Char Struct

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents a character as a UTF-16 code unit.

F#

```
type char = struct
    interface IConvertible
    interface IFormattable
    interface IParsable<char>
    interface ISpanFormattable
    interface ISpanParsable<char>
    interface IAdditionOperators<char, char, char>
    interface IAdditiveIdentity<char, char>
    interface IBinaryInteger<char>
    interface IBinaryNumber<char>
    interface IBitwiseOperators<char, char, char>
    interface IComparisonOperators<char, char, bool>
    interface IEqualityOperators<char, char, bool>
    interface IDecrementOperators<char>
    interface IDivisionOperators<char, char, char>
    interface IIncrementOperators<char>
    interface IModulusOperators<char, char, char>
    interface IMultiplicativeIdentity<char, char>
    interface IMultiplyOperators<char, char, char>
    interface INumber<char>
    interface INumberBase<char>
    interface ISubtractionOperators<char, char, char>
    interface IUnaryNegationOperators<char, char>
    interface IUnaryPlusOperators<char, char>
    interface IShiftOperators<char, int, char>
    interface IMinMaxValue<char>
    interface IUnsignedNumber<char>
```

Inheritance Object → ValueType → Char

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

Examples

The following code example demonstrates some of the methods in Char.

F#

```
open System

let chA = 'A'
let ch1 = '1'
let str = "test string"

printfn $"{chA.CompareTo 'B'}"          //-----------  Output: "-1" (meaning 'A' is 1 less than 'B')
printfn $"{chA.Equals 'A'}"             //-----------  Output: "True"
printfn $"{Char.GetNumericValue ch1}"   //-----------  Output: "1"
printfn $"{Char.IsControl '\t'}"        //-----------  Output: "True"
printfn $"{Char.IsDigit ch1}"           //-----------  Output: "True"
printfn $"{Char.IsLetter ','}"          //-----------  Output: "False"
printfn $"{Char.IsLower 'u'}"           //-----------  Output: "True"
printfn $"{Char.IsNumber ch1}"          //-----------  Output: "True"
printfn $"{Char.IsPunctuation '.'}"     //-----------  Output: "True"
printfn $"{Char.IsSeparator(str, 4)}"   //-----------  Output: "True"
printfn $"{Char.IsSymbol '+'}"          //-----------  Output: "True"
printfn $"{Char.IsWhiteSpace(str, 4)}"  //-----------  Output: "True"
printfn $"""{Char.Parse "S"}"""         //-----------  Output: "S"
printfn $"{Char.ToLower 'M'}"           //-----------  Output: "m"
printfn $"{'x'}"                        //-----------  Output: "x"
```

Remarks

.NET uses the Char structure to represent Unicode code points by using UTF-16 encoding. The value of a Char object is its 16-bit numeric (ordinal) value.

If you aren't familiar with Unicode, scalar values, code points, surrogate pairs, UTF-16, and the Rune type, see Introduction to character encoding in .NET.

The following sections examine the relationship between a Char object and a character and discuss some common tasks performed with Char instances. We recommend that you consider the Rune type, introduced in .NET Core 3.0, as an alternative to Char for performing some of these tasks.

- Char objects, Unicode characters, and strings
- Characters and character categories
- Characters and text elements
- Common operations
- Char values and interop

Char objects, Unicode characters, and strings

A String object is a sequential collection of Char structures that represents a string of text. Most Unicode characters can be represented by a single Char object, but a character that is encoded as a base character, surrogate pair, and/or combining character sequence is represented by multiple Char objects. For this reason, a Char structure in a String object is not necessarily equivalent to a single Unicode character.

Multiple 16-bit code units are used to represent single Unicode characters in the following cases:

- Glyphs, which may consist of a single character or of a base character followed by one or more combining characters. For example, the character ä is represented by a Char object whose code unit is U+0061 followed by a Char object whose code unit is U+0308. (The character ä can also be defined by a single Char object that has a code unit of U+00E4.) The following example illustrates that the character ä consists of two Char objects.

F#

```fsharp
open System
open System.IO

let sw = new StreamWriter("chars1.txt")
let chars = [| '\u0061'; '\u0308' |]
let string = String chars
sw.WriteLine string
sw.Close()

// The example produces the following output:
//       ä
```

- Characters outside the Unicode Basic Multilingual Plane (BMP). Unicode supports sixteen planes in addition to the BMP, which represents plane 0. A Unicode code point is represented in UTF-32 by a 21-bit value that includes the plane. For example, U+1D160 represents the MUSICAL SYMBOL EIGHTH NOTE character. Because UTF-16 encoding has only 16 bits, characters outside the BMP are represented by surrogate pairs in UTF-16. The following example illustrates that the UTF-32 equivalent of U+1D160, the MUSICAL SYMBOL EIGHTH NOTE character, is U+D834 U+DD60. U+D834 is the high surrogate; high surrogates range from U+D800 through U+DBFF. U+DD60 is the low surrogate; low surrogates range from U+DC00 through U+DFFF.

F#

```fsharp
open System
open System.IO

let showCodePoints (value: char seq) =
    let str =
        value
        |> Seq.map (fun ch -> $"U+{Convert.ToUInt16 ch:X4}")
        |> String.concat ""
    str.Trim()

let sw = new StreamWriter(@".\chars2.txt")
let utf32 = 0x1D160
let surrogate = Char.ConvertFromUtf32 utf32
sw.WriteLine $"U+{utf32:X6} UTF-32 = {surrogate} ({showCodePoints surrogate}) UTF-16"
sw.Close()
```

```
// The example produces the following output:
//       U+01D160 UTF-32 = ð (U+D834 U+DD60) UTF-16
```

Characters and character categories

Each Unicode character or valid surrogate pair belongs to a Unicode category. In .NET, Unicode categories are represented by members of the UnicodeCategory enumeration and include values such as UnicodeCategory.CurrencySymbol, UnicodeCategory.LowercaseLetter, and UnicodeCategory.SpaceSeparator, for example.

To determine the Unicode category of a character, call the GetUnicodeCategory method. For example, the following example calls the GetUnicodeCategory to display the Unicode category of each character in a string. The example works correctly only if there are no surrogate pairs in the String instance.

F#

```
open System

// Define a string with a variety of character categories.
let s = "The red car drove down the long, narrow, secluded road."
// Determine the category of each character.
for ch in s do
    printfn $"'{ch}': {Char.GetUnicodeCategory ch}"

// The example displays the following output:
//       'T': UppercaseLetter
//       'h': LowercaseLetter
//       'e': LowercaseLetter
//       ' ': SpaceSeparator
//       'r': LowercaseLetter
//       'e': LowercaseLetter
//       'd': LowercaseLetter
//       ' ': SpaceSeparator
//       'c': LowercaseLetter
//       'a': LowercaseLetter
//       'r': LowercaseLetter
//       ' ': SpaceSeparator
//       'd': LowercaseLetter
//       'r': LowercaseLetter
//       'o': LowercaseLetter
//       'v': LowercaseLetter
//       'e': LowercaseLetter
//      ' ': SpaceSeparator
//      'd': LowercaseLetter
//      'o': LowercaseLetter
//      'w': LowercaseLetter
//      'n': LowercaseLetter
//      ' ': SpaceSeparator
//      't': LowercaseLetter
//      'h': LowercaseLetter
//      'e': LowercaseLetter
//      ' ': SpaceSeparator
//      'l': LowercaseLetter
//      'o': LowercaseLetter
//      'n': LowercaseLetter
//      'g': LowercaseLetter
//      ',': OtherPunctuation
//      ' ': SpaceSeparator
//      'n': LowercaseLetter
//      'a': LowercaseLetter
//      'r': LowercaseLetter
//      'r': LowercaseLetter
//      'o': LowercaseLetter
//      'w': LowercaseLetter
//      ',': OtherPunctuation
//      ' ': SpaceSeparator
//      's': LowercaseLetter
//      'e': LowercaseLetter
//      'c': LowercaseLetter
//      'l': LowercaseLetter
//      'u': LowercaseLetter
//      'd': LowercaseLetter
//      'e': LowercaseLetter
//      'd': LowercaseLetter
//      ' ': SpaceSeparator
//      'r': LowercaseLetter
//      'o': LowercaseLetter
//      'a': LowercaseLetter
//      'd': LowercaseLetter
//      '.': OtherPunctuation
```

Internally, for characters outside the ASCII range (U+0000 through U+00FF), the GetUnicodeCategory method depends on Unicode categories reported by the CharUnicodeInfo class. Starting with .NET Framework 4.6.2, Unicode characters are classified based on The Unicode Standard, Version 8.0.0 . In versions of the .NET Framework from .NET Framework 4 to .NET Framework 4.6.1, they are classified based on The Unicode Standard, Version 6.3.0 .

Characters and text elements

Because a single character can be represented by multiple Char objects, it is not always meaningful to work with individual Char objects. For instance, the following example converts the Unicode code points that represent the Aegean numbers zero through 9 to UTF-16 encoded code units. Because it erroneously equates Char objects with characters, it inaccurately reports that the resulting string has 20 characters.

F#

```fsharp
open System

let result =
    [ for i in 0x10107..0x10110 do  // Range of Aegean numbers.
        Char.ConvertFromUtf32 i ]
    |> String.concat ""

printfn $"The string contains {result.Length} characters."


// The example displays the following output:
//     The string contains 20 characters.
```

You can do the following to avoid the assumption that a Char object represents a single character:

- You can work with a String object in its entirety instead of working with its individual characters to represent and analyze linguistic content.
- You can use String.EnumerateRunes as shown in the following example:

  F#

  ```fsharp
  let countLetters (s: string) =
      let mutable letterCount = 0

      for rune in s.EnumerateRunes() do
          if Rune.IsLetter rune then
              letterCount <- letterCount + 1

      letterCount
  ```

- You can use the StringInfo class to work with text elements instead of individual Char objects. The following example uses the StringInfo object to count the number of text elements in a string that consists of the Aegean numbers zero through nine. Because it considers a surrogate pair a single character, it correctly reports that the string contains ten characters.

  F#

  ```fsharp
  open System
  open System.Globalization

  let result =
      [ for i in 0x10107..0x10110 do  // Range of Aegean numbers.
          Char.ConvertFromUtf32 i ]
      |> String.concat ""


  let si = StringInfo result
  printfn $"The string contains {si.LengthInTextElements} characters."

  // The example displays the following output:
  //       The string contains 10 characters.
  ```

- If a string contains a base character that has one or more combining characters, you can call the String.Normalize method to convert the substring to a single UTF-16 encoded code unit. The following example calls the String.Normalize method to convert the base character U+0061 (LATIN SMALL LETTER A) and combining character U+0308 (COMBINING DIAERESIS) to U+00E4 (LATIN SMALL LETTER A WITH DIAERESIS).

  F#

  ```fsharp
  open System

  let showString (s: string) =
      printf $"Length of string: {s.Length} ("
      for i = 0 to s.Length - 1 do
          printf $"U+{Convert.ToUInt16 s[i]:X4}"
          if i <> s.Length - 1 then printf " "
      printfn ")\n"

  let combining = "\u0061\u0308"
  ```

```
showString combining

let normalized = combining.Normalize()
showString normalized

// The example displays the following output:
//         Length of string: 2 (U+0061 U+0308)
//
//         Length of string: 1 (U+00E4)
```

Common operations

The Char structure provides methods to compare Char objects, convert the value of the current Char object to an object of another type, and determine the Unicode category of a Char object:

To do this	**Use these `System.Char` methods**
Compare Char objects	CompareTo and Equals
Convert a code point to a string	ConvertFromUtf32 See also the Rune type.
Convert a Char object or a surrogate pair of Char objects to a code point	For a single character: Convert.ToInt32(Char) For a surrogate pair or a character in a string: Char.ConvertToUtf32 See also the Rune type.
Get the Unicode category of a character	GetUnicodeCategory See also Rune.GetUnicodeCategory.
Determine whether a character is in a particular Unicode category such as digit, letter, punctuation, control character, and so on	IsControl, IsDigit, IsHighSurrogate, IsLetter, IsLetterOrDigit, IsLower, IsLowSurrogate, IsNumber, IsPunctuation, IsSeparator, IsSurrogate, IsSurrogatePair, IsSymbol, IsUpper, and IsWhiteSpace See also corresponding methods on the Rune type.
Convert a Char object that	GetNumericValue

represents a number to a numeric value type	See also Rune.GetNumericValue.
Convert a character in a string into a Char object	Parse and TryParse
Convert a Char object to a String object	ToString
Change the case of a Char object	ToLower, ToLowerInvariant, ToUpper, and ToUpperInvariant See also corresponding methods on the Rune type.

Char values and interop

When a managed Char type, which is represented as a Unicode UTF-16 encoded code unit, is passed to unmanaged code, the interop marshaler converts the character set to ANSI by default. You can apply the DllImportAttribute attribute to platform invoke declarations and the StructLayoutAttribute attribute to a COM interop declaration to control which character set a marshaled Char type uses.

Fields

MaxValue	Represents the largest possible value of a Char. This field is constant.
MinValue	Represents the smallest possible value of a Char. This field is constant.

Methods

CompareTo(Char)	Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.
CompareTo(Object)	Compares this instance to a specified object and indicates

	whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.
ConvertFromUtf32(Int32)	Converts the specified Unicode code point into a UTF-16 encoded string.
ConvertToUtf32(Char, Char)	Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.
ConvertToUtf32(String, Int32)	Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.
Equals(Char)	Returns a value that indicates whether this instance is equal to the specified Char object.
Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.
GetHashCode()	Returns the hash code for this instance.
GetNumericValue(Char)	Converts the specified numeric Unicode character to a double-precision floating point number.
GetNumericValue(String, Int32)	Converts the numeric Unicode character at the specified position in a specified string to a double-precision floating point number.
GetTypeCode()	Returns the TypeCode for value type Char.
GetUnicodeCategory(Char)	Categorizes a specified Unicode character into a group identified by one of the UnicodeCategory values.
GetUnicodeCategory(String, Int32)	Categorizes the character at the specified position in a specified string into a group identified by one of the UnicodeCategory values.
IsAscii(Char)	Returns `true` if `c` is an ASCII character ([U+0000..U+007F]).
IsAsciiDigit(Char)	Indicates whether a character is categorized as an ASCII digit.
IsAsciiHexDigit(Char)	Indicates whether a character is categorized as an ASCII hexademical digit.
IsAsciiHexDigitLower(Char)	Indicates whether a character is categorized as an ASCII lower-case hexademical digit.

IsAsciiHexDigitUpper(Char)	Indicates whether a character is categorized as an ASCII upper-case hexademical digit.
IsAsciiLetter(Char)	Indicates whether a character is categorized as an ASCII letter.
IsAsciiLetterLower(Char)	Indicates whether a character is categorized as a lowercase ASCII letter.
IsAsciiLetterOrDigit(Char)	Indicates whether a character is categorized as an ASCII letter or digit.
IsAsciiLetterUpper(Char)	Indicates whether a character is categorized as an uppercase ASCII letter.
IsBetween(Char, Char, Char)	Indicates whether a character is within the specified inclusive range.
IsControl(Char)	Indicates whether the specified Unicode character is categorized as a control character.
IsControl(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a control character.
IsDigit(Char)	Indicates whether the specified Unicode character is categorized as a decimal digit.
IsDigit(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a decimal digit.
IsHighSurrogate(Char)	Indicates whether the specified Char object is a high surrogate.
IsHighSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a high surrogate.
IsLetter(Char)	Indicates whether the specified Unicode character is categorized as a Unicode letter.
IsLetter(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a Unicode letter.
IsLetterOrDigit(Char)	Indicates whether the specified Unicode character is categorized as a letter or a decimal digit.
IsLetterOrDigit(String, Int32)	Indicates whether the character at the specified position in a

	specified string is categorized as a letter or a decimal digit.
IsLower(Char)	Indicates whether the specified Unicode character is categorized as a lowercase letter.
IsLower(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a lowercase letter.
IsLowSurrogate(Char)	Indicates whether the specified Char object is a low surrogate.
IsLowSurrogate(String, Int32)	Indicates whether the Char object at the specified position in a string is a low surrogate.
IsNumber(Char)	Indicates whether the specified Unicode character is categorized as a number.
IsNumber(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a number.
IsPunctuation(Char)	Indicates whether the specified Unicode character is categorized as a punctuation mark.
IsPunctuation(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a punctuation mark.
IsSeparator(Char)	Indicates whether the specified Unicode character is categorized as a separator character.
IsSeparator(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a separator character.
IsSurrogate(Char)	Indicates whether the specified character has a surrogate code unit.
IsSurrogate(String, Int32)	Indicates whether the character at the specified position in a specified string has a surrogate code unit.
IsSurrogatePair(Char, Char)	Indicates whether the two specified Char objects form a surrogate pair.
IsSurrogatePair(String, Int32)	Indicates whether two adjacent Char objects at a specified position in a string form a surrogate pair.
IsSymbol(Char)	Indicates whether the specified Unicode character is categorized as a symbol character.

IsSymbol(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as a symbol character.
IsUpper(Char)	Indicates whether the specified Unicode character is categorized as an uppercase letter.
IsUpper(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as an uppercase letter.
IsWhiteSpace(Char)	Indicates whether the specified Unicode character is categorized as white space.
IsWhiteSpace(String, Int32)	Indicates whether the character at the specified position in a specified string is categorized as white space.
Parse(String)	Converts the value of the specified string to its equivalent Unicode character.
ToLower(Char)	Converts the value of a Unicode character to its lowercase equivalent.
ToLower(Char, CultureInfo)	Converts the value of a specified Unicode character to its lowercase equivalent using specified culture-specific formatting information.
ToLowerInvariant(Char)	Converts the value of a Unicode character to its lowercase equivalent using the casing rules of the invariant culture.
ToString()	Converts the value of this instance to its equivalent string representation.
ToString(Char)	Converts the specified Unicode character to its equivalent string representation.
ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation using the specified culture-specific format information.
ToUpper(Char)	Converts the value of a Unicode character to its uppercase equivalent.
ToUpper(Char, CultureInfo)	Converts the value of a specified Unicode character to its uppercase equivalent using specified culture-specific formatting information.
ToUpperInvariant(Char)	Converts the value of a Unicode character to its uppercase

	equivalent using the casing rules of the invariant culture.
TryParse(String, Char)	Converts the value of the specified string to its equivalent Unicode character. A return code indicates whether the conversion succeeded or failed.

Explicit Interface Implementations

IAdditionOperators<Char,Char,Char>.Addition(Char, Char)	
IAdditionOperators<Char,Char,Char>.CheckedAddition(Char, Char)	
IAdditiveIdentity<Char,Char>.AdditiveIdentity	
IBinaryInteger<Char>.GetByteCount()	Gets the number of bytes that will be written as part of TryWriteLittleEndian(Span<Byte>, Int32).
IBinaryInteger<Char>.GetShortestBitLength()	Gets the length, in bits, of the shortest two's complement representation of the current value.
IBinaryInteger<Char>.LeadingZeroCount(Char)	
IBinaryInteger<Char>.PopCount(Char)	
IBinaryInteger<Char>.RotateLeft(Char, Int32)	
IBinaryInteger<Char>.RotateRight(Char, Int32)	
IBinaryInteger<Char>.TrailingZeroCount(Char)	
IBinaryInteger<Char>.TryReadBigEndian(ReadOnlySpan<Byte>, Boolean, Char)	
IBinaryInteger<Char>.TryReadLittleEndian(ReadOnlySpan<Byte>, Boolean, Char)	
IBinaryInteger<Char>.TryWriteBigEndian(Span<Byte>, Int32)	Tries to write the current value, in big-endian format, to a given span.
IBinaryInteger<Char>.TryWriteLittleEndian(Span<Byte>, Int32)	Tries to write the current value, in little-endian format, to a given span.
IBinaryNumber<Char>.AllBitsSet	

IBinaryNumber<Char>.IsPow2(Char)	
IBinaryNumber<Char>.Log2(Char)	
IBitwiseOperators<Char,Char,Char>.BitwiseAnd(Char, Char)	
IBitwiseOperators<Char,Char,Char>.BitwiseOr(Char, Char)	
IBitwiseOperators<Char,Char,Char>.ExclusiveOr(Char, Char)	
IBitwiseOperators<Char,Char,Char>.OnesComplement(Char)	
IComparisonOperators<Char,Char,Boolean>.GreaterThan(Char, Char)	
IComparisonOperators<Char,Char,Boolean>.GreaterThanOrEqual(Char, Char)	
IComparisonOperators<Char,Char,Boolean>.LessThan(Char, Char)	
IComparisonOperators<Char,Char,Boolean>.LessThanOrEqual(Char, Char)	
IConvertible.ToBoolean(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToByte(IFormatProvider)	For a description of this member, see ToByte(IFormatProvider).
IConvertible.ToChar(IFormatProvider)	For a description of this member, see ToChar(IFormatProvider).
IConvertible.ToDateTime(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToDecimal(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToDouble(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToInt16(IFormatProvider)	For a description of this member, see ToInt16(IFormatProvider).
IConvertible.ToInt32(IFormatProvider)	For a description of this member, see ToInt32(IFormatProvider).
IConvertible.ToInt64(IFormatProvider)	For a description of this member, see ToInt64(IFormatProvider).

IConvertible.ToSByte(IFormatProvider)	For a description of this member, see ToSByte(IFormatProvider).
IConvertible.ToSingle(IFormatProvider)	**Note** This conversion is not supported. Attempting to do so throws an InvalidCastException.
IConvertible.ToType(Type, IFormatProvider)	For a description of this member, see ToType(Type, IFormatProvider).
IConvertible.ToUInt16(IFormatProvider)	For a description of this member, see ToUInt16(IFormatProvider).
IConvertible.ToUInt32(IFormatProvider)	For a description of this member, see ToUInt32(IFormatProvider).
IConvertible.ToUInt64(IFormatProvider)	For a description of this member, see ToUInt64(IFormatProvider).
IDecrementOperators<Char>.CheckedDecrement(Char)	
IDecrementOperators<Char>.Decrement(Char)	
IDivisionOperators<Char,Char,Char>.Division(Char, Char)	
IEqualityOperators<Char,Char,Boolean>.Equality(Char, Char)	
IEqualityOperators<Char,Char,Boolean>.Inequality(Char, Char)	
IFormattable.ToString(String, IFormatProvider)	Formats the value of the current instance using the specified format.
IIncrementOperators<Char>.CheckedIncrement(Char)	
IIncrementOperators<Char>.Increment(Char)	
IMinMaxValue<Char>.MaxValue	
IMinMaxValue<Char>.MinValue	
IModulusOperators<Char,Char,Char>.Modulus(Char, Char)	
IMultiplicativeIdentity<Char,Char>.MultiplicativeIdentity	
IMultiplyOperators<Char,Char,Char>.CheckedMultiply(Char, Char)	
IMultiplyOperators<Char,Char,Char>.Multiply(Char, Char)	

INumberBase<Char>.Abs(Char)
INumberBase<Char>.IsCanonical(Char)
INumberBase<Char>.IsComplexNumber(Char)
INumberBase<Char>.IsEvenInteger(Char)
INumberBase<Char>.IsFinite(Char)
INumberBase<Char>.IsImaginaryNumber(Char)
INumberBase<Char>.IsInfinity(Char)
INumberBase<Char>.IsInteger(Char)
INumberBase<Char>.IsNaN(Char)
INumberBase<Char>.IsNegative(Char)
INumberBase<Char>.IsNegativeInfinity(Char)
INumberBase<Char>.IsNormal(Char)
INumberBase<Char>.IsOddInteger(Char)
INumberBase<Char>.IsPositive(Char)
INumberBase<Char>.IsPositiveInfinity(Char)
INumberBase<Char>.IsRealNumber(Char)
INumberBase<Char>.IsSubnormal(Char)
INumberBase<Char>.IsZero(Char)
INumberBase<Char>.MaxMagnitude(Char, Char)
INumberBase<Char>.MaxMagnitudeNumber(Char, Char)
INumberBase<Char>.MinMagnitude(Char, Char)
INumberBase<Char>.MinMagnitudeNumber(Char, Char)
INumberBase<Char>.One
INumberBase<Char>.Parse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider)

INumberBase<Char>.Parse(String, NumberStyles, IFormatProvider)	
INumberBase<Char>.Radix	
INumberBase<Char>.TryConvertFromChecked<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertFromSaturating<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertFromTruncating<TOther>(TOther, Char)	
INumberBase<Char>.TryConvertToChecked<TOther>(Char, TOther)	
INumberBase<Char>.TryConvertToSaturating<TOther>(Char, TOther)	
INumberBase<Char>.TryConvertToTruncating<TOther>(Char, TOther)	
INumberBase<Char>.TryParse(ReadOnlySpan<Char>, NumberStyles, IFormatProvider, Char)	
INumberBase<Char>.TryParse(String, NumberStyles, IFormatProvider, Char)	
INumberBase<Char>.Zero	
IParsable<Char>.Parse(String, IFormatProvider)	
IParsable<Char>.TryParse(String, IFormatProvider, Char)	
IShiftOperators<Char,Int32,Char>.LeftShift(Char, Int32)	
IShiftOperators<Char,Int32,Char>.RightShift(Char, Int32)	
IShiftOperators<Char,Int32,Char>.UnsignedRightShift(Char, Int32)	
ISpanFormattable.TryFormat(Span<Char>, Int32, ReadOnlySpan<Char>, IFormatProvider)	Tries to format the value of the current instance into the provided span of characters.
ISpanParsable<Char>.Parse(ReadOnlySpan<Char>, IFormatProvider)	
ISpanParsable<Char>.TryParse(ReadOnlySpan<Char>, IFormatProvider, Char)	
ISubtractionOperators<Char,Char,Char>.CheckedSubtraction(Char, Char)	
ISubtractionOperators<Char,Char,Char>.Subtraction(Char, Char)	
IUnaryNegationOperators<Char,Char>.CheckedUnaryNegation(Char)	

IUnaryNegationOperators<Char,Char>.UnaryNegation(Char)
IUnaryPlusOperators<Char,Char>.UnaryPlus(Char)

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

All members of this type are thread safe. Members that appear to modify instance state actually return a new instance initialized with the new value. As with any other type, reading and writing to a shared variable that contains an instance of this type must be protected by a lock to guarantee thread safety.

See also

- IComparable
- IConvertible
- String