM SDK methods when possible. However, you may also pass in the device serial number or the device "friendly name" as well. Caution should be taken when using the "friendly name" since multiple devices could have the same "friendly name". A safe scenario to use the "friendly name" is when only one mobile device is attached to the Windows Operating System.

Getting the mobile device storage path

A mobile device can have more than one storage object associated with it. Currently, Android, Windows and Blackberry devices allow the end-user to insert a SD-Card into the mobile device. When the device is connected to the Windows Operating System, it will expose both storage objects. secRMM treats each storage object separately. Therefore, the secRMM SDK methods accept the device storage path as a parameter. For the secRMM SDK methods, the device storage path always starts with the diskname followed by the file system path on the disk. It is very similar to how the NTFS file system references storage locations (i.e. <drive name>\temp\subfoler1). On the mobile device though (with the exception of Blackberry devices), the <drive name> is a simple word such as Phone, Table, Card, etc. Currently, when specifying the device storage path, you must use a "\" instead of a "/".

Getting mobile device Ids and storage paths

When you are integrating the secRMM SDK functionality into your program, you will want to retrieve the information about the mobile devices being connected to the Windows Operating System over the USB connection. secRMM makes this very easy to do by integrating the mobile devices into the very popular Microsoft technology called "Windows Management Instrumentation" (or WMI for short). With WMI, you can write a query to see the mobile devices and their properties (i.e. Id, storage paths, friendly names, etc.) or even register with WMI to get events in real-time. To see the secRMM WMI interface in action, please review the vbscript that comes with the secRMM product named:
C:\Program Files\secRMM\AdminUtils\GetWPDDevices.vbs

Getting error information

If a secRMM SDK method call fails, detailed information about the error will be returned in a COM error object. In managed code, a COM error object is wrapped inside of a COMException object. Below is a C# snippet that is calling the secRMM SDK EnumerateDevicePath method. The snippet shows an error handling method in the C# code called ShowCOMException.

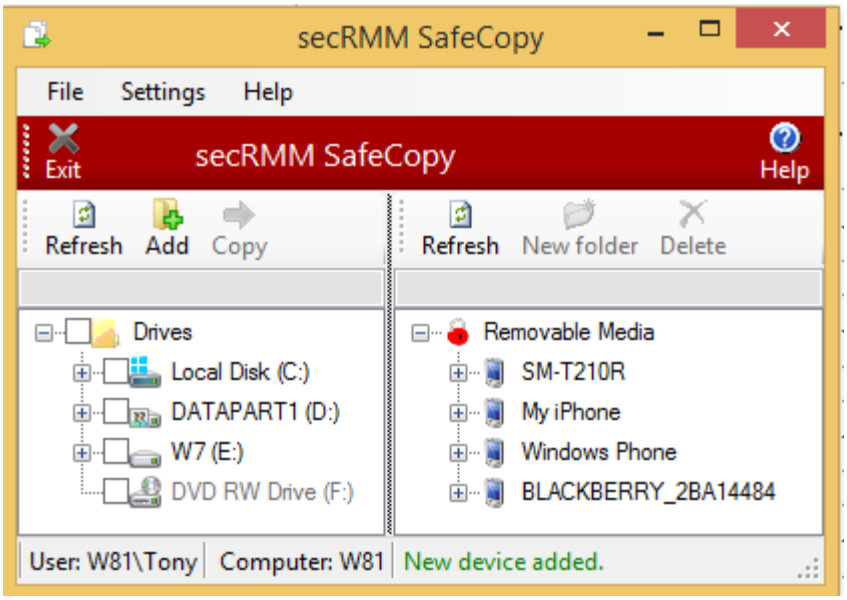
```
try
{
string l_stringReturnCode =
   (string)m_secRMMWPDApiCOM2Class.EnumerateDevicePath(
      stringDeviceId_in,
      stringStoragePath_in,
      uintContentType_in);

   if (string.IsNullOrEmpty(l_stringReturnCode) == false)
   {
      l_stringArray = l_stringReturnCode.Split(new char[] { '~' });
      Array.Sort(l_stringArray);
   }
}
catch (System.Runtime.InteropServices.COMException CE)
{
   ShowCOMException(CE, stringStoragePath_in);
}

//=====================================================================
private void ShowCOMException(
```

```
   System.Runtime.InteropServices.COMException CE_in,
   string stringAdditionalInfo_in)
{
   MessageBox.Show(CE_in.Message + "\n" +
                   (stringAdditionalInfo_in==null?
                   "\n": stringAdditionalInfo_in + "\n\n") +
                   CE_in.Source,
                   "My mobile device program",
                   MessageBoxButtons.OK,
                   MessageBoxIcon.Error);
}
```

secRMM SafeCopy



secRMM ships with a Windows Explorer-like program that is specific to removable media storage. It contains extra security features that are not found in the Windows Operating System. In addition, it contains an automation component that is controlled by the command-line parameters you pass to it. This automation component will (based on the command line parameters) copy a folder of data from the Windows local file system to a mobile device folder. This lets you "batch" a file synch operation from the Windows computer to the mobile device.

If copying data from the Windows computer to the mobile device is your primary programming task, you can consider utilizing the secRMM SafeCopy command-line parameters versus using the secRMM SDK.

secRMM SafeCopy can be called with the following command line parameters:

1. **treeview1** - expand the left tree (Drives) to the node specified by path1

   ex: secRMMSafeCopy treeview1 "C:\Users\Barbara\Pictures\My Tennis Vacation"

2. **treeview2** - expand the right tree (Removable Media) to the node specified by path2

   ex: secRMMSafeCopy treeview2 My iPhone\Internal Storage_Apps\com.squadratechnologies.secRMM.mobileApp\Documents

3. **showcopydetails** - set the "Show details" setting to enabled

   ex: secRMMSafeCopy showcopydetails

4. **practicemode** - set the "Practice mode" setting to enabled

ex: secRMMSafeCopy practicemode

5. **tracemode** - set the "Show trace" setting to enabled

   ex: secRMMSafeCopy tracemode

6. **performcopy** - if parameters treeview1 and treeview2 are specified, start the copy operation automatically

   ex: secRMMSafeCopy treeview1 C:\MyCopyFolder treeview2 MyAndroid\Tablet\Documents performcopy

Android Utilities

secRMM ships with a collection of utility programs for android mobile devices. The following significant operations can be performed with these utilities: app management, device property management and file transfers.

The android utilities are located in the subdirectory AdminUtils\AndroidUtils of the secRMM installation directory.

Apple Utilities

secRMM ships with a collection of utility programs which come from the libimobiledevice web site (please see http://www.libimobiledevice.org/). This powerful utility suite allows you to interact with apple devices from the Windows command-line (or to call from another program). The following significant operations can be performed with the libimobiledevice utilities: activate, backup, diagnose, app management, provisioning profile management and view the devices syslog. In addition, there are two additional utilities called secRMMCopyFileToDevice and secRMMCopyFileFromDevice which perform file copies to/from the apple mobile device and the Windows computer over a USB connection.

Since these programs come from the *nix community, you can get the syntax/paramters of each utility by typing the program name in a CMD window and pressing the enter key.

The apple utilities are located in the subdirectory AdminUtils\AppleUtils of the secRMM installation directory.

Windows Utilities

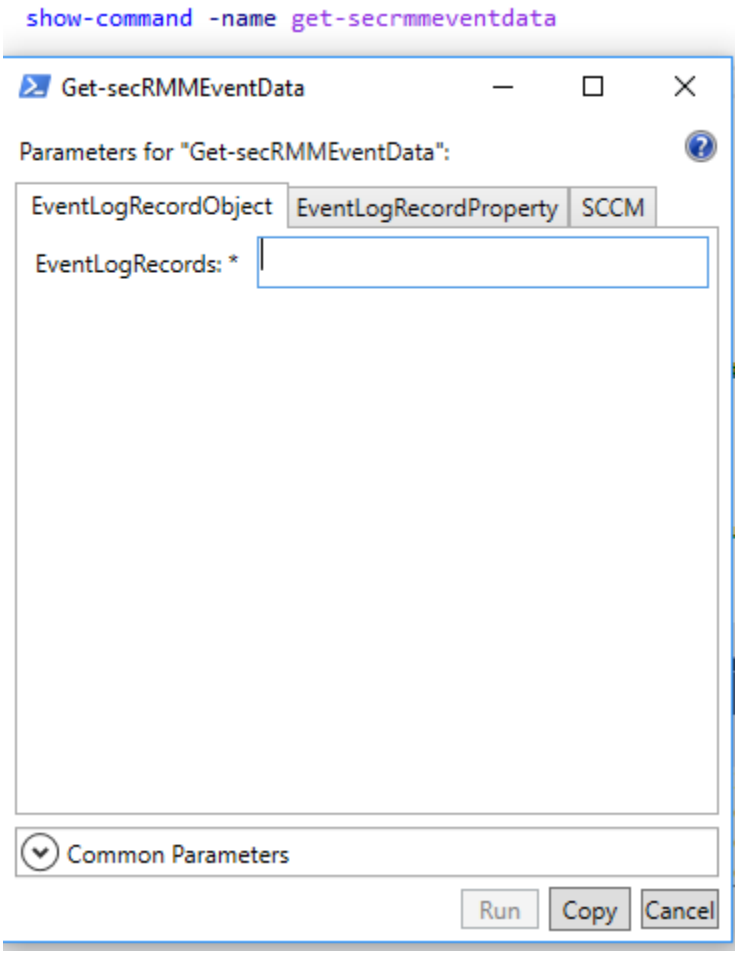
secRMM ships with a collection of utility programs for windows mobile devices. The following significant operations can be performed with these utilities: app management, device property management and file transfers.

The windows utilities are located in the subdirectory AdminUtils\WindowsUtils of the secRMM installation directory.

Powershell

secRMM ships with a Powershell cmdlet that will convert an event from either the secRMM or secRMMCentral event log into an object with properties that represent each secRMM property for the given event id (ex: online, offline, file write completed, authorization failure, etc.). This cmdlet supports the Powershell pipeline. There is a complete example of this in C:\Program Files\secRMM\AdminUtils\SDK\Powershell\GetSecRMMEvents.ps1.

The cmdlet has 3 different Powershell "parametersets" as shown in the screen shot below. Using the SCCM parameter set will read the secRMM events from the SCCM database rather than the secRMM/secRMMCentral event log. To use the SCCM parameter set, be sure that the secRMM property named SCCMConnection is set.



When you install secRMM, the secRMM installation updates the environment variable PSMODULEPATH with the directory to the secRMM cmdlet.

Using Powershell, you can also get and set secRMM properties. There are also Powershell scripts that let you read and write a file to a mobile device.

VBScripts

secRMM ships with several examples of VBScripts in the SDK. The scripts in the AdminUtils are also written in VBScript. The VBScripts in the SDK are functionaly equivalent to the scripts in the Powershell section of the SDK so you can compare them (for example if you are just getting familiar with Powershell).

Contacting Squadra Technologies Support

Squadra Technologies Support is available to customers who have purchased a commercial version of secRMM and have a valid maintenance contract or who are in a trial mode of the product.

When you contact Support please include the following information:

1. The version of secRMM you have installed.
2. The Windows versions you have installed: XP, 2003 Server, 2008 Server R2, Vista, Windows 7, etc.
3. Whether the Windows Operating System is 32bit or 64bit.
4. The specific issue you are contacting support for.

About Squadra Technologies, LLC.

Squadra Technologies delivers innovative products that help organizations get more data protection within the computer infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Squadra Technologies is helping customers worldwide.

Contacting Squadra Technologies, LLC.

Phone	562.221.3079 (United States and Canada)
Email	info@squadratechnologies.com
Mail	Squadra Technologies, LLC. World Headquarters 7575 West Washington Ave. Suite 127-252 Las Vegas, NV 89128 USA
Web site	http://www.squadratechnologies.com/