# UIntPtr Struct

Reference

## Definition

Namespace: System

Assembly: System.Runtime.dll

> Important
>
> This API is not CLS-compliant.

Represents an unsigned integer where the bit-width is the same as a pointer.

C#

```
[System.CLSCompliant(false)]
public readonly struct UIntPtr : IComparable<UIntPtr>,
IEquatable<UIntPtr>, IParsable<UIntPtr>, ISpanParsable<UIntPtr>,
System.Numerics.IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IAdditiveIdentity<UIntPtr,UIntPtr>,
System.Numerics.IBinaryInteger<UIntPtr>,
System.Numerics.IBinaryNumber<UIntPtr>,
System.Numerics.IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IComparisonOperators<UIntPtr,UIntPtr,bool>,
System.Numerics.IDecrementOperators<UIntPtr>,
System.Numerics.IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IEqualityOperators<UIntPtr,UIntPtr,bool>,
System.Numerics.IIncrementOperators<UIntPtr>,
System.Numerics.IMinMaxValue<UIntPtr>,
System.Numerics.IModulusOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IMultiplicativeIdentity<UIntPtr,UIntPtr>,
System.Numerics.IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.INumber<UIntPtr>,
System.Numerics.INumberBase<UIntPtr>,
System.Numerics.IShiftOperators<UIntPtr,int,UIntPtr>,
System.Numerics.ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>,
System.Numerics.IUnaryNegationOperators<UIntPtr,UIntPtr>,
System.Numerics.IUnaryPlusOperators<UIntPtr,UIntPtr>,
System.Numerics.IUnsignedNumber<UIntPtr>,
System.Runtime.Serialization.ISerializable
```

Inheritance Object → ValueType → UIntPtr

Attributes CLSCompliantAttribute

Implements IComparable , IComparable<UIntPtr> , IEquatable<UIntPtr> , IFormattable , ISerializable , ISpanFormattable , IComparable<TSelf> , IEquatable<TSelf> , IParsable<UIntPtr> , IParsable<TSelf> , ISpanParsable<UIntPtr> , ISpanParsable<TSelf> , IAdditionOperators<UIntPtr,UIntPtr,UIntPtr> , IAdditionOperators<TSelf,TSelf,TSelf> , IAdditiveIdentity<UIntPtr,UIntPtr> , IAdditiveIdentity<TSelf,TSelf> , IBinaryInteger<UIntPtr> , IBinaryNumber<UIntPtr> , IBinaryNumber<TSelf> , IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr> , IBitwiseOperators<TSelf,TSelf,TSelf> , IComparisonOperators<UIntPtr,UIntPtr,Boolean> , IComparisonOperators<TSelf,TSelf,Boolean> , IDecrementOperators<UIntPtr> , IDecrementOperators<TSelf> , IDivisionOperators<UIntPtr,UIntPtr,UIntPtr> , IDivisionOperators<TSelf,TSelf,TSelf> , IEqualityOperators<UIntPtr,UIntPtr,Boolean> , IEqualityOperators<TSelf,TOther,TResult> , IEqualityOperators<TSelf,TSelf,Boolean> , IIncrementOperators<UIntPtr> , IIncrementOperators<TSelf> , IMinMaxValue<UIntPtr> , IModulusOperators<UIntPtr,UIntPtr,UIntPtr> , IModulusOperators<TSelf,TSelf,TSelf> , IMultiplicativeIdentity<UIntPtr,UIntPtr> , IMultiplicativeIdentity<TSelf,TSelf> , IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr> , IMultiplyOperators<TSelf,TSelf,TSelf> , INumber<UIntPtr> , INumber<TSelf> , INumberBase<UIntPtr> , INumberBase<TSelf> , IShiftOperators<UIntPtr,Int32,UIntPtr> , IShiftOperators<TSelf,Int32,TSelf> , ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr> , ISubtractionOperators<TSelf,TSelf,TSelf> , IUnaryNegationOperators<UIntPtr,UIntPtr> , IUnaryNegationOperators<TSelf,TSelf> , IUnaryPlusOperators<UIntPtr,UIntPtr> , IUnaryPlusOperators<TSelf,TSelf> , IUnsignedNumber<UIntPtr>

# Remarks

The UIntPtr type is designed to be an integer whose size is the same as a pointer. That is, an instance of this type is expected to be 32-bits in a 32-bit process and 64-bits in a 64-bit process.

The UIntPtr type can be used by languages that support pointers, and as a common means of referring to data between languages that do and do not support pointers. UIntPtr objects can also be used to hold handles.

> **Note**
>
> Using **UIntPtr** as a pointer or a handle is error prone and unsafe. It is simply an integer type that can be used as an interchange format for pointers and handles due to being the same size. Outside of specific interchange requirements, such as for passing data to a language that doesn't support pointers, a correctly typed pointer should be used to represent pointers and **SafeHandle** should be used to represent handles.

This type implements the ISerializable. In .NET 5 and later versions, this type also implements the IFormattable interfaces. In .NET 7 and later versions, this type also implements the IBinaryInteger<TSelf>, IMinMaxValue<TSelf>, and IUnsignedNumber<TSelf> interfaces.

In C# starting from version 9.0, you can use the built-in `nuint` type to define native-sized integers. This type is represented by the UIntPtr type internally and provides operations and conversions that are appropriate for integer types. For more information, see nint and nuint types.

In C# starting from version 11 and when targeting the .NET 7 or later runtime, `nuint` is an alias for UIntPtr in the same way that `uint` is an alias for UInt32.

# Constructors

| | |
|---|---|
| UIntPtr(UInt32) | Initializes a new instance of the UIntPtr structure using the specified 32-bit unsigned integer. |
| UIntPtr(UInt64) | Initializes a new instance of UIntPtr using the specified 64-bit unsigned integer. |
| UIntPtr(Void*) | Initializes a new instance of UIntPtr using the specified pointer to an unspecified type. |

# Fields

| | |
|---|---|
| Zero | A read-only field that represents an unsigned integer that has been initialized to zero. |

# Properties

| | |
|---|---|
| MaxValue | Represents the largest possible value of UIntPtr. |
| MinValue | Represents the smallest possible value of UIntPtr. |
| Size | Gets the size of this instance. |

# Methods

| | |
|---|---|
| Add(UIntPtr, Int32) | Adds an offset to an unsigned integer. |
| Clamp(UIntPtr, UIntPtr, UIntPtr) | Clamps a value to an inclusive minimum and maximum value. |
| CompareTo(Object) | Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object. |
| CompareTo(UIntPtr) | Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object. |
| CreateChecked<TOther>(TOther) | Creates an instance of the current type from a value, throwing an overflow exception for any values that fall outside the representable range of the current type. |
| CreateSaturating<TOther>(TOther) | Creates an instance of the current type from a value, saturating any values that fall outside the representable range of the current type. |
| CreateTruncating<TOther>(TOther) | Creates an instance of the current type from a value, truncating any values that fall outside the representable range of the current type. |
| DivRem(UIntPtr, UIntPtr) | Computes the quotient and remainder of two values. |
| Equals(Object) | Returns a value indicating whether this instance is equal to a specified object. |

| | |
|---|---|
| Equals(UIntPtr) | Indicates whether the current object is equal to another object of the same type. |
| GetHashCode() | Returns the hash code for this instance. |
| IsEvenInteger(UIntPtr) | Determines if a value represents an even integral number. |
| IsOddInteger(UIntPtr) | Determines if a value represents an odd integral number. |
| IsPow2(UIntPtr) | Determines if a value is a power of two. |
| LeadingZeroCount(UIntPtr) | Computes the number of leading zeros in a value. |
| Log2(UIntPtr) | Computes the log2 of a value. |
| Max(UIntPtr, UIntPtr) | Compares two values to compute which is greater. |
| Min(UIntPtr, UIntPtr) | Compares two values to compute which is lesser. |
| Parse(ReadOnlySpan<Char>, IFormatProvider) | Parses a span of characters into a value. |
| Parse(ReadOnlySpan<Char>, NumberStyles, IFormat Provider) | Converts the read-only span of characters representation of a number in optionally specified style and optionally specified culture-specific format to its unsigned native integer equivalent. |
| Parse(String) | Converts the string representation of a number to its unsigned native integer equivalent. |
| Parse(String, IFormatProvider) | Converts the string representation of a number in a specified culture-specific format to its unsigned native integer equivalent. |
| Parse(String, NumberStyles) | Converts the string representation of a number in a specified style to its unsigned native integer equivalent. |
| Parse(String, NumberStyles, IFormatProvider) | Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. |
| PopCount(UIntPtr) | Computes the number of bits that are set in a value. |
| RotateLeft(UIntPtr, Int32) | Rotates a value left by a given amount. |
| RotateRight(UIntPtr, Int32) | Rotates a value right by a given amount. |
| Sign(UIntPtr) | Computes the sign of a value. |
| Subtract(UIntPtr, Int32) | Subtracts an offset from an unsigned integer. |
| ToPointer() | Converts the value of this instance to a pointer to an unspecified type. |

| | |
|---|---|
| ToString() | Converts the numeric value of this instance to its equivalent string representation. |
| ToString(IFormatProvider) | Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information. |
| ToString(String) | Converts the numeric value of this instance to its equivalent string representation, using the specified format. |
| ToString(String, IFormat Provider) | Formats the value of the current instance using the specified format. |
| ToUInt32() | Converts the value of this instance to a 32-bit unsigned integer. |
| ToUInt64() | Converts the value of this instance to a 64-bit unsigned integer. |
| TrailingZeroCount(UIntPtr) | Computes the number of trailing zeros in a value. |
| TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider) | Tries to format the value of the current instance into the provided span of characters. |
| TryParse(ReadOnly Span<Char>, IFormatProvider, UIntPtr) | Tries to parse a string into a value. |
| TryParse(ReadOnly Span<Char>, NumberStyles, IFormatProvider, UIntPtr) | Converts the read-only span of characters representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded. |
| TryParse(ReadOnly Span<Char>, UIntPtr) | Converts the read-only span of characters representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded. |
| TryParse(String, IFormat Provider, UIntPtr) | Tries to parse a string into a value. |
| TryParse(String, NumberStyles, IFormatProvider, UIntPtr) | Converts the string representation of a number in a specified style and culture-specific format to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded. |
| TryParse(String, UIntPtr) | Converts the string representation of a number to its unsigned native integer equivalent. A return value indicates whether the conversion succeeded. |

# Operators

| | |
|---|---|
| Addition(UIntPtr, Int32) | Adds an offset to an unsigned integer. |
| Equality(UIntPtr, UIntPtr) | Determines whether two specified instances of UIntPtr are equal. |
| Explicit(UInt32 to UIntPtr) | Converts the value of a 32-bit unsigned integer to an UIntPtr. |
| Explicit(UInt64 to UIntPtr) | Converts the value of a 64-bit unsigned integer to an UIntPtr. |
| Explicit(UIntPtr to UInt32) | Converts the value of the specified UIntPtr to a 32-bit unsigned integer. |
| Explicit(UIntPtr to UInt64) | Converts the value of the specified UIntPtr to a 64-bit unsigned integer. |
| Explicit(UIntPtr to Void*) | Converts the value of the specified UIntPtr to a pointer to an unspecified type.<br><br>This API is not CLS-compliant. |
| Explicit(Void* to UIntPtr) | Converts the specified pointer to an unspecified type to an UIntPtr.<br><br>This API is not CLS-compliant. |
| Inequality(UIntPtr, UIntPtr) | Determines whether two specified instances of UIntPtr are not equal. |
| Subtraction(UIntPtr, Int32) | Subtracts an offset from an unsigned integer. |

# Explicit Interface Implementations

| | |
|---|---|
| IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.Addition(UIntPtr, UIntPtr) | Adds two values together to compute their sum. |
| IAdditionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedAddition(UIntPtr, UIntPtr) | Adds two values together to compute their sum. |
| IAdditiveIdentity<UIntPtr,UIntPtr>.AdditiveIdentity | Gets the additive identity of the current type. |
| IBinaryInteger<UIntPtr>.GetByteCount() | Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32). |
| IBinaryInteger<UIntPtr>.GetShortestBitLength() | Gets the length, in bits, of the shortest two's complement representation of the current value. |
| IBinaryInteger<UIntPtr>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, UIntPtr) | |

| | |
|---|---|
| IBinaryInteger<UIntPtr>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, UIntPtr) | |
| IBinaryInteger<UIntPtr>.TryWriteBigEndian(Span<Byte>, Int32) | Tries to write the current value, in big-endian format, to a given span. |
| IBinaryInteger<UIntPtr>.TryWriteLittleEndian(Span<Byte>, Int32) | Tries to write the current value, in little-endian format, to a given span. |
| IBinaryNumber<UIntPtr>.AllBitsSet | Gets an instance of the binary type in which all bits are set. |
| IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseAnd(UIntPtr, UIntPtr) | Computes the bitwise-and of two values. |
| IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.BitwiseOr(UIntPtr, UIntPtr) | Computes the bitwise-or of two values. |
| IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.ExclusiveOr(UIntPtr, UIntPtr) | Computes the exclusive-or of two values. |
| IBitwiseOperators<UIntPtr,UIntPtr,UIntPtr>.OnesComplement(UIntPtr) | Computes the ones-complement representation of a given value. |
| IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThan(UIntPtr, UIntPtr) | Compares two values to determine which is greater. |
| IComparisonOperators<UIntPtr,UIntPtr,Boolean>.GreaterThanOrEqual(UIntPtr, UIntPtr) | Compares two values to determine which is greater or equal. |
| IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThan(UIntPtr, UIntPtr) | Compares two values to determine which is less. |
| IComparisonOperators<UIntPtr,UIntPtr,Boolean>.LessThanOrEqual(UIntPtr, UIntPtr) | Compares two values to determine which is less or equal. |
| IDecrementOperators<UIntPtr>.CheckedDecrement(UIntPtr) | Decrements a value. |
| IDecrementOperators<UIntPtr>.Decrement(UIntPtr) | Decrements a value. |

| | |
|---|---|
| IDivisionOperators<UIntPtr,UIntPtr,UIntPtr>.Division(UIntPtr, UIntPtr) | Divides one value by another to compute their quotient. |
| IIncrementOperators<UIntPtr>.CheckedIncrement(UIntPtr) | Increments a value. |
| IIncrementOperators<UIntPtr>.Increment(UIntPtr) | Increments a value. |
| IMinMaxValue<UIntPtr>.MaxValue | Gets the maximum value of the current type. |
| IMinMaxValue<UIntPtr>.MinValue | Gets the minimum value of the current type. |
| IModulusOperators<UIntPtr,UIntPtr,UIntPtr>.Modulus(UIntPtr, UIntPtr) | Divides two values together to compute their modulus or remainder. |
| IMultiplicativeIdentity<UIntPtr,UIntPtr>.MultiplicativeIdentity | Gets the multiplicative identity of the current type. |
| IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedMultiply(UIntPtr, UIntPtr) | Multiplies two values together to compute their product. |
| IMultiplyOperators<UIntPtr,UIntPtr,UIntPtr>.Multiply(UIntPtr, UIntPtr) | Multiplies two values together to compute their product. |
| INumber<UIntPtr>.CopySign(UIntPtr, UIntPtr) | Copies the sign of a value to the sign of another value. |
| INumber<UIntPtr>.MaxNumber(UIntPtr, UIntPtr) | Compares two values to compute which is greater and returning the other value if an input is `NaN`. |
| INumber<UIntPtr>.MinNumber(UIntPtr, UIntPtr) | Compares two values to compute which is lesser and returning the other value if an input is `NaN`. |
| INumberBase<UIntPtr>.Abs(UIntPtr) | Computes the absolute of a value. |
| INumberBase<UIntPtr>.IsCanonical(UIntPtr) | Determines if a value is in its canonical representation. |
| INumberBase<UIntPtr>.IsComplexNumber(UIntPtr) | Determines if a value represents a complex number. |

| | |
|---|---|
| INumberBase<UIntPtr>.IsFinite(UIntPtr) | Determines if a value is finite. |
| INumberBase<UIntPtr>.IsImaginaryNumber(UIntPtr) | Determines if a value represents a pure imaginary number. |
| INumberBase<UIntPtr>.IsInfinity(UIntPtr) | Determines if a value is infinite. |
| INumberBase<UIntPtr>.IsInteger(UIntPtr) | Determines if a value represents an integral number. |
| INumberBase<UIntPtr>.IsNaN(UIntPtr) | Determines if a value is NaN. |
| INumberBase<UIntPtr>.IsNegative(UIntPtr) | Determines if a value is negative. |
| INumberBase<UIntPtr>.IsNegativeInfinity(UIntPtr) | Determines if a value is negative infinity. |
| INumberBase<UIntPtr>.IsNormal(UIntPtr) | Determines if a value is normal. |
| INumberBase<UIntPtr>.IsPositive(UIntPtr) | Determines if a value is positive. |
| INumberBase<UIntPtr>.IsPositiveInfinity(UIntPtr) | Determines if a value is positive infinity. |
| INumberBase<UIntPtr>.IsRealNumber(UIntPtr) | Determines if a value represents a real number. |
| INumberBase<UIntPtr>.IsSubnormal(UIntPtr) | Determines if a value is subnormal. |
| INumberBase<UIntPtr>.IsZero(UIntPtr) | Determines if a value is zero. |
| INumberBase<UIntPtr>.MaxMagnitude(UIntPtr, UIntPtr) | Compares two values to compute which is greater. |
| INumberBase<UIntPtr>.MaxMagnitudeNumber(UIntPtr, UIntPtr) | Compares two values to compute which has the greater magnitude and returning the other value if an input is `NaN`. |
| INumberBase<UIntPtr>.MinMagnitude(UIntPtr, UIntPtr) | Compares two values to compute which is lesser. |
| INumberBase<UIntPtr>.MinMagnitudeNumber(UIntPtr, UIntPtr) | Compares two values to compute which has the lesser magnitude and returning the other value if an input is `NaN`. |

| | |
|---|---|
| INumberBase<UIntPtr>.One | Gets the value `1` for the type. |
| INumberBase<UIntPtr>.Radix | Gets the radix, or base, for the type. |
| INumberBase<UIntPtr>.TryConvertFromChecked<TOther>(TOther, UIntPtr) | |
| INumberBase<UIntPtr>.TryConvertFromSaturating<TOther>(TOther, UIntPtr) | |
| INumberBase<UIntPtr>.TryConvertFromTruncating<TOther>(TOther, UIntPtr) | |
| INumberBase<UIntPtr>.TryConvertToChecked<TOther>(UIntPtr, TOther) | Tries to convert an instance of the the current type to another type, throwing an overflow exception for any values that fall outside the representable range of the current type. |
| INumberBase<UIntPtr>.TryConvertToSaturating<TOther>(UIntPtr, TOther) | Tries to convert an instance of the the current type to another type, saturating any values that fall outside the representable range of the current type. |
| INumberBase<UIntPtr>.TryConvertToTruncating<TOther>(UIntPtr, TOther) | Tries to convert an instance of the the current type to another type, truncating any values that fall outside the representable range of the current type. |
| INumberBase<UIntPtr>.Zero | Gets the value `0` for the type. |
| ISerializable.GetObjectData(SerializationInfo, StreamingContext) | Populates a SerializationInfo object with the data needed to serialize the current UIntPtr object. |
| IShiftOperators<UIntPtr,Int32,UIntPtr>.LeftShift(UIntPtr, Int32) | Shifts a value left by a given amount. |
| IShiftOperators<UIntPtr,Int32,UIntPtr>.RightShift(UIntPtr, Int32) | Shifts a value right by a given amount. |
| IShiftOperators<UIntPtr,Int32,UIntPtr>.UnsignedRightShift(UIntPtr, Int32) | Shifts a value right by a given amount. |
| ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.CheckedSubtraction(UIntPtr, UIntPtr) | Subtracts two values to compute their difference. |
| ISubtractionOperators<UIntPtr,UIntPtr,UIntPtr>.Subtraction(UIntPtr, UIntPtr) | Subtracts two values to compute their difference. |
| IUnaryNegationOperators<UIntPtr,UInt | Computes the checked unary negation of a value. |

| | |
|---|---|
| Ptr>.CheckedUnary Negation(UIntPtr) | |
| IUnaryNegation Operators<UIntPtr,UInt Ptr>.UnaryNegation(UIntPtr) | Computes the unary negation of a value. |
| IUnaryPlusOperators<UInt Ptr,UIntPtr>.UnaryPlus(UInt Ptr) | Computes the unary plus of a value. |

# Applies to

| Product | Versions |
|---|---|
| **.NET** | Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8 |
| **.NET Framework** | 1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1 |
| **.NET Standard** | 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1 |
| **UWP** | 10.0 |
| **Xamarin.iOS** | 10.8 |
| **Xamarin.Mac** | 3.0 |

# Thread Safety

This type is thread safe.

# See also

- IntPtr