Rational.TryParse Method

名前空間: WS.Theia.ExtremelyPrecise
アセンブリ: ExtremelyPrecise.dll
数値の文字列形式を対応する Rational 表現に変換できるかどうかを試行し、変換に成功したかどうかを示す値を返します。

オーバーロード

TryParse(String)	数値の文字列形式を対応する Rational 表現に変換できるかどうかを試行し、変換に成功したかどうかを示す値を返します。
TryParse(String,NumberStyles,IFormatProvider)	数値の文字列形式を対応する Rational 表現に変換できるかどうかを試行し、変換に成功したかどうかを示す値を返します。

TryParse(String)

数値の文字列形式を対応する Rational 表現に変換できるかどうかを試行し、変換に成功したかどうかを示す値を返します。

---

public static (bool status, WS.Theia.ExtremelyPrecise.Rational result)
TryParse(String value)

パラメーター
value　String
数値の文字列形式。

戻り値
status　Boolean
value が正常に変換できた場合は true。それ以外の場合は false。
result　Rational
このメソッドから制御が戻るときに、value と等価の Rational が格納されます。value パラメーターが null の場合、または正しい形式ではない場合、変換は失敗します。変換に失敗した場合このパラメーターは初期化せずに渡されます。

例

次の例では TryParse(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。

```
Rational number1, number2;
bool succeeded1, succeeded2;
(succeeded1,number1) = Rational.TryParse("-12347534159895123");
(succeeded2,number2) = Rational.TryParse("987654321357159852");
if (succeeded1 && succeeded2)
{
    number1 *= 3;
    number2 *= 2;
    switch (Rational.Compare(number1, number2))
    {
        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;
    }
}
else
{
    if (! succeeded1)
        Console.WriteLine("Unable to initialize the first Rational value.");
    if (! succeeded2)
        Console.WriteLine("Unable to initialize the second Rational value.");
}
// The example displays the following output:
//         1975308642714319704 is greater than -37042602479685369.
```

注釈

TryParse(String)メソッドは Parse(String)メソッドと異なり、変換に失敗しても例外を発生しません。FormatException が発生する状況では、戻り値 status が false になり戻り値 result が無効な値になります。

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。

TryParse(String,NumberStyles,IFormatProvider)

数値の文字列形式を対応する Rational 表現に変換できるかどうかを試行し、変換に成功したかどうかを示す値を返します。

---

public static (bool status, WS.Theia.ExtremelyPrecise.Rational result)
TryParse(String value,NumberStyles style,IFormatProvider provider);

パラメーター
value　String
数値の文字列形式。
style　NumberStyles
value で存在する可能性を持つスタイル要素を示す、列挙値のビットごとの組み合わせ。通常指定する値は Integer です。
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
status　Boolean
value が正常に変換できた場合は true。それ以外の場合は false。
result　Rational
このメソッドから制御が戻るときに、value と等価の Rational が格納されます。value パラメーターが null の場合、または正しい形式ではない場合、変換は失敗します。変換に失敗した場合このパラメーターは初期化せずに渡されます。

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
TryParse(String,NumberStyle, IFormatProvider)を呼び出しています。

```
string numericString;
Rational number = Rational.Zero;
bool status=false;

// Call TryParse with default values of style and provider.
numericString = "    -300     ";
(status,number)=Rational.TryParse(numericString, NumberStyles.Integer,
                                  null);
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                      numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                      numericString);

// Call TryParse with the default value of style and
// a provider supporting the tilde as negative sign.
numericString = "    -300     ";
(status,number)=Rational.TryParse(numericString, NumberStyles.Integer,
                                  new RationalFormatProvider());
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                      numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                      numericString);

// Call TryParse with only AllowLeadingWhite and AllowTrailingWhite.
// Method returns false because of presence of negative sign.
numericString = "    -500     ";
(status,number)=Rational.TryParse(numericString,
```

```
                                NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite,
                                new RationalFormatProvider());
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                                numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                numericString);

// Call TryParse with AllowHexSpecifier and a hex value.
numericString = "F14237FFAAC086455192";
(status,number)=Rational.TryParse(numericString,
                                NumberStyles.AllowHexSpecifier, null);
if (status)
    Console.WriteLine("'{0}' was converted to {1} (0x{1:x}).",
                                numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                numericString);

// Call TryParse with AllowHexSpecifier and a negative hex value.
// Conversion fails because of presence of negative sign.
numericString = "-3af";
(status,number)=Rational.TryParse(numericString,
NumberStyles.AllowHexSpecifier,new RationalFormatProvider());
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                                numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                numericString);

// Call TryParse with only NumberStyles.None.
// Conversion fails because of presence of white space and sign.
numericString = " -300 ";
```

```
(status,number)=Rational.TryParse(numericString, NumberStyles.None,
                                  new RationalFormatProvider());
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                                  numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                  numericString);

// Call TryParse with NumberStyles.Any and a provider for the fr-FR culture.
// Conversion fails because the string is formatted for the en-US culture.
numericString = "9,031,425,666,123,546.00";
(status,number)=Rational.TryParse(numericString, NumberStyles.Any,
                                  new CultureInfo("fr-FR"));
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                                  numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                  numericString);

// Call TryParse with NumberStyles.Any and a provider for the fr-FR culture.
// Conversion succeeds because the string is properly formatted
// For the fr-FR culture.
numericString = "9 031 425 666 123 546,00";
(status,number)=Rational.TryParse(numericString, NumberStyles.Any,
                                  new CultureInfo("fr-FR"));
if (status)
    Console.WriteLine("'{0}' was converted to {1}.",
                                  numericString, number);
else
    Console.WriteLine("Conversion of '{0}' to a Rational failed.",
                                  numericString);
// The example displays the following output:
//       '    -300     ' was converted to -300.
//       Conversion of '    -300     ' to a Rational failed.
```

```
//       Conversion of '    -500      ' to a Rational failed.
//       'F14237FFAAC086455192' was converted to -
69613977002644837412462 (0xf14237ffaac086455192).
//       Conversion of '-3af' to a Rational failed.
//       Conversion of ' -300 ' to a Rational failed.
//       Conversion of '9,031,425,666,123,546.00' to a Rational failed.
//       '9 031 425 666 123 546,00' was converted to 9031425666123546.
```

TryParse(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

TryParse(String,NumberStyles,IFormatProvider)メソッドは Parse(String,NumberStyles,IFormatProvider)メソッドと異なり、変換に失敗しても例外を発生しません。FormatException が発生する状況では、戻り値 status が false になり戻り値 result が無効な値になります。

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