How can I get a signature for a Windows system that will remain unchanged even if the user reinstalls Windows?

devblogs.microsoft.com/oldnewthing/20180131-00

January 31, 2018

Raymond Chen

The `SystemIdentification` runtime class (introduced in the Windows 10 Anniversary Update) gives you a signature for a Windows system that will remain unchanged even if the user reinstalls Windows. There are some caveats, though.

To obtain a value that is consistent across reinstalls of Windows, the method uses the Trusted Platform Module (TPM), or if a TPM is not available, the Unified Extensible Firmware Interface (UEFI). If neither is available, then starting in the Fall Creators Update, the method creates a unique ID and saves it in the registry. The registry value is preserved across upgrades, but is lost if the user performs a clean install of Windows. You can use the `Source` property to determine how the signature was generated.

The value you receive is specific to the publisher specified in your application manifest. If you are a classic Win32 app with no manifest, then the system will use a generic "publisher" that is used for all publisher-less apps. Signatures generated for apps with the same publisher will match. Signatures generated for apps with different publishers will not match.

Here's some sample code:

```
// JavaScript

var buffer = Windows.System.Profile.SystemIdentification.
                                      getSystemIdForPublisher();
var id  = buffer.id;
var asHex = Windows.Security.Cryptography.CryptographicBuffer.
                                      encodeToHexString(id);
var asBase64 = Windows.Security.Cryptography.CryptographicBuffer.
                                      encodeToBase64String(id);

// C#

var buffer = Windows.System.Profile.SystemIdentification.
                                      GetSystemIdForPublisher();
var id = buffer.Id;
var asHex = Windows.Security.Cryptography.CryptographicBuffer.
                                      EncodeToHexString(id);
var asBase64 = Windows.Security.Cryptography.CryptographicBuffer.
                                      EncodeToBase64String(id);

// C++/CX

auto buffer = Windows::System::Profile::SystemIdentification::
                                      GetSystemIdForPublisher();
auto id = buffer->Id;
auto asHex = Windows::Security::Cryptography::CryptographicBuffer::
                                      EncodeToHexString(id);
auto asBase64 = Windows::Security::Cryptography::CryptographicBuffer::
                                      EncodeToBase64String(id);
// C++/WinRT

using namespace winrt;

auto buffer = Windows::System::Profile::SystemIdentification::
                                      GetSystemIdForPublisher();
auto id = buffer.Id();
auto asHex = Windows::Security::Cryptography::CryptographicBuffer::
                                      EncodeToHexString(id);
auto asBase64 = Windows::Security::Cryptography::CryptographicBuffer::
                                      EncodeToBase64String(id);

// Raw C++ with WRL

using namespace ABI::Windows::Storage::Streams;
using namespace ABI::Windows::System::Profile;
using namespace Microsoft::WRL;
using namespace Microsoft::WRL::Wrappers;

ComPtr<ISystemIdentificationStatics> systemIdStatics;
RoGetActivationFactory(HStringReference(
    RuntimeClass_Windows_System_Profile_SystemIdentification).Get(),
    IID_PPV_ARGS(&systemIdStatics));
```

```
ComPtr<ISystemIdentificationInfo> info;
systemIdStatics->GetSystemIdForPublisher(&info);

ComPtr<IBuffer> id;
info->get_Id(&id);

ComPtr<ICryptographicBufferStatics> cryptoBufferStatics;
RoGetActivationFactory(HStringReference(
    RuntimeClass_Windows_Security_Cryptography_CryptographicBuffer).Get(),
    IID_PPV_ARGS(&cryptoBufferStatics));

HString asHex;
cryptoBufferStatics->EncodeToHexString(id.Get(),
                                       asHex.GetAddressOf());

HString asBase64;
cryptoBufferStatics->EncodeToBase64String(id.Get(),
                                          asBase64.GetAddressOf());
```

If you want to operate with the raw bytes instead of just encoding them into hex or base64, you can use the `IBufferByteAccess` interface or the `CryptographicBuffer.CopyToByteArray` method.

Raymond Chen

Follow