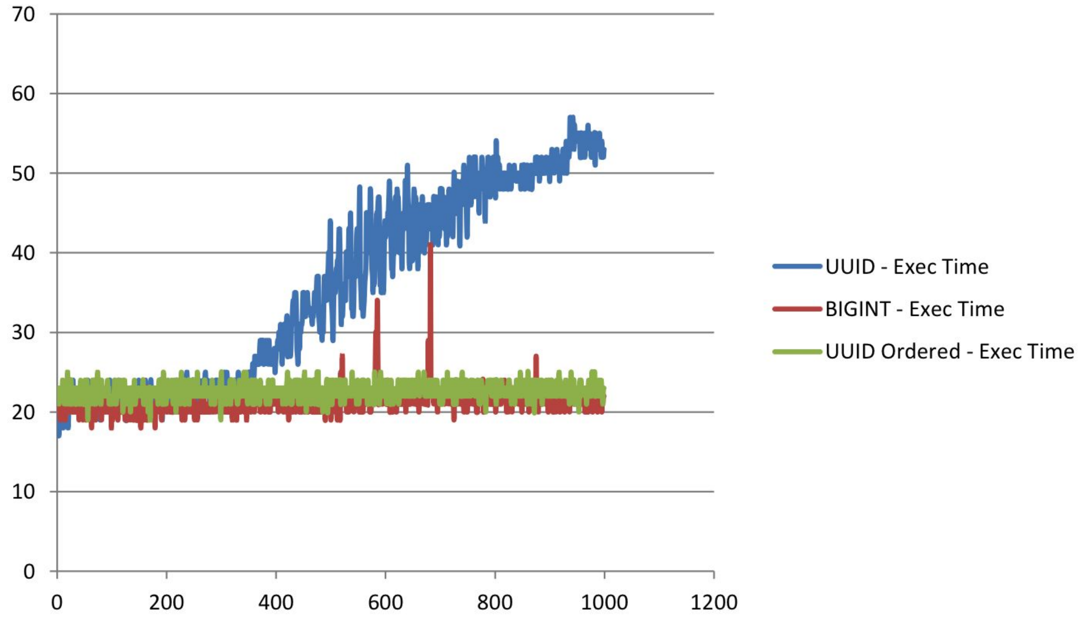
Uuid

Большая история маленькой структуры

Long story short

https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/

Time taken, seconds



Number of inserts x 25,000

Dodo.Tools

Заходи – не бойся,
выходи – не плачь

Dodo.Tools: class

```
namespace Dodo.Tools.Types
{
    // ...
    public class UUId
    {
        // ...
    }
}
```

Dodo.Tools: string

```
namespace Dodo.Tools.Types
{
    // …
    public class UUId
    {
        private readonly string _uuid;
        // …
    }
}
```

Dodo.Tools: аллокации

```
// ...
public UUId(string uuid)
{
    // ...
    if (!IsGuid(uuid))
    {
        throw new ArgumentException("UUId must have the same characters like guid");
    }
    _uuid = uuid.ToUpper();
}

public static bool IsGuid(string value)
{
    Guid x;
    return Guid.TryParse(value, out x);
}
```

Dodo.Tools: ещё аллокации

```
public UUId(byte[] bytes)
{
    // ...
    var val = ByteArrayToString(bytes);
    // ...
    _uuid = val;
}
// ...
private string ByteArrayToString(byte[] bytes)
{
    var hex = BitConverter.ToString(bytes);
    return hex.Replace("-", "");
}
```

Dodo.Tools: Gold

```
public Byte[] ToByteArray()
{
    if (_uuid.Length % 2 != 0)
    {
        throw new ArgumentException("hexString must have an even length");
    }
    Byte[] bytes = new Byte[_uuid.Length / 2];
    for (Int32 i = 0; i < bytes.Length; i++)
    {
        String currentHex = _uuid.Substring(i * 2, 2);
        bytes[i] = Convert.ToByte(currentHex, 16);
    }
    return bytes;
}
```

Dodo.Tools: Platinum

```
private static Func<Guid> GenerateSequentialGuid;
public static UUId NewUUId() => new UUId(GenerateSequentialGuid());
private UUId(Guid guid) { _uuid = GetOrderedUUId(guid).ToUpper(); }

private static string GetOrderedUUId(Guid guid)
{
    var g = guid.ToString();

    return string.Concat(
        g.Substring(24),
        g.Substring(19, 4),
        g.Substring(14, 4),
        g.Substring(9, 4),
        g.Substring(0, 8));
}
```

Dodo.Tools: Performance

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_NewUUId	447.7 ns	1.99 ns	1.86 ns	0.1545	-	-	728 B
dodo_CtorByteArray	471.91 ns	5.44 ns	5.09 ns	0.0486	-	-	232 B
dodo_CtorString	151.7 ns	1.54 ns	1.44 ns	0.0050	-	-	24 B
dodo_ToString	0.4915 ns	0.0454 ns	0.0402 ns	-	-	-	-
dodo_GetHashCode	20.9091 ns	0.1695 ns	0.1585 ns	-	-	-	-
dodo_ToByteArray	587.5497 ns	9.2041 ns	8.6095 ns	0.1173	-	-	552 B

Before we go

.NET Core/Standard/Framework
&
Endianness

.NET Framework

"Given many of the API additions in ***.NET Standard 2.1 require runtime changes*** *in order to be meaningful,* ***.NET Framework 4.8 will remain on .NET Standard 2.0*** *rather than implement .NET Standard 2.1. .NET Core 3.0 as well as upcoming versions of Xamarin, Mono, and Unity will be updated to implement .NET Standard 2.1."*
https://devblogs.microsoft.com/dotnet/announcing-net-standard-2-1/

"Many of the C# 8.0 language features have platform dependencies. ***Async streams, indexers and ranges all rely on new framework types that will be part of .NET Standard 2.1*** *.NET Core 3.0 as well as Xamarin, Unity and Mono will all implement .NET Standard 2.1, but .NET Framework 4.8 will not. This means that the types required to use these features won't be available on .NET Framework 4.8. Likewise,* ***default interface member implementations rely on new runtime enhancements, and we will not make those in the .NET Runtime 4.8 either.****"*
https://devblogs.microsoft.com/dotnet/building-c-8-0/

.NET Standard

.NET Standard	**2.0**	**2.1**
.NET Core	2.0	3.0
.NET Framework	4.7.2	**N/A**
Mono	5.4	6.4
Xamarin.iOS	10.14	12.16
Xamarin.Mac	3.8	5.16
Xamarin.Android	8.0	10.0
UWP	10.0.16299	TBD
Unity	2018.1	TBD

ASP.NET Core: internals

```
public class KestrelServer : IServer
{
    public async Task StartAsync<TContext>(
        IHttpApplication<TContext> application,
        CancellationToken cancellationToken)
    {
        try
        {
            if (!BitConverter.IsLittleEndian)
            {
                throw new PlatformNotSupportedException(CoreStrings.BigEndianNotSupported);
            }
            // … some code
```

https://github.com/dotnet/aspnetcore/blob/v3.1.1/src/Servers/Kestrel/Core/src/KestrelServer.cs#L108-L111

Conclusions

Dodo.Tools: useless

.NET Framework: legacy

.NET Standard: no sense

Big Endian: unsupported

.NET Core: This is the way

Uuid

Что ты такое?

Uuid: RFC

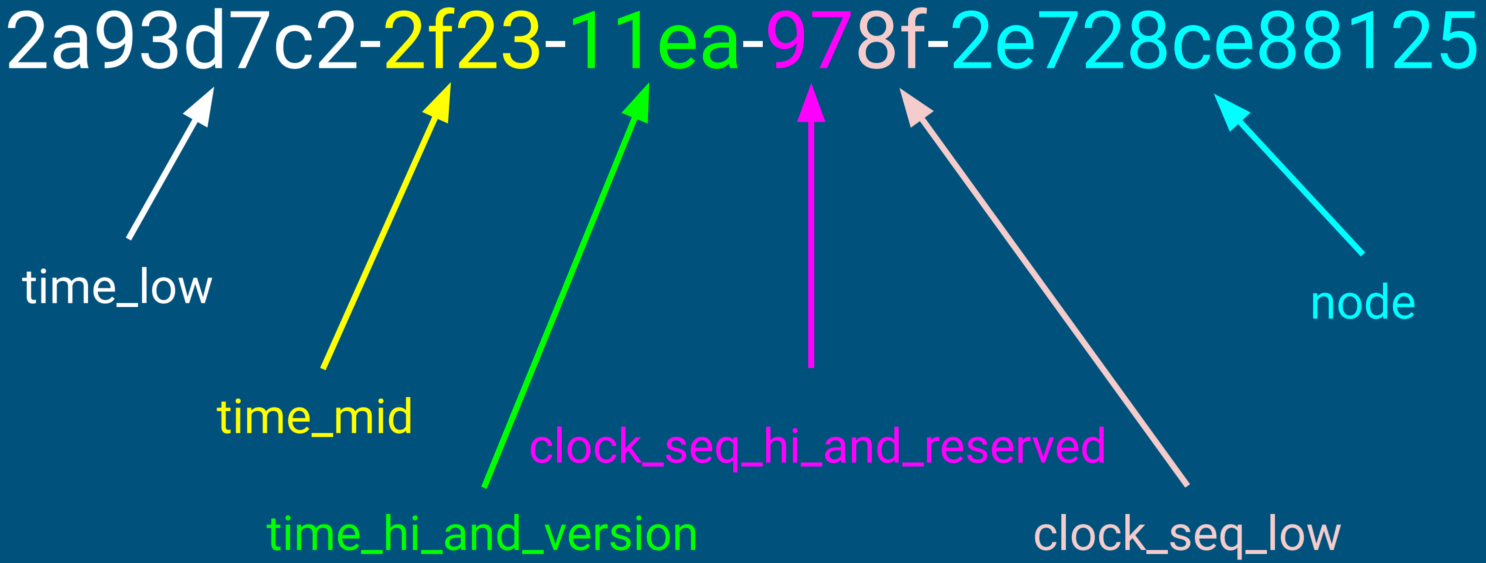
- 128 bits long (16 octets)
- Big-endian byte order

https://tools.ietf.org/html/rfc4122

Layout

Field	Data Type	Octet	Note
time_low	unsigned 32 bit integer	0-3	The low field of the timestamp
time_mid	unsigned 16 bit integer	4-5	The middle field of the timestamp
time_hi_and_version	unsigned 16 bit integer	6-7	The high field of the timestamp multiplexed with the version number
clock_seq_hi_and_reserved	unsigned 8 bit integer	8	The high field of the clock sequence multiplexed with the variant
clock_seq_low	unsigned 8 bit integer	9	The low field of the clock sequence
node	unsigned 48 bit integer	10-15	The spatially unique node identifier

The formal definition of the UUID string


2a93d7c2-2f23-11ea-978f-2e728ce88125
time_low
time_mid
time_hi_and_version
clock_seq_hi_and_reserved
clock_seq_low
node

Uuid

We can make it right

Guid: internals

```
namespace System
{
    [StructLayout(LayoutKind.Sequential)]
    public partial struct Guid : IFormattable, IComparable, IComparable<Guid>, IEquatable<Guid>, ISpanFormattable
    {
        private int _a;
        private short _b;
        private short _c;
        private byte _d;
        private byte _e;
        private byte _f;
        private byte _g;
        private byte _h;
        private byte _i;
        private byte _j;
        private byte _k;
    }
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L23-L33

Guid: public API

```
namespace System
{
    [StructLayout(LayoutKind.Sequential)]
    public partial struct Guid : IFormattable, IComparable, IComparable<Guid>, IEquatable<Guid>, ~~ISpanFormattable~~
    {
        public Guid(byte[] b)
        public Guid(ReadOnlySpan<byte> b)
        ~~public Guid(uint a, ushort b, ushort c, byte d, byte e, byte f, byte g, byte h, byte i, byte j, byte k)~~
        ~~public Guid(int a, short b, short c, byte[] d)~~
        public Guid(string g)
        public static Guid Parse(string input)
        public static Guid Parse(ReadOnlySpan<char> input)
        public static bool TryParse(string? input, out Guid result)
        public static bool TryParse(ReadOnlySpan<char> input, out Guid result)
        public static Guid ParseExact(string input, string format)
        public static Guid ParseExact(ReadOnlySpan<char> input, ReadOnlySpan<char> format)
        public static bool TryParseExact(string? input, string? format, out Guid result)
        public static bool TryParseExact(ReadOnlySpan<char> input, ReadOnlySpan<char> format, out Guid result)
        public byte[] ToByteArray()
        public bool TryWriteBytes(Span<byte> destination)
        ~~public bool TryFormat(Span<char> destination, out int charsWritten, ReadOnlySpan<char> format = default)~~
    }
}
```

Guid: System.ValueType override

```
namespace System
{
    [StructLayout(LayoutKind.Sequential)]
    public partial struct Guid : IFormattable, IComparable, IComparable<Guid>, IEquatable<Guid>, ~~ISpanFormattable~~
    {
        public override string ToString()
        public override int GetHashCode()
        public override bool Equals(object? o)
        public static bool operator ==(Guid a, Guid b)
        public static bool operator !=(Guid a, Guid b)
    }
}
```

Guid: interface methods

```
namespace System
{
    [StructLayout(LayoutKind.Sequential)]
    public partial struct Guid : IFormattable, IComparable, IComparable<Guid>, IEquatable<Guid>, ~~ISpanFormattable~~
    {
        public bool Equals(Guid g)
        public int CompareTo(object? value)
        public int CompareTo(Guid value)
        public string ToString(string? format, IFormatProvider? provider)
        ~~bool TryFormat(Span<char> destination, out int charsWritten, ReadOnlySpan<char> format, IFormatProvider? provider)~~
    }
}
```

Byte order: Uuid vs Guid

Source: 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

$\longrightarrow$

Guid: 33 22 11 00 55 44 77 66 88 99 AA BB CC DD EE FF

$\longleftarrow \; \longleftarrow \; \longleftarrow \; \longrightarrow$

Uuid: 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

$\longrightarrow$

Uuid: Layout

```
namespace Uuids
{
    [StructLayout(LayoutKind.Explicit, Pack = 1)]
    public struct Uuid : IFormattable, IComparable, IComparable<Uuid>, IEquatable<Uuid>
    {
        [FieldOffset(0)] private byte _byte0;
        [FieldOffset(1)] private byte _byte1;
        [FieldOffset(2)] private byte _byte2;
        [FieldOffset(3)] private byte _byte3;
        [FieldOffset(4)] private byte _byte4;
        [FieldOffset(5)] private byte _byte5;
        [FieldOffset(6)] private byte _byte6;
        [FieldOffset(7)] private byte _byte7;
        [FieldOffset(8)] private byte _byte8;
        [FieldOffset(9)] private byte _byte9;
        [FieldOffset(10)] private byte _byte10;
        [FieldOffset(11)] private byte _byte11;
        [FieldOffset(12)] private byte _byte12;
        [FieldOffset(13)] private byte _byte13;
        [FieldOffset(14)] private byte _byte14;
        [FieldOffset(15)] private byte _byte15;
    }
}
```

Guid: byte[] ctor

```
public Guid(byte[] b) : this(
    new ReadOnlySpan<byte>(b ?? throw new ArgumentNullException(nameof(b)))) {  }

public Guid(ReadOnlySpan<byte> b)
{
    if ((uint)b.Length != 16)
        throw new ArgumentException(SR.Format(SR.Arg_GuidArrayCtor, "16"), nameof(b));
    if (BitConverter.IsLittleEndian)
    {
        this = MemoryMarshal.Read<Guid>(b);
        return;
    }
    // … slower path for BigEndian
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L35-L53

MemoryMarshal.Read

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static T Read<T>(ReadOnlySpan<byte> source) where T : struct
{
    if (RuntimeHelpers.IsReferenceOrContainsReferences<T>())
    {
        ThrowHelper.ThrowInvalidTypeWithPointersNotSupported(typeof(T));
    }
    if (Unsafe.SizeOf<T>() > source.Length)
    {
        ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgument.length);
    }
    return Unsafe.ReadUnaligned<T>(ref GetReference(source));
}

public static ref T GetReference<T>(ReadOnlySpan<T> span) => ref span._pointer.Value;
```

https://github.com/dotnet/corefx/blob/v3.1.1/src/Common/src/CoreLib/System/Runtime/InteropServices/MemoryMarshal.cs#L165-L181

Unsafe.ReadUnaligned

```
namespace Internal.Runtime.CompilerServices
{
    public static unsafe class Unsafe
    {
        [Intrinsic]
        [NonVersionable]
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T ReadUnaligned<T>(ref byte source)
        {
            #if CORECLR
                typeof(T).ToString(); // Type token used by the actual method body
                throw new PlatformNotSupportedException();
            #else
                return Unsafe.As<byte, T>(ref *(byte*) source);
            #endif
}
```

https://github.com/dotnet/corefx/blob/v3.1.1/src/Common/src/CoreLib/Internal/Runtime/CompilerServices/Unsafe.cs#L242-L256

```
else if (tk == MscorlibBinder::GetMethod(METHOD__UNSAFE__BYREF_READ_UNALIGNED)->GetMemberDef() ||
         tk == MscorlibBinder::GetMethod(METHOD__UNSAFE__PTR_READ_UNALIGNED)->GetMemberDef())
{
    Instantiation inst = ftn->GetMethodInstantiation();
    mdToken tokGenericArg = FindGenericMethodArgTypeSpec(MscorlibBinder::GetModule()->GetMDImport());
    static const BYTE ilcode[]
    {
        CEE_LDARG_0,
        CEE_PREFIX1, (CEE_UNALIGNED & 0xFF), 1,
        CEE_LDOBJ, (BYTE)(tokGenericArg), (BYTE)(tokGenericArg >> 8), (BYTE)(tokGenericArg >> 16), (BYTE)(tokGenericArg >> 24),
        CEE_RET
    };
    methInfo->ILCode = const_cast<BYTE*>(ilcode);
    methInfo->ILCodeSize = sizeof(ilcode);
    methInfo->maxStack = 2;
    methInfo->EHcount = 0;
    methInfo->options = (CorInfoOptions)0;
    return true;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/vm/jitinterface.cpp#L7301-L7322

Unsafe.ReadUnaligned

```
// coreassembly.h
#define CORE_ASSEMBLY "System.Runtime"

// System.Runtime.CompilerServices.Unsafe.il
#include "coreassembly.h"
.assembly System.Runtime.CompilerServices.Unsafe
{
    .class public abstract auto ansi sealed beforefieldinit System.Runtime.CompilerServices.Unsafe
        extends [CORE_ASSEMBLY]System.Object
    {
        .method public hidebysig static !!T ReadUnaligned<T>(uint8& source) cil managed aggressiveinlining
        {
            .custom instance void System.Runtime.Versioning.NonVersionableAttribute::.ctor() = ( 01 00 00 00 )
            .maxstack 1
            ldarg.0
            unaligned. 0x1
            ldobj !!T
            ret
        } // end of method Unsafe::ReadUnaligned
    }
}
```

https://github.com/dotnet/corefx/blob/v3.1.1/src/System.Runtime.CompilerServices.Unsafe/src/System.Runtime.CompilerServices.Unsafe.il

Uuid: byte[] ctor

```
public Uuid(byte[] bytes)
{
   if (bytes == null)
      throw new ArgumentNullException(nameof(bytes));
   if ((uint) bytes.Length != 16)
      throw new ArgumentException("Byte array for Uuid must be exactly 16 bytes long.", nameof(bytes));
   this = Unsafe.ReadUnaligned<Uuid>(ref MemoryMarshal.GetReference(new ReadOnlySpan<byte>(bytes)));
}
```

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_CtorByteArray	471.9055 ns	5.4415 ns	5.0900 ns	0.0486	-	-	232 B
guid_CtorByteArray	6.939 ns	0.0831 ns	0.0777 ns	-	-	-	-
uuid_CtorByteArray	1.759 ns	0.0213 ns	0.0200 ns	-	-	-	-

Uuid: ReadOnlySpan<byte> ctor

```
public Uuid(ReadOnlySpan<byte> bytes)
{
   if ((uint) bytes.Length != 16)
      throw new ArgumentException("Byte array for Uuid must be exactly 16 bytes long.", nameof(bytes));
   this = Unsafe.ReadUnaligned<Uuid>(ref MemoryMarshal.GetReference(bytes));
}
```

Method	**Mean**	**Error**	**StdDev**	**Gen 0**	**Gen 1**	**Gen 2**	**Allocated**
dodo_CtorReadOnlySpan	N/A	N/A	N/A	N/A	N/A	N/A	N/A
guid_CtorReadOnlySpan	3.113 ns	0.0467 ns	0.0414 ns	-	-	-	-
uuid_CtorReadOnlySpan	2.725 ns	0.0276 ns	0.0258 ns	-	-	-	-

Dodo.Tools: ToByteArray

```
public Byte[] ToByteArray()
{
    if (_uuid.Length % 2 != 0)
    {
        throw new ArgumentException("hexString must have an even length");
    }
    Byte[] bytes = new Byte[_uuid.Length / 2];
    for (Int32 i = 0; i < bytes.Length; i++)
    {
        String currentHex = _uuid.Substring(i * 2, 2);
        bytes[i] = Convert.ToByte(currentHex, 16);
    }
    return bytes;
}
```

Guid: ToByteArray

```
public byte[] ToByteArray()
{
    var g = new byte[16];
    if (BitConverter.IsLittleEndian)
    {
        MemoryMarshal.TryWrite<Guid>(g, ref this);
    }
    else
    {
        TryWriteBytes(g);
    }
    return g;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L753-L765

MemoryMarshal.TryWrite

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool TryWrite<T>(Span<byte> destination, ref T value) where T : struct
{
    if (RuntimeHelpers.IsReferenceOrContainsReferences<T>())
    {
        ThrowHelper.ThrowInvalidTypeWithPointersNotSupported(typeof(T));
    }
    if (Unsafe.SizeOf<T>() > (uint)destination.Length)
    {
        return false;
    }
    Unsafe.WriteUnaligned<T>(ref GetReference(destination), value);
    return true;
}

public static ref T GetReference<T>(Span<T> span) => ref span._pointer.Value;
```

https://github.com/dotnet/corefx/blob/v3.1.1/src/Common/src/CoreLib/System/Runtime/InteropServices/MemoryMarshal.cs#L222-L240

Uuid: ToByteArray

```
public byte[] ToByteArray()
{
    var result = new byte[16];
    Unsafe.WriteUnaligned(ref MemoryMarshal.GetReference(new Span<byte>(result)), this);
    return result;
}
```

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_ToByteArray	587.5497 ns	9.2041 ns	8.6095 ns	0.1173	-	-	552 B
guid_ToByteArray	6.993 ns	0.2260 ns	0.2320 ns	0.0085	-	-	40 B
uuid_ToByteArray	6.751 ns	0.1750 ns	0.1637 ns	0.0085	-	-	40 B

Guid: TryWriteBytes

```
public bool TryWriteBytes(Span<byte> destination)
{
    if (BitConverter.IsLittleEndian)
        return MemoryMarshal.TryWrite(destination, ref this);
    if (destination.Length < 16)
        return false;
    destination[15] = _k; // hoist bounds checks
    destination[0] = (byte)(_a);
    destination[1] = (byte)(_a >> 8);
    destination[2] = (byte)(_a >> 16);
    destination[3] = (byte)(_a >> 24);
    destination[4] = (byte)(_b);
    destination[5] = (byte)(_b >> 8);
    destination[6] = (byte)(_c);
    destination[7] = (byte)(_c >> 8);
    destination[8] = _d;
    destination[9] = _e;
    destination[10] = _f;
    destination[11] = _g;
    destination[12] = _h;
    destination[13] = _i;
    destination[14] = _j;
    return true;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L768-L796

Uuid: TryWriteBytes

```
public bool TryWriteBytes(Span<byte> destination)
{
    if (Unsafe.SizeOf<Uuid>() > (uint) destination.Length)
        return false;
    Unsafe.WriteUnaligned(ref MemoryMarshal.GetReference(destination), this);
    return true;
}
```

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_TryWriteBytes	N/A	N/A	N/A	N/A	N/A	N/A	N/A
guid_TryWriteBytes	6.993 ns	0.2260 ns	0.2320 ns	0.0085	-	-	40 B
uuid_TryWriteBytes	6.751 ns	0.1750 ns	0.1637 ns	0.0085	-	-	40 B

Dodo.Tools: GetHashCode

```
namespace Dodo.Tools.Types
{
    public class UUId
    {
        private readonly String _uuid;
        // ...
        public override Int32 GetHashCode() => _uuid.GetHashCode();
        // ...
    }
}
```

Guid: GetHashCode

```
public override int GetHashCode()
{
    return _a ^ Unsafe.Add(ref _a, 1) ^ Unsafe.Add(ref _a, 2) ^ Unsafe.Add(ref _a, 3);
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L801-L805

Uuid: GetHashCode

```
namespace Uuids
{
    [StructLayout(LayoutKind.Explicit, Pack = 1)]
    public struct Uuid : IFormattable, IComparable, IComparable<Uuid>, IEquatable<Uuid>
    {
        [FieldOffset(0)] private byte _byte0;
        [FieldOffset(1)] private byte _byte1;
        [FieldOffset(2)] private byte _byte2;
        // ...
        [FieldOffset(15)] private byte _byte15;

        [FieldOffset(0)] private int _int0;
        [FieldOffset(4)] private int _int4;
        [FieldOffset(8)] private int _int8;
        [FieldOffset(12)] private int _int12;

        public override int GetHashCode()
        {
            return _int0 ^ _int4 ^ _int8 ^ _int12;
        }
    }
}
```

GetHashCode

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_GetHashCode	20.9091 ns	0.1695 ns	0.1585 ns	-	-	-	-
guid_GetHashCode	1.372 ns	0.0152 ns	0.0135 ns	-	-	-	-
uuid_GetHashCode	1.127 ns	0.0176 ns	0.0164 ns	-	-	-	-

Dodo.Tools: Equals

```
namespace Dodo.Tools.Types
{
    public class UUId
    {
        private readonly String _uuid;
        // ...
        public override Boolean Equals(Object obj)
        {
            if (ReferenceEquals(null, obj))
                return false;
            if (ReferenceEquals(this, obj))
                return true;
            if (obj.GetType() != GetType())
                return false;
            return Equals((UUId) obj);
        }

        protected Boolean Equals(UUId other)
        {
            return String.Equals(_uuid, other._uuid);
        }
    }
}
```

Guid: Equals

```
public override bool Equals(object? o)
{
    Guid g;
    if (o == null || !(o is Guid))
        return false;
    else g = (Guid)o;

    return g._a == _a &&
        Unsafe.Add(ref g._a, 1) == Unsafe.Add(ref _a, 1) &&
        Unsafe.Add(ref g._a, 2) == Unsafe.Add(ref _a, 2) &&
        Unsafe.Add(ref g._a, 3) == Unsafe.Add(ref _a, 3);
}

public bool Equals(Guid g)
{
    return g._a == _a &&
        Unsafe.Add(ref g._a, 1) == Unsafe.Add(ref _a, 1) &&
        Unsafe.Add(ref g._a, 2) == Unsafe.Add(ref _a, 2) &&
        Unsafe.Add(ref g._a, 3) == Unsafe.Add(ref _a, 3);
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L807-L831

Uuid: Equals

```
[StructLayout(LayoutKind.Explicit, Pack = 1)]
public struct Uuid : IFormattable, IComparable, IComparable<Uuid>, IEquatable<Uuid>
{
    [FieldOffset(0)] private byte _byte0;
    // ...
    [FieldOffset(15)] private byte _byte15;

    [FieldOffset(0)] private ulong _ulong0;
    [FieldOffset(8)] private ulong _ulong8;

    public override bool Equals(object? obj)
    {
        Uuid other;
        if (obj == null || !(obj is Uuid))
            return false;
        else other = (Uuid) obj;
        return _ulong0 == other._ulong0 && _ulong8 == other._ulong8;
    }

    public bool Equals(Uuid other)
    {
        return _ulong0 == other._ulong0 && _ulong8 == other._ulong8;
    }
}
```

Equals

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_EqualsSame_Object	9.687 ns	0.0744 ns	0.0660 ns	-	-	-	-
guid_EqualsSame_Object	3.430 ns	0.0423 ns	0.0375 ns	-	-	-	-
uuid_EqualsSame_Object	2.939 ns	0.0387 ns	0.0362 ns	-	-	-	-
dodo_EqualsSame_T	4.442 ns	0.0369 ns	0.0345 ns	-	-	-	-
guid_EqualsSame_T	2.029 ns	0.0184 ns	0.0164 ns	-	-	-	-
uuid_EqualsSame_T	1.505 ns	0.0252 ns	0.0236 ns	-	-	-	-

Guid: CompareTo

```
private int GetResult(uint me, uint them) => me < them ? -1 : 1;

public int CompareTo(object? value)
{
    if (value == null)
        return 1;
    if (!(value is Guid))
        throw new ArgumentException(SR.Arg_MustBeGuid, nameof(value));
    Guid g = (Guid)value;
    if (g._a != _a)
        return GetResult((uint)_a, (uint)g._a);
    if (g._b != _b)
        return GetResult((uint)_b, (uint)g._b);
    // … code here
    if (g._k != _k)
        return GetResult(_k, g._k);
   return 0;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L833-L903

Uuid: CompareTo

```
public int CompareTo(object? value)
{
    if (value == null) return 1;
    if (!(value is Uuid)) throw new ArgumentException("Object must be of type Uuid.", nameof(value));
    var other = (Uuid) value;
    if (other._byte0 != _byte0)
        return _byte0 < other._byte0 ? -1 : 1;
    if (other._byte1 != _byte1)
        return _byte1 < other._byte1 ? -1 : 1;
    // … code here
    if (other._byte13 != _byte13)
        return _byte13 < other._byte13 ? -1 : 1;
    if (other._byte14 != _byte14)
        return _byte14 < other._byte14 ? -1 : 1;
    if (other._byte15 != _byte15)
        return _byte15 < other._byte15 ? -1 : 1;
    return 0;
}
```

CompareTo

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_CompareTo_Object	N/A	N/A	N/A	N/A	N/A	N/A	N/A
guid_CompareTo_Object	6.806 ns	0.0771 ns	0.0721 ns	-	-	-	-
uuid_CompareTo_Object	7.412 ns	0.0827 ns	0.0773 ns	-	-	-	-
dodo_CompareTo_T	N/A	N/A	N/A	N/A	N/A	N/A	N/A
guid_CompareTo_T	4.952 ns	0.0410 ns	0.0384 ns	-	-	-	-
uuid_CompareTo_T	6.352 ns	0.0444 ns	0.0416 ns	-	-	-	-

ToString

Tips and tricks

Guid: ToString formats

- D – 00112233-4455-6677-8899-AABBCCDDEEFF
- N – 00112233445566778899AABBCCDDEEFF
- B – {00112233-4455-6677-8899-AABBCCDDEEFF}
- P – (00112233-4455-6677-8899-AABBCCDDEEFF)
- X – {0x00112233,0x4455,0x6677,{0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF}}

Guid: ToString

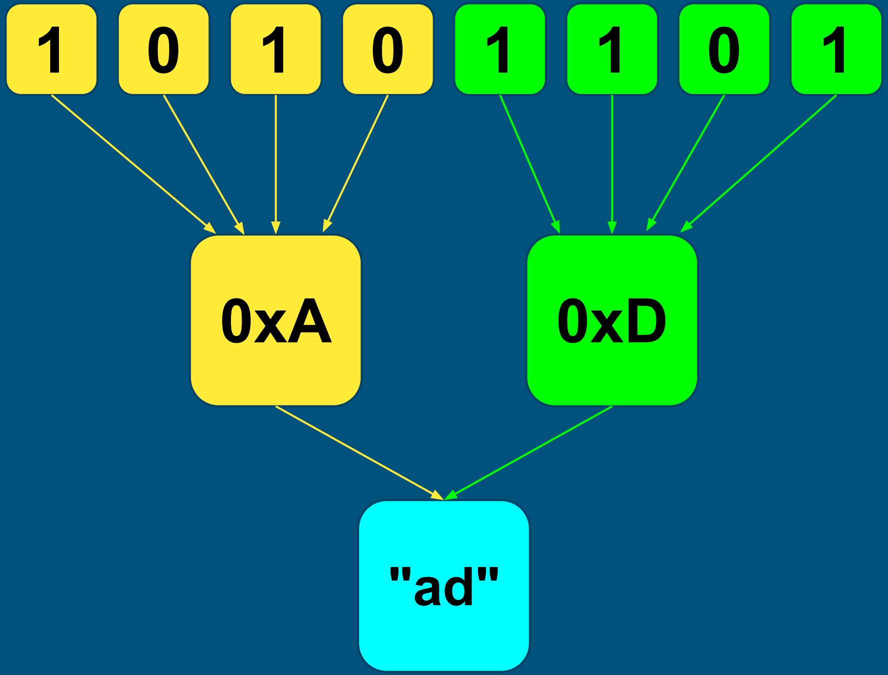
```
public override string ToString() => ToString("D", null);

public string ToString(string? format, IFormatProvider? provider)
{
    if (string.IsNullOrEmpty(format))
        format = "D";
    if (format.Length != 1)
        throw new FormatException(SR.Format_InvalidGuidFormatSpecification);
    int guidSize;
    switch (format[0])
    {
        case 'D':
        case 'd':
            guidSize = 36;
            break;
        case 'N':
        case 'n':
            guidSize = 32;
            break;
        case 'B':
        case 'b':
        case 'P':
        case 'p':
            guidSize = 38;
            break;
        case 'X':
        case 'x':
            guidSize = 68;
            break;
        default:
            throw new FormatException(SR.Format_InvalidGuidFormatSpecification);
    }
    string guidString = string.FastAllocateString(guidSize);

    int bytesWritten;
    bool result = TryFormat(new Span<char>(
        ref guidString.GetRawStringData(),
        guidString.Length),
        out bytesWritten,
        format);
    Debug.Assert(result && bytesWritten == guidString.Length, "Formatting guid
should have succeeded.");

    return guidString;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L1024-L1071

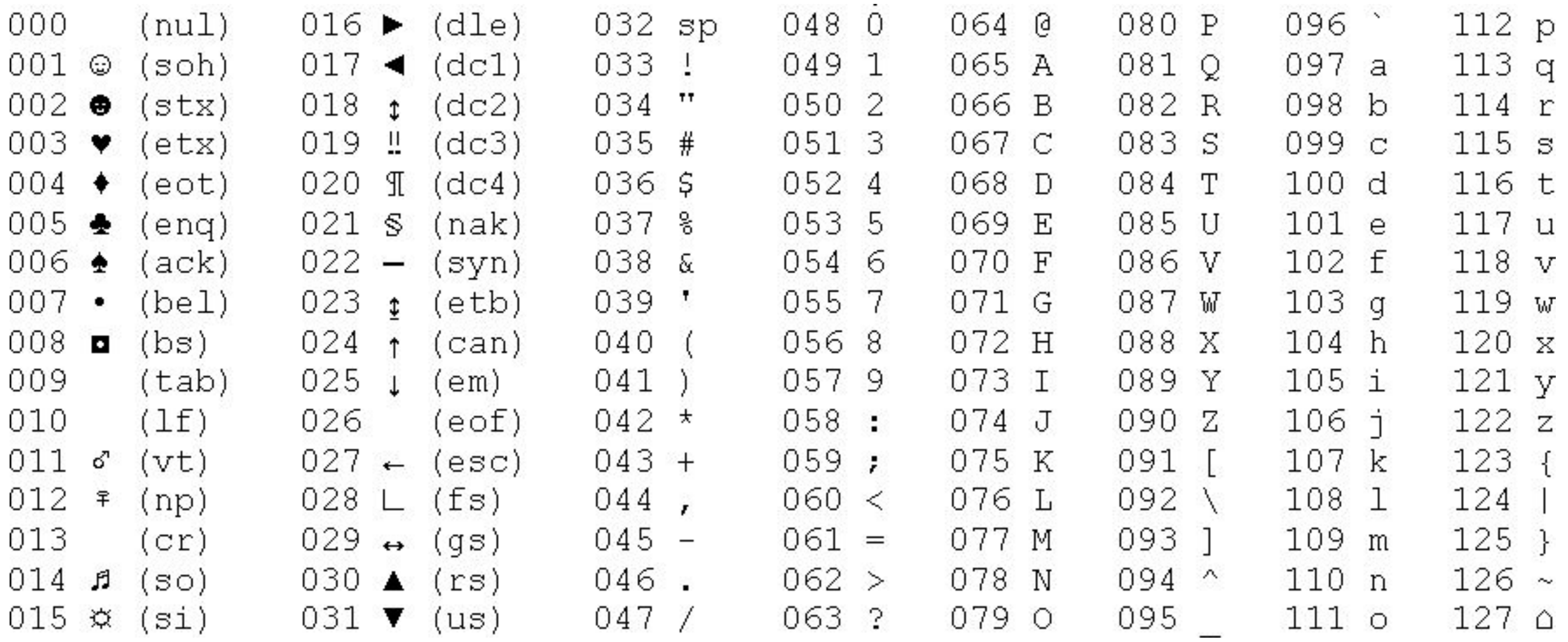
```
public bool TryFormat(
    Span<char> destination,
    out int charsWritten,
    ReadOnlySpan<char> format = default)
{
    if (format.Length == 0) format = "D";
    if (format.Length != 1) throw new FormatException(
        SR.Format_InvalidGuidFormatSpecification);
    bool dash = true;
    bool hex = false;
    int braces = 0;
    int guidSize;
    switch (format[0])
    {
        case 'D':
        case 'd':
            guidSize = 36;
            break;
        case 'N':
        case 'n':
            dash = false;
            guidSize = 32;
            break;
        case 'B':
        case 'b':
            braces = '{' + ('}' << 16);
            guidSize = 38;
            break;
        case 'P':
        case 'p':
            braces = '(' + (')' << 16);
            guidSize = 38;
            break;
        case 'X':
        case 'x':
            braces = '{' + ('}' << 16);
            dash = false;
            hex = true;
            guidSize = 68;
            break;
        default:
            throw new FormatException(
                SR.Format_InvalidGuidFormatSpecification);
    }

    if (destination.Length < guidSize)
    {
        charsWritten = 0;
        return false;
    }
    // ... more ...
// ...
unsafe
{
    fixed (char* guidChars = &MemoryMarshal.GetReference(destination))
    {
        char * p = guidChars;
        if (braces != 0)
            *p++ = (char)braces;
        if (hex)
        {
            // X
            *p++ = '0';
            *p++ = 'x';
            p += HexsToChars(p, _a >> 24, _a >> 16);
            p += HexsToChars(p, _a >> 8, _a);
            *p++ = ',';
            *p++ = '0';
            *p++ = 'x';
            p += HexsToChars(p, _b >> 8, _b);
            *p++ = ',';
            *p++ = '0';
            *p++ = 'x';
            p += HexsToChars(p, _c >> 8, _c);
            *p++ = ',';
            *p++ = '{';
            p += HexsToCharsHexOutput(p, _d, _e);
            *p++ = ',';
            p += HexsToCharsHexOutput(p, _f, _g);
            *p++ = ',';
            p += HexsToCharsHexOutput(p, _h, _i);
            *p++ = ',';
            p += HexsToCharsHexOutput(p, _j, _k);
            *p++ = '}';
        }
        else
        {
            // N, D, B, P
            p += HexsToChars(p, _a >> 24, _a >> 16);
            p += HexsToChars(p, _a >> 8, _a);
            if (dash)
                *p++ = '-';
            p += HexsToChars(p, _b >> 8, _b);
            if (dash)
                *p++ = '-';
            p += HexsToChars(p, _c >> 8, _c);
            if (dash)
                *p++ = '-';
            p += HexsToChars(p, _d, _e);
            if (dash)
                *p++ = '-';
            p += HexsToChars(p, _f, _g);
            p += HexsToChars(p, _h, _i);
            p += HexsToChars(p, _j, _k);
        }
        if (braces != 0)
            *p++ = (char)(braces >> 16);
        Debug.Assert(p - guidChars == guidSize);
    }
}
charsWritten = guidSize;
return true;
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/Guid.cs#L1074-L1191

byte to hex string



ASCII



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Uuid: ToString("N")

```
public override string ToString() { return ToString("D", null); }

public string ToString(string? format, IFormatProvider? provider)
{
    if (string.IsNullOrEmpty(format)) format = "D";
    if (format.Length != 1)
        throw new FormatException(
            "Format string can be only \"D\", \"d\", \"N\", \"n\", \"P\", \"p\", \"B\",
\"b\", \"X\" or \"x\".");
    switch (format[0])
    {
        case 'N':
        case 'n':
            return FormatN();
        default:
            throw new FormatException(
                "Format string can be only \"D\", \"d\", \"N\", \"n\", \"P\", \"p\",
\"B\", \"b\", \"X\" or \"x\".");
    }
}
```

```
private string FormatN() { return string.Create(32, this, FormatNAction); }

private static readonly SpanAction<char, Uuid> FormatNAction = FormatNStatic;

private static void FormatNStatic(Span<char> result, Uuid uuid)
{
    result[0] = HexToChar((byte) (uuid._byte0 >> 4));
    result[1] = HexToChar(uuid._byte0);
    result[2] = HexToChar((byte) (uuid._byte1 >> 4));
    result[3] = HexToChar(uuid._byte1);
    // ...
    result[28] = HexToChar((byte) (uuid._byte14 >> 4));
    result[29] = HexToChar(uuid._byte14);
    result[30] = HexToChar((byte) (uuid._byte15 >> 4));
    result[31] = HexToChar(uuid._byte15);
}

private static char HexToChar(byte a)
{
    a &= 0x0F;
    return (char) (a > 9 ? a + 97 - 10 : a + 48);
}
```

ToString("N")

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N	**43.6670 ns**	**0.5117 ns**	**0.4787 ns**	**0.0187**	**-**	**-**	**88 B**

System.String

Deep dive

System.String: Ctor(byte[])

```
private string Ctor(char[]? value)
{
    if (value == null || value.Length == 0)
        return Empty;
    string result = FastAllocateString(value.Length);
    unsafe
    {
        fixed (char* dest = &result._firstChar, source = value)
            wstrcpy(dest, source, value.Length);
    }
   return result;
}

internal static unsafe void wstrcpy(char* dmem, char* smem, int charCount)
{
    Buffer.Memmove((byte*)dmem, (byte*)smem, ((uint)charCount) * 2);
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/shared/System/String.cs#L53-L65

System.String: FastAllocateString

```
namespace System
{
    public partial class String
    {
        // ...
        [MethodImplAttribute(MethodImplOptions.InternalCall)]
        internal static extern string FastAllocateString(int length);
        // ...
    }
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/System.Private.CoreLib/src/System/String.CoreCLR.cs#L61-L62

System.String: FastAllocateString

```
#ifdef FEATURE_PAL
    ECall::DynamicallyAssignFCallImpl(GetEEFuncEntryPoint(AllocateString_MP_FastPortable), ECall::FastAllocateString);
#else
    // if (multi-proc || server GC)
    if (GCHeapUtilities::UseThreadAllocationContexts())
    {
        ECall::DynamicallyAssignFCallImpl(GetEEFuncEntryPoint(AllocateStringFastMP_InlineGetThread), ECall::FastAllocateString);
    }
    else
    {
        ECall::DynamicallyAssignFCallImpl(GetEEFuncEntryPoint(AllocateStringFastUP), ECall::FastAllocateString);
    }
#endif // FEATURE_PAL
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/vm/jitinterfacegen.cpp#L76-L118

System.String: FastAllocateString

```
// CMakeLists.txt
include(configurecompiler.cmake)
// ..code
include(clrdefinitions.cmake)


// configurecompiler.cmake
if(CMAKE_SYSTEM_NAME STREQUAL Linux)
   set(CLR_CMAKE_PLATFORM_UNIX 1)


// clrdefinitions.cmake
if(CLR_CMAKE_PLATFORM_UNIX)
    add_definitions(-DFEATURE_PAL)
    add_definitions(-DFEATURE_PAL_SXS)
    add_definitions(-DFEATURE_PAL_ANSI)
endif(CLR_CMAKE_PLATFORM_UNIX)
```

https://github.com/dotnet/coreclr/blob/v3.1.1/CMakeLists.txt#L137
https://github.com/dotnet/coreclr/blob/v3.1.1/configurecompiler.cmake#L21-L22
https://github.com/dotnet/coreclr/blob/v3.1.1/clrdefinitions.cmake#L172-L176

System.String: FastAllocateString

```
HCIMPL1(StringObject*, AllocateString_MP_FastPortable, DWORD stringLength)
{
  FCALL_CONTRACT;
  do
  {
     if (stringLength >= (LARGE_OBJECT_SIZE - 256) / sizeof(WCHAR))
     {
        break;
     }
     Thread *thread = GetThread();
     SIZE_T totalSize = StringObject::GetSize(stringLength);
     SIZE_T alignedTotalSize = ALIGN_UP(totalSize, DATA_ALIGNMENT);
     totalSize = alignedTotalSize;
     gc_alloc_context *allocContext = thread->GetAllocContext();
     BYTE *allocPtr = allocContext->alloc_ptr;
     if (totalSize > static_cast<SIZE_T>(allocContext->alloc_limit - allocPtr))
     {
        break;
     }
     allocContext->alloc_ptr = allocPtr + totalSize;
     StringObject *stringObject = reinterpret_cast<StringObject *>(allocPtr);
     stringObject->SetMethodTable(g_pStringClass);
     stringObject->SetStringLength(stringLength);
     return stringObject;
  } while (false);
  ENDFORBIDGC();
  return HCCALL1(FramedAllocateString, stringLength);
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/vm/jithelpers.cpp#L2744-L2796

Uuid: ToString("N") v2

```
public unsafe struct Uuid : IFormattable, IComparable, IComparable<Uuid>, IEquatable<Uuid>
{
   static Uuid()
   {
      FastAllocateString = (Func<int, string>) typeof(string)
         .GetMethod(nameof(FastAllocateString),BindingFlags.Static | BindingFlags.NonPublic)
         .CreateDelegate(typeof(Func<int, string>));
   }
   private static readonly Func<int, string> FastAllocateString;
   public string ToString(string? format, IFormatProvider? provider)
   {
   // ...
      switch (format[0])
      {
         case 'N':
         case 'n':
         {
            var uuidString = FastAllocateString(32);
            fixed (char* uuidChars = uuidString)
            {
               FormatN(uuidChars);
            }
            return uuidString;
         }
         // ...
```

```
private void FormatN(char* result)
{
   *result++ = HexToChar((byte) (_byte0 >> 4));
   *result++ = HexToChar(_byte0);
   *result++ = HexToChar((byte) (_byte1 >> 4));
   *result++ = HexToChar(_byte1);
   *result++ = HexToChar((byte) (_byte2 >> 4));
   *result++ = HexToChar(_byte2);
   // ...
   *result++ = HexToChar((byte) (_byte14 >> 4));
   *result++ = HexToChar(_byte14);
   *result++ = HexToChar((byte) (_byte15 >> 4));
   *result = HexToChar(_byte15);
}


private static char HexToChar(byte a)
{
   a &= 0x0F;
   return (char) (a > 9 ? a + 97 - 10 : a + 48);
}
```

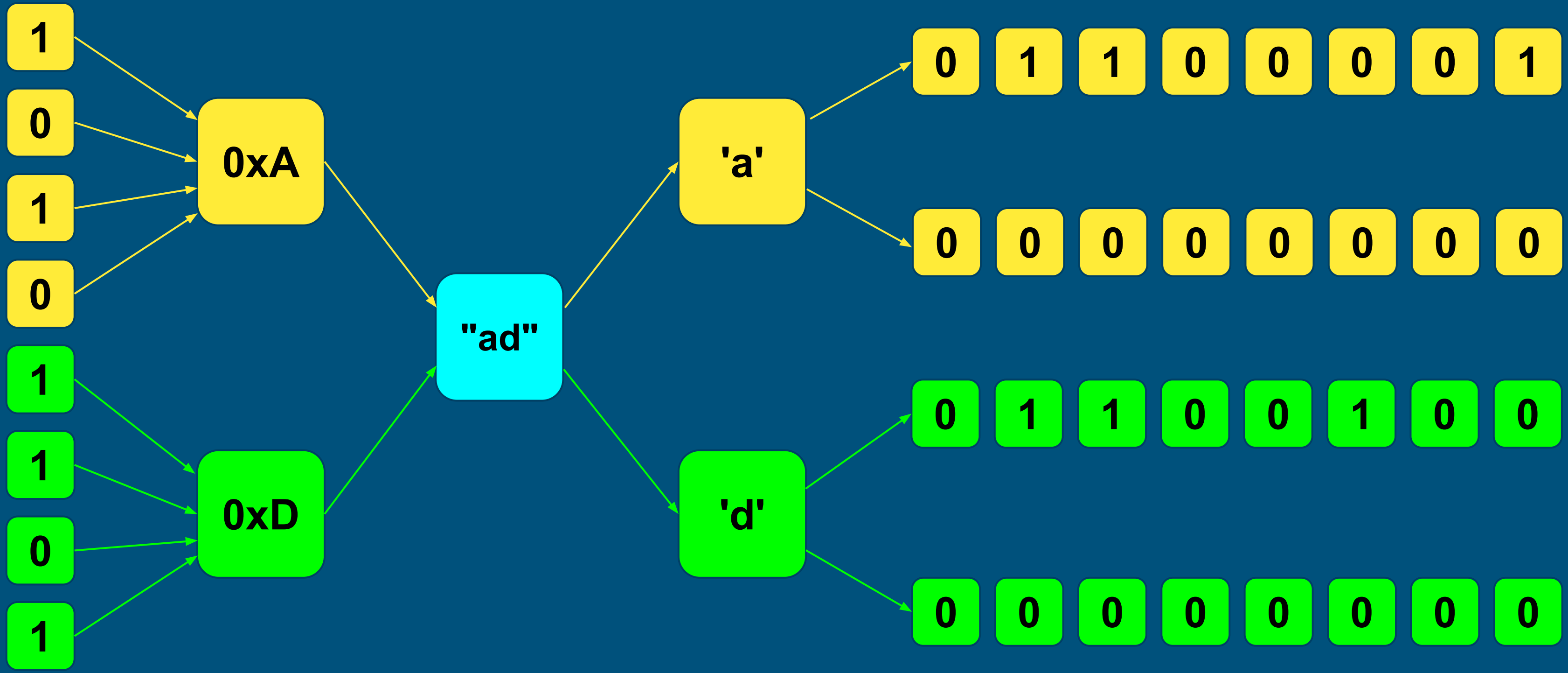
ToString("N") v2

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N_v1	43.6670 ns	0.5117 ns	0.4787 ns	0.0187	-	-	88 B
uuid_ToString_N_v2	**37.1856 ns**	**0.5368 ns**	**0.5021 ns**	**0.0187**	**-**	**-**	**88 B**

Too slow

ofc

byte to hex string



char + char == (u)int

Uuid: ToString("N") v3

```
public unsafe struct Uuid
{
    static Uuid()
    {
        FastAllocateString = (Func<int, string>) typeof(string)
            .GetMethod(nameof(FastAllocateString),
                BindingFlags.Static | BindingFlags.NonPublic)
            .CreateDelegate(typeof(Func<int, string>));

        TableToHex = new uint[256];
        for (var i = 0; i < 256; i++)
        {
            var chars = Convert.ToString(i, 16).PadLeft(2, '0');
            TableToHex[i] = ((uint) chars[1] << 16) | chars[0];
        }
    }

    private static readonly Func<int, string> FastAllocateString;
    private static readonly uint[] TableToHex;
```

```
private void FormatN(char* result)
{
    var destUints = (uint*) result;
    *destUints++ = TableToHex[_byte0];
    *destUints++ = TableToHex[_byte1];
    *destUints++ = TableToHex[_byte2];
    *destUints++ = TableToHex[_byte3];
    *destUints++ = TableToHex[_byte4];
    *destUints++ = TableToHex[_byte5];
    *destUints++ = TableToHex[_byte6];
    *destUints++ = TableToHex[_byte7];
    *destUints++ = TableToHex[_byte8];
    *destUints++ = TableToHex[_byte9];
    *destUints++ = TableToHex[_byte10];
    *destUints++ = TableToHex[_byte11];
    *destUints++ = TableToHex[_byte12];
    *destUints++ = TableToHex[_byte13];
    *destUints++ = TableToHex[_byte14];
    *destUints = TableToHex[_byte15];
}
```

ToString("N") v3

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N_v1	43.6670 ns	0.5117 ns	0.4787 ns	0.0187	-	-	88 B
uuid_ToString_N_v2	37.1856 ns	0.5368 ns	0.5021 ns	0.0187	-	-	88 B
uuid_ToString_N_v3	**29.4303 ns**	**0.4738 ns**	**0.4432 ns**	**0.0187**	**-**	**-**	**88 B**

uint[] ?

nope

uint[] vs *uint

```
public unsafe class ArrayBenchmarks
{
    public ArrayBenchmarks()
    {
        _byte0 = 0x11;
        _byte1 = 0x22;
    }
    static ArrayBenchmarks()
    {
        TableToHexArray = new uint[256];
        for (var i = 0; i < 256; i++)
        {
            var chars = Convert.ToString(i, 16).PadLeft(2, '0');
            TableToHexArray[i] = ((uint)chars[1] << 16) | chars[0];
        }
        TableToHexPtr = (uint*)Marshal.AllocHGlobal(sizeof(uint) * 256).ToPointer();
        for (var i = 0; i < 256; i++)
        {
            var chars = Convert.ToString(i, 16).PadLeft(2, '0');
            TableToHexPtr[i] = ((uint)chars[1] << 16) | chars[0];
        }
    }
    private static readonly uint[] TableToHexArray;
    private static readonly uint* TableToHexPtr;
    private byte _byte0;
    private byte _byte1;

    public void FillArray(uint* dest)
    {
        dest[0] = TableToHexArray[_byte0];
        dest[1] = TableToHexArray[_byte1];
    }

    public void FillPtr(uint* dest)
    {
        dest[0] = TableToHexPtr[_byte0];
        dest[1] = TableToHexPtr[_byte1];
    }

    [Benchmark]
    public void FillPtr()
    {
        var dest = stackalloc uint[2];
        FillPtr(dest);
    }

    [Benchmark]
    public void FillArray()
    {
        var dest = stackalloc uint[2];
        FillArray(dest);
    }
}
```

uint[] vs *uint

FillPtr(UInt32*)

```
movzx   eax,byte ptr [rcx+8]
movsxd  rax,eax
mov     r8,1D2EEA3DFC0h
mov     eax,dword ptr [r8+rax*4]
mov     dword ptr [rdx],eax
movzx   eax,byte ptr [rcx+9]
movsxd  rax,eax
mov     eax,dword ptr [r8+rax*4]
mov     dword ptr [rdx+4],eax
ret
```

FillArray(UInt32*)

```
mov     rax,1CCA54874C8h
mov     rax,qword ptr [rax]
movzx   r8d,byte ptr [rcx+8]
mov     r9d,dword ptr [rax+8]
cmp     r8d,r9d
jae     00007ffe`3789a222
movsxd  r8,r8d
mov     eax,dword ptr [rax+r8*4+10h]
mov     dword ptr [rdx],eax
mov     rax,1CCA54874C8h
mov     rax,qword ptr [rax]
movzx   ecx,byte ptr [rcx+9]
mov     r8d,dword ptr [rax+8]
cmp     ecx,r8d
jae     00007ffe`3789a222
movsxd  rcx,ecx
mov     eax,dword ptr [rax+rcx*4+10h]
mov     dword ptr [rdx+4],eax
add     rsp,28h
```

Uuid: ToString("N") v4

```
public unsafe struct Uuid : IFormattable, IComparable, IComparable<Uuid>, IEquatable<Uuid>
{
    static Uuid()
    {
        FastAllocateString = (Func<int, string>) typeof(string)
            .GetMethod(nameof(FastAllocateString),BindingFlags.Static | BindingFlags.NonPublic)
            .CreateDelegate(typeof(Func<int, string>));

        TableToHex = (uint*) Marshal.AllocHGlobal(sizeof(uint) * 256).ToPointer();
        for (var i = 0; i < 256; i++)
        {
            var chars = Convert.ToString(i, 16).PadLeft(2, '0');
            TableToHex[i] = ((uint) chars[1] << 16) | chars[0];
        }
    }

    private static readonly Func<int, string> FastAllocateString;
    private static readonly uint* TableToHex;
```

```
private void FormatN(char* dest)
{
    var destUints = (uint*) dest;
    destUints[0] = TableToHex[_byte0];
    destUints[1] = TableToHex[_byte1];
    destUints[2] = TableToHex[_byte2];
    destUints[3] = TableToHex[_byte3];
    destUints[4] = TableToHex[_byte4];
    destUints[5] = TableToHex[_byte5];
    destUints[6] = TableToHex[_byte6];
    destUints[7] = TableToHex[_byte7];
    destUints[8] = TableToHex[_byte8];
    destUints[9] = TableToHex[_byte9];
    destUints[10] = TableToHex[_byte10];
    destUints[11] = TableToHex[_byte11];
    destUints[12] = TableToHex[_byte12];
    destUints[13] = TableToHex[_byte13];
    destUints[14] = TableToHex[_byte14];
    destUints[15] = TableToHex[_byte15];
}
```

ToString("N") v4

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N_v1	43.6670 ns	0.5117 ns	0.4787 ns	0.0187	-	-	88 B
uuid_ToString_N_v2	37.1856 ns	0.5368 ns	0.5021 ns	0.0187	-	-	88 B
uuid_ToString_N_v3	29.4303 ns	0.4738 ns	0.4432 ns	0.0187	-	-	88 B
uuid_ToString_N_v4	**22.8331 ns**	**0.2602 ns**	**0.2434 ns**	**0.0187**	**-**	**-**	**88 B**

Hardware Intrinsics

Since .NET Core 2.1

Uuid: ToString("N") v5

```
private void FormatN(char* dest)
{
   if (Avx2.IsSupported)
   {
      FormatNAvx(dest);
   }
   else
   {
      FormatNTable(dest);
   }
}

private static Vector256<byte> ShuffleMask = Vector256.Create(
   255, 0, 255, 2, 255, 4, 255, 6, 255, 8, 255, 10, 255, 12, 255, 14,
   255, 0, 255, 2, 255, 4, 255, 6, 255, 8, 255, 10, 255, 12, 255, 14);


private static Vector256<byte> AsciiTable = Vector256.Create(
   (byte) 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 97, 98, 99, 100, 101, 102,
   48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 97, 98, 99, 100, 101, 102);
   // '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
```

```
private void FormatNAvx(char* dest)
{
   // dddddddddddddddddddddddddddddddd
   fixed (Uuid* thisPtr = &this)
   {
      var uuidVector = Avx2.ConvertToVector256Int16(Sse3.LoadDquVector128((byte*) thisPtr));
      var hi = Avx2.ShiftRightLogical(uuidVector, 4).AsByte();
      var lo = Avx2.Shuffle(uuidVector.AsByte(), ShuffleMask);
      var asciiBytes = Avx2.Shuffle(AsciiTable, Avx2.And(Avx2.Or(hi, lo), Vector256.Create((byte) 0x0F)));
      Avx.Store((short*) dest, Avx2.ConvertToVector256Int16(asciiBytes.GetLower()));
      Avx.Store((((short*) dest) + 16), Avx2.ConvertToVector256Int16(asciiBytes.GetUpper()));
   }
}
```

ToString("N") v5

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N_v1	43.6670 ns	0.5117 ns	0.4787 ns	0.0187	-	-	88 B
uuid_ToString_N_v2	37.1856 ns	0.5368 ns	0.5021 ns	0.0187	-	-	88 B
uuid_ToString_N_v3	29.4303 ns	0.4738 ns	0.4432 ns	0.0187	-	-	88 B
uuid_ToString_N_v4	22.8331 ns	0.2602 ns	0.2434 ns	0.0187	-	-	88 B
uuid_ToString_N_v5	**19.5746 ns**	**0.4559 ns**	**0.5067 ns**	**0.0187**	**-**	**-**	**88 B**

Reflection is useless

Gotta go fast

Assembly

```
// assembly.cpp
FriendAssemblyDescriptor *FriendAssemblyDescriptor::CreateFriendAssemblyDescriptor(PEAssembly *pAssembly)
{
    // ...
    for( int count = 0 ; count < 2 ; ++count)
    {
        // ...
        if (count == 0)
        {
            hr = pImport->EnumCustomAttributeByNameInit(TokenFromRid(1, mdtAssembly), FRIEND_ASSEMBLY_TYPE, &hEnum);
        }
        else
        {
            hr = pImport->EnumCustomAttributeByNameInit(TokenFromRid(1, mdtAssembly), SUBJECT_ASSEMBLY_TYPE, &hEnum);
        }

// corhdr.h
#define FRIEND_ASSEMBLY_TYPE                    "System.Runtime.CompilerServices.InternalsVisibleToAttribute"
// ...
#define SUBJECT_ASSEMBLY_TYPE                   "System.Runtime.CompilerServices.IgnoresAccessChecksToAttribute"
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/vm/assembly.cpp#L2412-L2513
https://github.com/dotnet/coreclr/blob/v3.1.1/src/inc/corhdr.h#L1803-L1808

IgnoresAccessChecksToAttribute

```
[assembly: System.Runtime.CompilerServices.IgnoresAccessChecksTo("System.Private.CoreLib")]

namespace System.Runtime.CompilerServices
{
    [AttributeUsage(AttributeTargets.Assembly, AllowMultiple = true)]
    public class IgnoresAccessChecksToAttribute : Attribute
    {
        public IgnoresAccessChecksToAttribute(string assemblyName)
        {
            AssemblyName = assemblyName;
        }

        public string AssemblyName { get; }
    }
}
```

https://github.com/dotnet/coreclr/blob/v3.1.1/src/vm/assembly.cpp#L2412-L2513
https://github.com/dotnet/coreclr/blob/v3.1.1/src/inc/corhdr.h#L1803-L1808

ToString("N") v6

```
// NuGet.config
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <packageSources>
      <add key="nuget.org" value="https://www.nuget.org/api/v3/index.json" />
      <add key="DotnetCore" value="https://dotnet.myget.org/F/dotnet-core/api/v3/index.json" />
   </packageSources>
</configuration>

// global.json
{
   "msbuild-sdks": { "Microsoft.NET.Sdk.IL": "3.0.0-preview-27318-01" },
   "sdk": { "version": "3.1.100" }
}

// Uuids.CoreLib.ilproj
<Project Sdk="Microsoft.NET.Sdk.IL">
   <PropertyGroup>
      <TargetFramework>netcoreapp3.1</TargetFramework>
      <MicrosoftNetCoreIlasmPackageVersion>3.0.0-preview-27318-01</MicrosoftNetCoreIlasmPackageVersion>
  </PropertyGroup>
</Project>
```

ToString("N") v6

```
// CoreLibInternal.il
.assembly extern System.Runtime
{
  .publickeytoken = ( B0 3F 5F 7F 11 D5 0A 3A )
  .ver 4:2:2:0
}
.assembly extern System.Runtime.Extensions
{
  .publickeytoken = ( B0 3F 5F 7F 11 D5 0A 3A )
  .ver 4:2:2:0
}
.assembly extern System.Private.CoreLib
{
  .publickeytoken = ( 7C EC 85 D7 BE A7 79 8E )
  .ver 4:0:0:0
}
.assembly Uuids.CoreLib
{
  .custom instance void System.Runtime.CompilerServices.IgnoresAccessChecksToAttribute::.ctor(string) = (
    01 00 16 53 79 73 74 65 6d 2e 50 72 69 76 61 74
    65 2e 43 6f 72 65 4c 69 62 00 00
  )
}
```

```
.module Uuids.CoreLib.dll

.class public abstract sealed auto ansi beforefieldinit
  Uuids.CoreLib.Internal
    extends [System.Runtime]System.Object
{
  .method public hidebysig static string
    FastAllocateString(
      int32 length
    ) cil managed aggressiveinlining
  {
    .maxstack 8
    ldarg.0      // length
    call         string System.String::FastAllocateString(int32)
    ret
  }
}
```

ToString("N") v6

```
// Uuid.cs
public string ToString(string? format, IFormatProvider? provider)
{
    // ...
    var uuidString = Uuids.CoreLib.Internal.FastAllocateString(32);
    // ...
```

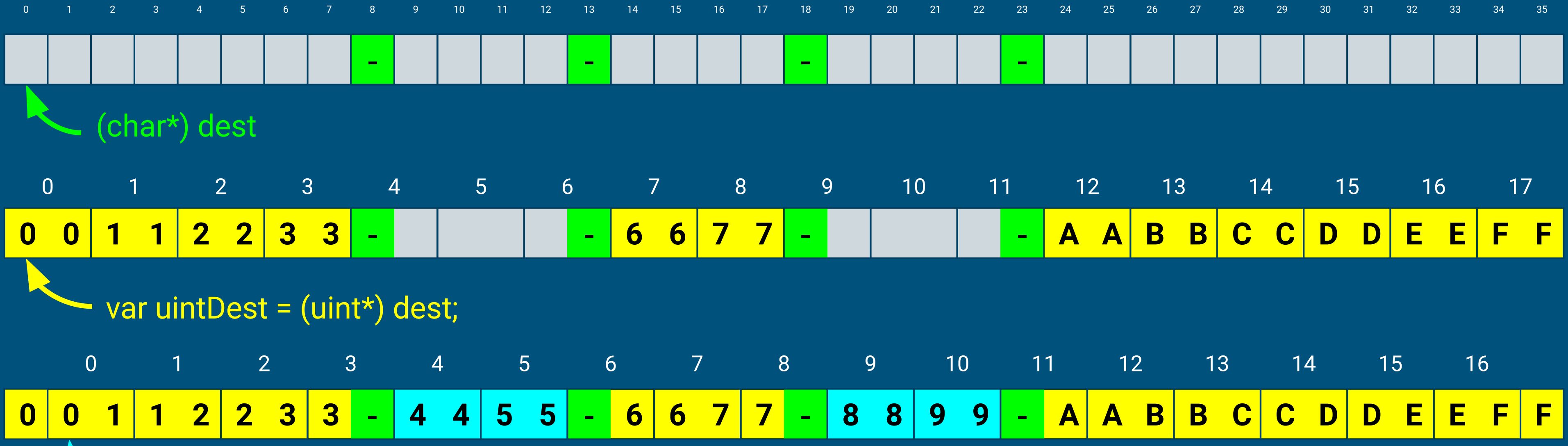
ToString("N") v6

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N_v1	43.6670 ns	0.5117 ns	0.4787 ns	0.0187	-	-	88 B
uuid_ToString_N_v2	37.1856 ns	0.5368 ns	0.5021 ns	0.0187	-	-	88 B
uuid_ToString_N_v3	29.4303 ns	0.4738 ns	0.4432 ns	0.0187	-	-	88 B
uuid_ToString_N_v4	22.8331 ns	0.2602 ns	0.2434 ns	0.0187	-	-	88 B
uuid_ToString_N_v5	19.5746 ns	0.4559 ns	0.5067 ns	0.0187	-	-	88 B
uuid_ToString_N_v6	**17.8747 ns**	**0.2416 ns**	**0.2260 ns**	**0.0187**	**-**	**-**	**88 B**

ToString("D")

```
[MethodImpl(MethodImplOptions.AggressiveInlining | MethodImplOptions.AggressiveOptimization)]
private void FormatD(char* dest)
{
   // dddddddd-dddd-dddd-dddd-dddddddddddd
   var uintDest = (uint*) dest;
   var uintDestAsChars = (char**) &uintDest;
   dest[8] = dest[13] = dest[18] = dest[23] = '-';
   uintDest[0] = TableToHex[_byte0];
   uintDest[1] = TableToHex[_byte1];
   uintDest[2] = TableToHex[_byte2];
   uintDest[3] = TableToHex[_byte3];
   uintDest[7] = TableToHex[_byte6];
   uintDest[8] = TableToHex[_byte7];
   uintDest[12] = TableToHex[_byte10];
   uintDest[13] = TableToHex[_byte11];
   uintDest[14] = TableToHex[_byte12];
   uintDest[15] = TableToHex[_byte13];
   uintDest[16] = TableToHex[_byte14];
   uintDest[17] = TableToHex[_byte15];
   *uintDestAsChars += 1;
   uintDest[4] = TableToHex[_byte4];
   uintDest[5] = TableToHex[_byte5];
   uintDest[9] = TableToHex[_byte8];
   uintDest[10] = TableToHex[_byte9];
}
```

D: 00112233-4455-6677-8899-AABBCCDDEEFF



```
var uintDest = (uint*) dest;
var uintDestAsChars = (char**) &uintDest;
*uintDestAsChars += 1;
```

ToString()

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_ToString_N	47.1792 ns	0.3581 ns	0.3350 ns	0.0187	-	-	88 B
uuid_ToString_N	17.8747 ns	0.2416 ns	0.2260 ns	0.0187	-	-	88 B
guid_ToString_D	47.10 ns	0.227 ns	0.212 ns	0.0204	-	-	96 B
uuid_ToString_D	27.15 ns	0.176 ns	0.156 ns	0.0204	-	-	96 B
guid_ToString_B	46.58 ns	0.397 ns	0.352 ns	0.0221	-	-	104 B
uuid_ToString_B	27.71 ns	0.209 ns	0.185 ns	0.0221	-	-	104 B
guid_ToString_P	47.02 ns	0.232 ns	0.205 ns	0.0221	-	-	104 B
uuid_ToString_P	27.64 ns	0.251 ns	0.235 ns	0.0221	-	-	104 B
guid_ToString_X	55.53 ns	0.980 ns	0.917 ns	0.0340	-	-	160 B
uuid_ToString_X	35.16 ns	0.598 ns	0.530 ns	0.0340	-	-	160 B

Parse

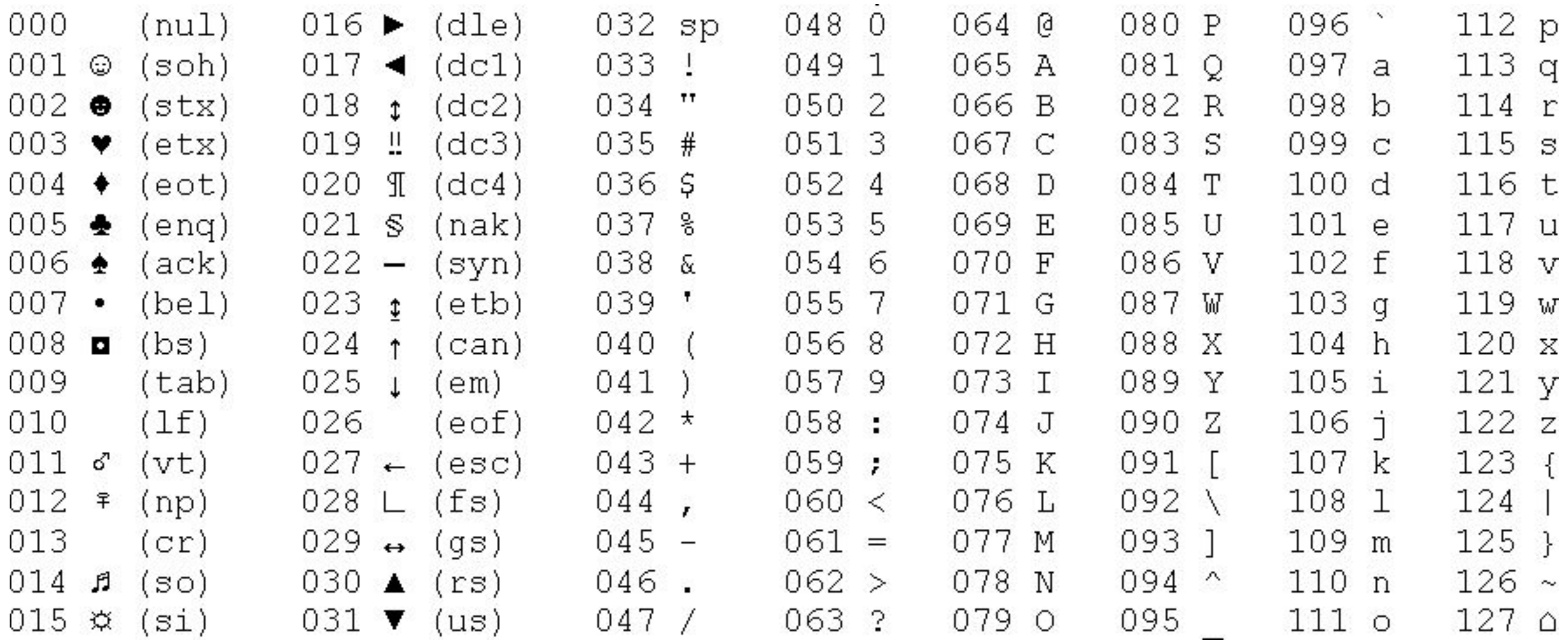
So much pain

Guid: Parse

```
public static Guid Parse(string input) =>
    Parse(input != null
        ? (ReadOnlySpan<char>)input
        : throw new ArgumentNullException(nameof(input)));

public static Guid Parse(ReadOnlySpan<char> input)
{
    var result = new GuidResult(GuidParseThrowStyle.AllButOverflow);
    bool success = TryParseGuid(input, ref result);
    return result._parsedGuid;
}
```

```
private static bool TryParseGuid(ReadOnlySpan<char> guidString, ref GuidResult result)
{
    guidString = guidString.Trim();
    if (guidString.Length == 0)
    {
        result.SetFailure(overflow: false, nameof(SR.Format_GuidUnrecognized));
        return false;
    }
   switch (guidString[0])
   {
       case '(':
           return TryParseExactP(guidString, ref result);

       case '{':
           return guidString.Contains('-') ?
               TryParseExactB(guidString, ref result) :
               TryParseExactX(guidString, ref result);

       default:
           return guidString.Contains('-') ?
               TryParseExactD(guidString, ref result) :
               TryParseExactN(guidString, ref result);
   }
}
```

ASCII



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Uuid: Parse

```
public unsafe struct Uuid
{
    static Uuid()
    {
        TableFromHexToBytes = (byte*) Marshal.AllocHGlobal(103).ToPointer();
        for (var i = 0; i < 103; i++)
            TableFromHexToBytes[i] = (char) i switch
            {
                '0' => (byte) 0x0,
                '1' => (byte) 0x1,
                '2' => (byte) 0x2,
                '3' => (byte) 0x3,
                '4' => (byte) 0x4,
                '5' => (byte) 0x5,
                //...
                'f' => (byte) 0xF,
                'F' => (byte) 0xF,
                _ => byte.MaxValue
            };
    }

    private const ushort MaximalChar = 103;

    private static readonly byte* TableFromHexToBytes;

    //...struct code here
}
```

Uuid: Parse

```
private static bool TryParsePtrN(char* value, byte* resultPtr)
{
    // e.g. "d85b1407351d4694939203acc5870eb1"
    byte hexByteHi;
    byte hexByteLow;
    // 0 byte
    if (value[0] < MaximalChar
        && (hexByteHi = TableFromHexToBytes[value[0]]) != 0xFF
        && value[1] < MaximalChar
        && (hexByteLow = TableFromHexToBytes[value[1]]) != 0xFF)
    {
        resultPtr[0] = (byte) ((byte) (hexByteHi << 4) | hexByteLow);
        // 1 byte
        if (value[2] < MaximalChar
            && (hexByteHi = TableFromHexToBytes[value[2]]) != 0xFF
            && value[3] < MaximalChar
            && (hexByteLow = TableFromHexToBytes[value[3]]) != 0xFF)
        {
            resultPtr[1] = (byte) ((byte) (hexByteHi << 4) | hexByteLow);
            // … and so on
```

TryParse

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
guid_TryParse_N	104.40 ns	1.796 ns	1.680 ns	-	-	-	-
uuid_TryParse_N	43.89 ns	0.867 ns	0.811 ns	-	-	-	-
guid_TryParse_D	80.62 ns	0.923 ns	0.863 ns	-	-	-	-
uuid_TryParse_D	35.87 ns	0.701 ns	1.171 ns	-	-	-	-
guid_TryParse_B	83.92 ns	1.257 ns	1.176 ns	-	-	-	-
uuid_TryParse_B	33.67 ns	0.651 ns	0.775 ns	-	-	-	-
guid_TryParse_P	79.17 ns	1.524 ns	1.565 ns	-	-	-	-
uuid_TryParse_P	25.81 ns	0.507 ns	0.497 ns	-	-	-	-
guid_TryParse_X	284.95 ns	2.798 ns	2.617 ns	-	-	-	-
uuid_TryParse_X	43.72 ns	0.472 ns	0.442 ns	-	-	-	-

Uuid.New()

Wrap up

Guid: NewGuid

```
public static unsafe Guid NewGuid()
{
    Guid g;
    Interop.GetRandomBytes((byte*)&g, sizeof(Guid));

    const ushort VersionMask = 0xF000;
    const ushort RandomGuidVersion = 0x4000;

    const byte ClockSeqHiAndReservedMask = 0xC0;
    const byte ClockSeqHiAndReservedValue = 0x80;

    unchecked
    {
        // time_hi_and_version
        g._c = (short)((g._c & ~VersionMask) | RandomGuidVersion);
        // clock_seq_hi_and_reserved
        g._d = (byte)((g._d & ~ClockSeqHiAndReservedMask) | ClockSeqHiAndReservedValue);
    }
    return g;
}
```

Dodo.Tools: GenerateTimeBasedGuid

```
public static Guid GenerateTimeBasedGuid()
{
    return GenerateTimeBasedGuid(DateTimeOffset.UtcNow, DefaultNode);
}

public static Guid GenerateTimeBasedGuid(DateTimeOffset dateTime, byte[] node)
{
    if (node == null) throw new ArgumentNullException("node");
    if (node.Length != 6) throw new ArgumentOutOfRangeException("node", "The node must be 6 bytes.");
    long ticks = (dateTime - GregorianCalendarStart).Ticks;
    byte[] guid = new byte[ByteArraySize];
    byte[] timestamp = BitConverter.GetBytes(ticks);
    // copy node
    Array.Copy(node, 0, guid, NodeByte, Math.Min(6, node.Length));
    // copy clock sequence
    byte[] clockSequenceBytes = BitConverter.GetBytes(Interlocked.Increment(ref DefaultClockSequence)).Reverse().ToArray();
    Array.Copy(clockSequenceBytes, 2, guid, GuidClockSequenceByte, 2);
    // copy timestamp
    Array.Copy(timestamp, 0, guid, TimestampByte, Math.Min(8, timestamp.Length));
    guid[VariantByte] &= (byte)VariantByteMask;
    guid[VariantByte] |= (byte)VariantByteShift;
    guid[VersionByte] &= (byte)VersionByteMask;
    guid[VersionByte] |= (byte)((byte)GuidVersion.TimeBased << VersionByteShift);
    return new Guid(guid);
}
```

Uuid: NewTimeBased

```
private const long GregorianCalendarReformDateTicks = 499_163_040_000_000_000L;

private const byte ResetVersionMask = 0b0000_1111;
private const byte Version1Flag = 0b0001_0000;

private const byte ResetReservedMask = 0b0011_1111;
private const byte ReservedFlag = 0b1000_0000;

public static Uuid NewTimeBased()
{
    var result = stackalloc byte[16];
    CoreLib.Internal.GetRandomBytes(result + 8, 8);
    var currentTicks = DateTime.UtcNow.Ticks - GregorianCalendarReformDateTicks;
    var ticksPtr = (byte*) &currentTicks;
    result[0] = ticksPtr[3];
    result[1] = ticksPtr[2];
    result[2] = ticksPtr[1];
    result[3] = ticksPtr[0];
    result[4] = ticksPtr[5];
    result[5] = ticksPtr[4];
    result[6] = (byte) ((ticksPtr[7] & ResetVersionMask) | Version1Flag);
    result[7] = ticksPtr[6];
    result[8] = (byte) ((result[8] & ResetReservedMask) | ReservedFlag);
    return new Uuid(result);
}
```

Generation

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
dodo_New	447.7 ns	1.99 ns	1.86 ns	0.1545	-	-	728 B
guid_New	271.4 ns	2.93 ns	2.74 ns	-	-	-	-
uuid_New	308.6 ns	1.26 ns	1.12 ns	-	-	-	-

Available now

https://nuget.org/packages/Uuids
https://github.com/vanbukin/Uuids