# Rational.Parse Method

名前空間: WS.Theia.ExtremelyPrecise

アセンブリ: ExtremelyPrecise.dll

数値の文字列形式を、それと等価の Rational に変換します。

# オーバーロード

|  |  |
| --- | --- |
| Parse(String) | 数値の文字列形式を、それと等価の Rational に変換します。 |
| Parse(String,NumberStyles) | 指定のスタイルで表現された数値の文字列形式を、それと等価な Rational に変換します。 |
| Parse(String,IFormatProvider) | 指定されたカルチャ固有の書式で表現された文字列形式の数値を、それと等価の Rational に変換します。 |
| Parse(String,NumberStyles,IFormatProvider) | 指定したスタイルおよびカルチャ固有の書式の数値の文字列形式を、それと等価の Rational に変換します。 |

# Parse(String)

数値の文字列形式を、それと等価の Rational に変換します。

public static WS.Theia.ExtremelyPrecise.Rational Parse(String value);

## パラメーター

value　String  
変換する数値を含んだ文字列。

## 戻り値

Rational

value パラメーターで指定されている数値と等価の値。

## 例外

ArgumentNullException  
valueはnullです。

FormatException  
valueが正しい形式ではありません。

## 例

次の例ではParse(String)メソッドを使って2つのRationalオブジェクトのインスタンスを生成しています。その後各オブジェクトを乗算しCompareメソッドで2つの値の大小を判定しています。

string stringToParse = String.Empty;  
try  
{  
 // Parse two strings.  
 string string1, string2;  
 string1 = "12347534159895123";  
 string2 = "987654321357159852";  
 stringToParse = string1;  
 Rational number1 = Rational.Parse(stringToParse);  
 Console.WriteLine("Converted '{0}' to {1:N0}.", stringToParse, number1);   
 stringToParse = string2;  
 Rational number2 = Rational.Parse(stringToParse);  
 Console.WriteLine("Converted '{0}' to {1:N0}.", stringToParse, number2);  
 // Perform arithmetic operations on the two numbers.  
 number1 \*= 3;  
 number2 \*= 2;  
 // Compare the numbers.  
 int result = Rational.Compare(number1, number2);  
 switch (result)  
 {  
 case -1:  
 Console.WriteLine("{0} is greater than {1}.", number2, number1);  
 break;  
 case 0:  
 Console.WriteLine("{0} is equal to {1}.", number1, number2);  
 break;  
 case 1:  
 Console.WriteLine("{0} is greater than {1}.", number1, number2);  
 break;  
 }   
}  
catch (FormatException)  
{  
 Console.WriteLine("Unable to parse {0}.", stringToParse);  
}  
// The example displays the following output:  
// Converted '12347534159895123' to 12,347,534,159,895,123.  
// Converted '987654321357159852' to 987,654,321,357,159,852.  
// 1975308642714319704 is greater than 37042602479685369.

## 注釈

valueパラメーターは、次の形式で表された数字文字列でなければなりません。

[ws][sign]digits[ws]

角カッコ（[および]）内の要素は省略可能です。それぞれの要素は次の表のとおりです。

|  |  |
| --- | --- |
| 要素 | 説明 |
| ws | 空白文字。省略可能です。 |
| sign | 符号。省略可能です。有効な文字はカレントカルチャーのNumberFormatInfo.NegativeSignとNumberFormatInfo.PositiveSignプロパティによって決まります。 |
| digits | 数字列。0から9及び小数点で構成している必要があります。先頭の0は無視します。有効な小数点はカレントカルチャーのNumberFormatInfo.NumberDecimalSeparatorプロパティによって決まります。 |

# Parse(String,NumberStyles)

指定のスタイルで表現された数値の文字列形式を、それと等価な Rational に変換します。

public static WS.Theia.ExtremelyPrecise.Rational Parse(String value ,NumberStyles style);

## パラメーター

value　String  
変換する数値を含んだ文字列。

style　 NumberStyles

value に許可されている書式を指定する列挙値のビットごとの組み合わせ。

## 戻り値

Rational

value パラメーターで指定されている数値と等価の値。

## 例外

ArgumentException

styleがNumberStyle値ではない、または、seyleにAllowHexSpecifier または HexNumber フラグが含まれます。

ArgumentNullException  
valueはnullです。

FormatException  
valueが正しい形式ではありません。

## 例

次の例ではParse(String,NumberStyle)メソッドに、NumberStyleとして16進数値として文字列を解釈させる値、スペースを許可する値、符号を許可する値を与えて呼び出しています。

Rational number;   
// Method should succeed (white space and sign allowed)  
number = Rational.Parse(" -68054 ", NumberStyles.Integer);  
Console.WriteLine(number);  
// Method should succeed (string interpreted as hexadecimal)  
number = Rational.Parse("68054", NumberStyles.AllowHexSpecifier);  
Console.WriteLine(number);  
// Method call should fail: sign not allowed  
try  
{  
 number = Rational.Parse(" -68054 ", NumberStyles.AllowLeadingWhite | NumberStyles.AllowTrailingWhite);  
 Console.WriteLine(number);  
}   
catch (FormatException e)  
{  
 Console.WriteLine(e.Message);  
}   
// Method call should fail: white space not allowed  
try  
{  
 number = Rational.Parse(" 68054 ", NumberStyles.AllowLeadingSign);  
 Console.WriteLine(number);  
}  
catch (FormatException e)

{  
 Console.WriteLine(e.Message);  
}   
//  
// The method produces the following output:  
//  
// -68054  
// Input string was not in a correct format.  
// Input string was not in a correct format.  
// Input string was not in a correct format.

## 注釈

styleパラメーターは空白、符号、桁区切り記号、小数点記号など使用することのできる文字を指定することができます。valueパラメーターは、次の形式のうちstyleパラメーターで許可された要素を数字列に含めることができます。

[ws][$][sign][digits,]digits[.fractional\_digits][E[sign]exponential\_digits][ws]

角カッコ（[および]）内の要素は省略可能です。それぞれの要素は次の表のとおりです。

|  |  |
| --- | --- |
| 要素 | 説明 |
| ws | 空白文字。省略可能です。  NumberStyles.AllowLeadingWhiteフラグおよびNumberStyles.AllowTrailingWhiteフラグで使用可能かが決まります。 |
| $ | 通貨記号。有効な文字はカレントカルチャーのNumberFormatInfo.CurrencyNegativePatternとNumberFormatInfo.CurrencyPositivePatternプロパティの値で決まります。  NumberStyles.AllowCurrencySymbolフラグが有効な時に使用可能になります。 |
| sign | 符号。有効な文字はカレントカルチャーのNumberFormatInfo.NegativeSignとNumberFormatInfo.PositiveSignプロパティによって決まります。 |
| digits fractional\_digits exponential\_digits | 数字列。0から9及び小数点で構成している必要があります。fractional\_digits以外では先頭の0は無視します。 |
| , | 数字の桁区切りです。有効な文字はカレントカルチャーのCurrencyGroupSeparatorとNumberGroupSeparatorとPercentGroupSeparatorプロパティで決まります。  NumberStyles.AllowThousandsフラグが有効な時に使用可能になります。 |

|  |  |
| --- | --- |
| . | 小数点記号です。有効な文字はカレントカルチャーのCurrencyDecimalSeparator、NumberDecimalSeparator、PercentDecimalSeparatorプロパティで決まります。  NumberStyles.AllowDecimalPointフラグが有効な時に使用可能になります。 |
| E | “e”または”E”文字は、指数表記で表されている事を示します。NumberStyles.AllowExponentフラグが有効な時使用可能になります。 |

数字のみを含む文字列（NumberStyle.Noneが対応）は常に正常に解析できます。他のNumberStyleメンバーは多くの要素を許可しますが、valueパラメーターにはその全ての要素を含んでいる必要はありません。

|  |  |
| --- | --- |
| None | 数字のみです。 |
| AllowDecimalPoint | 整数部、小数点 (.) と桁の小数部を許容します。 |
| AllowExponent | "e"または"E"文字と共に、指数部を許容します。 |
| AllowLeadingWhite | valueの先頭に空白がある事を許容します。 |
| AllowTrailingWhite | valueの末尾に空白がある事を許容します。 |
| AllowLeadingSign | valueの先頭に符号がある事を許容します。 |
| AllowTrailingSign | valueの末尾に符号がある事を許容します。 |
| AllowParentheses | 負数をカッコで囲って表記する事を許容します。 |
| AllowThousands | 整数部を桁区切りする事を許容します。 |
| AllowCurrencySymbol | 通貨記号を使用する事を許容します。 |
| Currency | すべての要素を許容します。 ただし、 valueプロパティを 16 進数または指数表記で表す事はできません。 |
| Float | valueの先頭、末尾の空白、value先頭の符号、および小数点、指数表記を許容します。 |
| Number | valueの先頭、末尾の空白、value先頭、末尾の符号、桁区切り記号、および小数点といった10進数の全ての要素を許容します。 |
| Any | すべての要素を許容します。 ただし、valueパラメーターを16 進数で表記する事はできません。 |

# Parse(String,IFormatProvider)

指定されたカルチャ固有の書式で表現された文字列形式の数値を、それと等価の Rational に変換します。

public static WS.Theia.ExtremelyPrecise.Rational Parse(String value,IFormatProvider provider);

## パラメーター

value　String  
変換する数値を含んだ文字列。

provider　 IFormatProvider

value に関するカルチャ固有の書式情報を提供するオブジェクト。

## 戻り値

Rational

value パラメーターで指定されている数値と等価の値。

## 例外

ArgumentNullException  
valueはnullです。

FormatException  
valueが正しい形式ではありません。

## 例

次の例ではチルダ（~）を負の符号として定義する方法を2つ提示します。1つ目の例はIFormatProviderインタフェースのGetFormatメソッドを実装し、NumberFormatInfoオブジェクトを返却するクラスを作成する方法です。

まずNumberFormatInfoオブジェクトを返却するクラスを定義します。

public class RationalFormatProvider : IFormatProvider  
{  
 public object GetFormat(Type formatType)   
 {  
 if (formatType == typeof(NumberFormatInfo))   
 {  
 NumberFormatInfo numberFormat = new NumberFormatInfo();  
 numberFormat.NegativeSign = "~";  
 return numberFormat;  
 }  
 else  
 {  
 return null;  
 }   
 }  
}

次にNumberFormatInfoオブジェクトを提供するクラスをインスタンス化して使用します。

Rational number = Rational.Parse("~6354129876", new RationalFormatProvider());  
// Display value using same formatting information  
Console.WriteLine(number.ToString(new RationalFormatProvider()));  
// Display value using formatting of current culture  
Console.WriteLine(number);

2つ目の方法はNumberFormatInfoオブジェクトの値を書き換えてproviderパラメーターに渡す方法です。

NumberFormatInfo fmt = new NumberFormatInfo();  
fmt.NegativeSign = "~";  
  
Rational number = Rational.Parse("~6354129876", fmt);  
// Display value using same formatting information  
Console.WriteLine(number.ToString(fmt));  
// Display value using formatting of current culture  
Console.WriteLine(number);

## 注釈

valueパラメーターは、次の形式で表された数字文字列でなければなりません。

[ws][sign]digits[ws]

角カッコ（[および]）内の要素は省略可能です。それぞれの要素は次の表のとおりです。

|  |  |
| --- | --- |
| 要素 | 説明 |
| ws | 空白文字。省略可能です。 |
| sign | 符号。省略可能です。有効な文字はカレントカルチャーのNumberFormatInfo.NegativeSignとNumberFormatInfo.PositiveSignプロパティによって決まります。 |
| digits | 数字列。0から9及び小数点で構成している必要があります。先頭の0は無視します。有効な小数点はカレントカルチャーのNumberFormatInfo.NumberDecimalSeparatorプロパティによって決まります。 |

providerパラメーターは、IFormatProviderインタフェースのGetFormatメソッドが実装されたオブジェクトを設定することができます。GetFormatメソッドの戻り値はNumberFormatInfoオブジェクトです。Parse(String,IFormatProvider)メソッドには主に3種類方法でproviderを渡すことができます。

* CultureInfoが提供する書式を表すオブジェクト。（GetFormatメソッドがそのカルチャに合わせたNumberFormatInfoオブジェクトを返します。）
* 直接インスタンス化したNumberFormatInfoオブジェクト。（GetFormatがNumberFormatInfoオブジェクト自信を返します）
* IFormatProviderを実装したカスタムオブジェクト。（そのオブジェクトのGetFormatメソッドがNumberFormatInfoオブジェクトをインスタンス化して返します。）

# Parse(String,NumberStyles,IFormatProvider)

指定したスタイルおよびカルチャ固有の書式の数値の文字列形式を、それと等価の Rational に変換します。

public static WS.Theia.ExtremelyPrecise.Rational Parse(String value,NumberStyles style,IFormatProvider provider);

## パラメーター

value　String  
変換する数値を含んだ文字列。

style　 NumberStyles

value に許可されている書式を指定する列挙値のビットごとの組み合わせ。

provider　 IFormatProvider  
value に関するカルチャ固有の書式情報を提供するオブジェクト。

## 戻り値

Rational

value パラメーターで指定されている数値と等価の値。

## 例外

ArgumentException

styleがNumberStyle値ではない、または、seyleにAllowHexSpecifier または HexNumber フラグが含まれます。

ArgumentNullException  
valueはnullです。

FormatException  
valueが正しい形式ではありません。

## 例

次の例ではstyleとproviderパラメーターの様々な組み合わせでParse(String,NumberStyle, IFormatProvider)を呼び出しています。

// Call parse with default values of style and provider

Console.WriteLine(Rational.Parse(" -300 ",   
 NumberStyles.Integer, CultureInfo.CurrentCulture));  
// Call parse with default values of style and provider supporting tilde as negative sign  
Console.WriteLine(Rational.Parse(" ~300 ",   
 NumberStyles.Integer, new RationalFormatProvider()));  
// Call parse with only AllowLeadingWhite and AllowTrailingWhite  
// Exception thrown because of presence of negative sign  
try  
{  
 Console.WriteLine(Rational.Parse(" ~300 ",   
 NumberStyles.AllowLeadingWhite | NumberStyles.AllowTrailingWhite,   
 new RationalFormatProvider()));  
}  
catch (FormatException e)  
{  
 Console.WriteLine("{0}: \n {1}", e.GetType().Name, e.Message);  
}   
// Call parse with only AllowHexSpecifier  
// Exception thrown because of presence of negative sign  
try  
{  
 Console.WriteLine(Rational.Parse("-3af", NumberStyles.AllowHexSpecifier,   
 new RationalFormatProvider()));  
}  
catch (FormatException e)  
{  
 Console.WriteLine("{0}: \n {1}", e.GetType().Name, e.Message);  
}   
// Call parse with only NumberStyles.None  
// Exception thrown because of presence of white space and sign  
try  
{  
 Console.WriteLine(Rational.Parse(" -300 ", NumberStyles.None,   
 new RationalFormatProvider()));  
}  
catch (FormatException e)  
{  
 Console.WriteLine("{0}: \n {1}", e.GetType().Name, e.Message);  
}  
// The example displays the followingoutput:  
// -300  
// -300  
// FormatException:  
// The value could not be parsed.  
// FormatException:  
// The value could not be parsed.  
// FormatException:  
// The value could not be parsed.

Parse(String,NumberStyle, IFormatProvider)メソッドを呼び出す際に使っているRationalFormatProviderクラスは、負の符号としてチルダ（~）を定義しています。

public class RationalFormatProvider : IFormatProvider  
{  
 public object GetFormat(Type formatType)   
 {  
 if (formatType == typeof(NumberFormatInfo))   
 {  
 NumberFormatInfo numberFormat = new NumberFormatInfo();  
 numberFormat.NegativeSign = "~";  
 return numberFormat;  
 }  
 else  
 {  
 return null;  
 }   
 }  
}

## 注釈

styleパラメーターは空白、符号、桁区切り記号、小数点記号など使用することのできる文字を指定することができます。valueパラメーターは、次の形式のうちstyleパラメーターで許可された要素を数字列に含めることができます。

[ws][$][sign][digits,]digits[.fractional\_digits][E[sign]exponential\_digits][ws]

角カッコ（[および]）内の要素は省略可能です。それぞれの要素は次の表のとおりです。

|  |  |
| --- | --- |
| 要素 | 説明 |
| ws | 空白文字。省略可能です。  NumberStyles.AllowLeadingWhiteフラグおよびNumberStyles.AllowTrailingWhiteフラグで使用可能かが決まります。 |

|  |  |
| --- | --- |
| $ | 通貨記号。有効な文字はカレントカルチャーのNumberFormatInfo.CurrencyNegativePatternとNumberFormatInfo.CurrencyPositivePatternプロパティの値で決まります。  NumberStyles.AllowCurrencySymbolフラグが有効な時に使用可能になります。 |
| sign | 符号。有効な文字はカレントカルチャーのNumberFormatInfo.NegativeSignとNumberFormatInfo.PositiveSignプロパティによって決まります。 |
| digits fractional\_digits exponential\_digits | 数字列。0から9及び小数点で構成している必要があります。fractional\_digits以外では先頭の0は無視します。 |
| , | 数字の桁区切りです。有効な文字はカレントカルチャーのCurrencyGroupSeparatorとNumberGroupSeparatorとPercentGroupSeparatorプロパティで決まります。  NumberStyles.AllowThousandsフラグが有効な時に使用可能になります。 |
| . | 小数点記号です。有効な文字はカレントカルチャーのCurrencyDecimalSeparator、NumberDecimalSeparator、PercentDecimalSeparatorプロパティで決まります。  NumberStyles.AllowDecimalPointフラグが有効な時に使用可能になります。 |
| E | “e”または”E”文字は、指数表記で表されている事を示します。NumberStyles.AllowExponentフラグが有効な時使用可能になります。 |

数字のみを含む文字列（NumberStyle.Noneが対応）は常に正常に解析できます。他のNumberStyleメンバーは多くの要素を許可しますが、valueパラメーターにはその全ての要素を含んでいる必要はありません。

|  |  |
| --- | --- |
| None | 数字のみです。 |
| AllowDecimalPoint | 整数部、小数点 (.) と桁の小数部を許容します。 |
| AllowExponent | "e"または"E"文字と共に、指数部を許容します。 |
| AllowLeadingWhite | valueの先頭に空白がある事を許容します。 |
| AllowTrailingWhite | valueの末尾に空白がある事を許容します。 |
| AllowLeadingSign | valueの先頭に符号がある事を許容します。 |
| AllowTrailingSign | valueの末尾に符号がある事を許容します。 |
| AllowParentheses | 負数をカッコで囲って表記する事を許容します。 |
| AllowThousands | 整数部を桁区切りする事を許容します。 |
| AllowCurrencySymbol | 通貨記号を使用する事を許容します。 |
| Currency | すべての要素を許容します。 ただし、 valueプロパティを 16 進数または指数表記で表す事はできません。 |
| Float | valueの先頭、末尾の空白、value先頭の符号、および小数点、指数表記を許容します。 |
| Number | valueの先頭、末尾の空白、value先頭、末尾の符号、桁区切り記号、および小数点といった10進数の全ての要素を許容します。 |
| Any | すべての要素を許容します。 ただし、valueパラメーターを16 進数で表記する事はできません。 |

# 適用対象

### .NET Core

2.0

### .NET Framework

4.6.1

### .NET Standard

2.0

### UWP

10.0.16299

### Xamarin.Android

8.0

### Xamarin.iOS

10.14

### Xamarin.Mac

3.8