
DeviceId

A simple library providing functionality to generate a ‘device ID’ that can be used to uniquely identify a computer.

Quickstart

What packages are needed?

As of version 6, the packages have been split up so that users can pick-and-choose what they need, without having to pull down unnecessary references that they won’t use:

- The main DeviceId package contains the core functionality and a number of cross-platform components.
- The DeviceId.Windows package adds a few Windows-specific components.
- The DeviceId.Windows.Wmi package adds even more Windows-specific components, using WMI.
- The DeviceId.Windows.Mmi package adds the same components as above, but using MMI instead of WMI for those instances where WMI isn’t appropriate (such as where no .NET Framework is present on the machine).
- The DeviceId.Linux package adds a few Linux-specific components.
- The DeviceId.Mac package adds a few Mac-specific components.
- The DeviceId.SqlServer package adds support for generating a database ID for SQL Server databases.

You can pick-and-choose which packages to use based on your use case.

For a standard Windows app, the recommended packages are: `DeviceId` and `DeviceId.Windows`. If you want some extra advanced components you can also add `DeviceId.Windows.Wmi`.

```
1 PM> Install-Package DeviceId
2 PM> Install-Package DeviceId.Windows
```

Building a device identifier

Use the `DeviceIdBuilder` class to build up a device ID.

Here’s a Windows-specific device ID, using the `DeviceId.Windows` package to get the built-in Windows Device ID.

```
1 string deviceId = new DeviceIdBuilder()
2     .OnWindows(windows => windows.AddWindowsDeviceId())
3     .ToString();
```

Here's a simple cross-platform one, using only the `DeviceId` package, which is valid for both version 5 and version 6 of the library:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddOsVersion()
4     .AddFileToken("example-device-token.txt")
5     .ToString();
```

Here's a more complex device ID, making use of some of the advanced components from the `DeviceId.Windows.Wmi` (or `DeviceId.Windows.Mmi`) package:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddOsVersion()
4     .OnWindows(windows => windows
5         .AddProcessorId()
6         .AddMotherboardSerialNumber()
7         .AddSystemDriveSerialNumber())
8     .ToString();
```

Here's a complex cross-platform device ID, using `DeviceId.Windows.Wmi`, `DeviceId.Linux`, and `DeviceId.Mac`:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddOsVersion()
4     .OnWindows(windows => windows
5         .AddProcessorId()
6         .AddMotherboardSerialNumber()
7         .AddSystemDriveSerialNumber())
8     .OnLinux(linux => linux
9         .AddMotherboardSerialNumber()
10        .AddSystemDriveSerialNumber())
11    .OnMac(mac => mac
12        .AddSystemDriveSerialNumber()
13        .AddPlatformSerialNumber())
14    .ToString();
```

You can also generate a unique identifier for a database instance. Currently, only SQL Server is supported, but more may be added if there is demand and/or community support:

```
1 using SqlConnection connection = new SqlConnection(connectionString);
2 connection.Open();
3 string databaseId = new DeviceIdBuilder()
```

```
4     .AddSqlServer(connection, sql => sql
5         .AddServerName()
6         .AddDatabaseName()
7         .AddDatabaseId())
8     .ToString();
```

What can you include in a device identifier

The following extension methods are available out of the box to suit some common use cases:

From `DeviceId`:

- `AddUserName` adds the current user's username to the device identifier.
- `AddMachineName` adds the machine name to the device identifier.
- `AddOsVersion` adds the current OS version to the device identifier. Note: This uses `Environment.OSVersion`, so if you're targeting older .NET Framework versions, you'll get different values compared to when you target more modern versions of .NET (.NET Core, .NET 5, .NET 6, and anything later than that).
- `AddMacAddress` adds the MAC address to the device identifier.
- `AddFileToken` adds a unique token stored in a file to the device identifier. The file is created if it doesn't already exist. Fails silently if no permissions available to access the file.

From `DeviceId.Windows`:

- `AddWindowsDeviceId` adds the Windows Device ID (also known as Machine ID or Advertising ID) to the device identifier. This value is the one displayed as "Device ID" in the Windows Device Specifications UI.
- `AddWindowsProductId` adds the Windows Product ID to the device identifier. This value is the one displayed as "Product ID" in the Windows Device Specifications UI.
- `AddRegistryValue` adds a specified registry value to the device identifier.
- `AddMachineGuid` adds the machine GUID from `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography` to the device identifier.

From `DeviceId.Windows.Wmi` and `DeviceId.Windows.Mmi`:

- `AddMacAddressFromWmi` / `AddMacAddressFromMmi` adds the MAC address to the device identifier. These use the improved query functionality from WMI/MMI to provide additional functionality over the basic `AddMacAddress` method (such as being able to exclude non-physical device).
- `AddProcessorId` adds the processor ID to the device identifier.
- `AddSystemDriveSerialNumber` adds the system drive's serial number to the device identifier.

-
- `AddMotherboardSerialNumber` adds the motherboard serial number to the device identifier.
 - `AddSystemUuid` adds the system UUID to the device identifier.

From `DeviceId.Linux`:

- `AddSystemDriveSerialNumber` adds the system drive's serial number to the device identifier.
- `AddMotherboardSerialNumber` adds the motherboard serial number to the device identifier.
- `AddMachineId` adds the machine ID (from `/var/lib/dbus/machine-id` or `/etc/machine-id`) to the device identifier.
- `AddProductUuid` adds the product UUID (from `/sys/class/dmi/id/product_uuid`) to the device identifier.
- `AddCpuInfo` adds CPU info (from `/proc/cpuinfo`) to the device identifier.
- `AddDockerContainerId` adds the Docker container identifier (from `/proc/1/cgroup`) to the device identifier.

From `DeviceId.Mac`:

- `AddSystemDriveSerialNumber` adds the system drive's serial number to the device identifier.
- `AddPlatformSerialNumber` adds IOPlatformSerialNumber to the device identifier.

From `DeviceId.SqlServer`:

- `AddServerName` adds the server name to the device identifier.
- `AddDatabaseName` adds the server name to the device identifier.
- `AddDatabaseId` adds the database ID to the device identifier.
- `AddServerProperty` adds a specified server property to the device identifier.
- `AddServerProperty` adds a specified extended property to the device identifier.

Dealing with MAC Address randomization and virtual network adapters Non physical network adapters like VPN connections tend not to have fixed MAC addresses. For wireless (802.11 based) adapters hardware (MAC) address randomization is frequently applied to avoid tracking with many modern operating systems support this out of the box. This makes wireless network adapters bad candidates for device identification.

Using the cross-platform `AddMacAddress`, you can exclude wireless network adapters like so:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMacAddress(excludeWireless: true)
```

```
3     .ToString();
```

If you're on Windows, you can also exclude non-physical adapters using the `DeviceId.Windows.Wmi` or `DeviceId.Windows.Mmi` packages like so:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMacAddress(excludeWireless: true)
3     .OnWindows(windows => windows
4         .AddMacAddressFromWmi(excludeWireless: true, excludeNonPhysical
5             : true)
6         .ToString())
```

Controlling how the device identifier is formatted

Use the `UseFormatter` method to set the formatter:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddOsVersion()
4     .UseFormatter(new HashDeviceIdFormatter(() => SHA256.Create(), new
5         Base32ByteArrayEncoder()))
6     .ToString();
```

The “default” formatters are available in `DeviceIdFormatters` for quick reference. The default formatter changed between version 5 and version 6 of the library. If you're using version 6 but want to revert to the version 5 formatter, you can do so:

```
1 string deviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddOsVersion()
4     .UseFormatter(DeviceIdFormatters.DefaultV5)
5     .ToString();
```

For more advanced usage scenarios, you can use one of the out-of-the-box implementations of `IDeviceIdFormatter` in the `DeviceId.Formatters` namespace, or you can create your own.

- `StringDeviceIdFormatter` - Formats the device ID as a string containing each component ID, using any desired component encoding.
- `HashDeviceIdFormatter` - Formats the device ID as a hash string, using any desired hash algorithm and byte array encoding.
- `XmlDeviceIdFormatter` - Formats the device ID as an XML document, using any desired component encoding.

There are a number of encoders that can be used to customize the formatter. These implement

[IDeviceIdComponentEncoder](#) and [IByteArrayEncoder](#) and are found in the [DeviceId.Encoders](#) namespace.

- [PlainTextDeviceIdComponentEncoder](#) - Encodes a device ID component as plain text.
- [HashDeviceIdComponentEncoder](#) - Encodes a device ID component as a hash string, using any desired hash algorithm.
- [HexByteArrayEncoder](#) - Encodes a byte array as a hex string.
- [Base32UrlByteArrayEncoder](#) - Encodes a byte array as a base 64 url-encoded string.
- [Base64ByteArrayEncoder](#) - Encodes a byte array as a base 64 string.
- [Base64UrlByteArrayEncoder](#) - Encodes a byte array as a base 64 url-encoded string.

Supporting/validating multiple device ID formats with backwards compatibility

Let's say you shipped an app, and were using DeviceId to perform license checks. You may have done something like:

```
1 var currentDeviceId = new DeviceIdBuilder()
2     .AddMachineName()
3     .AddUserName()
4     .AddMacAddress()
5     .ToString();
6
7 var savedDeviceIdFromLicenseFile = ReadDeviceIdFromLicenseFile();
8
9 var isLicenseValid = currentDeviceId == savedDeviceIdFromLicenseFile;
```

Say you now want to release a new version of your app, and want to change how new device identifiers are generated (maybe just use MAC address and a file token), but you don't want to invalidate every single license file that currently exists. In other words, you want backwards compatible device ID validation.

In the latest version of DeviceId, you can use the [DeviceIdManager](#) to do so:

```
1
2 var deviceIdManager = new DeviceIdManager()
3     .AddBuilder(1, builder => builder
4         .AddMachineName()
5         .AddUserName()
6         .AddMacAddress())
7     .AddBuilder(2, builder => builder
8         .AddMacAddress()
9         .AddFileToken(TokenFilePath));
10
11 var savedDeviceIdFromLicenseFile = ReadDeviceIdFromLicenseFile();
12
```

```
13 var isValid = deviceIdManager.Validate(
    savedDeviceIdFromLicenseFile);
```

The device ID manager will work out which builder to use, and generate the current device ID in the correct format so that it can be sensibly compared to the device ID being validated.

Note that this functionality all works well but I'm not entirely happy with the naming or the API. I've currently built it so that there are no breaking changes for v6. In the future (v7 for example) I may rename some classes and break the API. In any case though I will keep the functionality, so it's safe to use this stuff.

Migration Guide 5.x -> 6.x

There were a few breaking changes going from v5 to v6.

- As mentioned above in the “What packages are needed” section, DeviceId was split into multiple packages, so you may need to add a reference to the packages for your platform (such as `DeviceId.Windows`, `DeviceId.Windows.Wmi`, `DeviceId.Linux`, etc.).
- As mentioned above in the “Controlling how the device identifier is formatted” section, the default formatter changed between version 5 and version 6. If you're using version 6 but want to revert to the version 5 formatter, you can do so via `.UseFormatter(DeviceIdFormatters.DefaultV5)`
- Some methods have been renamed or restricted to certain platforms. You can inspect the version 5.x methods and choose the corresponding new OS-specific methods. Note that these still may not be backwards compatible with a v5 device identifier due to the changes in the component names. Remember, if something is missing in v6 that you had in v5, you can always re-add it yourself via a custom component. Here are some examples of changes:

```
1 // V5:
2 builder.AddOSInstallationID();
3
4 // V6:
5 builder.OnWindows(x => x.AddMachineGuid())
6           .OnLinux(x => x.AddMachineId())
7           .OnMac(x => x.AddPlatformSerialNumber());
```

```
1 // V5:
2 builder.AddMotherboardSerialNumber();
3
4 // V6:
5 builder.OnWindows(x => x.AddMotherboardSerialNumber())
6           .OnLinux(x => x.AddMotherboardSerialNumber());
7           // not available on Mac
```

```
1 // V5:
2 builder.AddSystemUUID();
3
4 // V6:
5 builder.OnWindows(x => x.AddSystemUuid())
6     .OnLinux(x => x.AddProductUuid());
7     // not available on Mac
```

```
1 // V5:
2 builder.AddSystemUUID();
3
4 // V6:
5 builder.OnWindows(x => x.AddProcessorId())
6     .OnLinux(x => x.AddCpuInfo());
7     // not available on Mac
```

Strong naming

From version 5 onwards, the assemblies in this package are strong named for the convenience of those users who require strong naming. Please note, however, that the key files are checked in to this repository. This means that anyone can compile their own version and strong name it with the original keys. This is a common practice with open source projects, but it does mean that you shouldn't use the strong name as a guarantee of security or identity.

License and copyright

Copyright Matthew King 2015-2021. Distributed under the MIT License. Refer to license.txt for more information.

Support DeviceId

Community support is greatly appreciated. You can help in the following ways:

- Tackle some of our open issues that are flagged as 'help wanted'
- Become a sponsor on GitHub
- Buy me a coffee