Rational Constructors

名前空間: WS.Theia.ExtremelyPrecise
アセンブリ: ExtremelyPrecise.dll

オーバーロード

Rational(Boolean)	Boolean 値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Boolean,Byte[],Byte[])	バイト配列の値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Decimal)	Decimal 値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Double)	倍精度浮動小数点値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Int32)	32 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Int64)	64 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(Single)	単精度浮動小数点値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(UInt32)	32 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。
Rational(UInt64)	64 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。

Rational(Boolean)

Boolean 値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(Boolean value);
```

パラメーター

value　Boolean

Boolean 値(true=One、false=Zero と認識します)。

例

次の例では Boolean 値を使用して Rational オブジェクトを初期化しています。True を One、False を Zero として扱います。

```
Rational number = new Rational (true);
Console.WriteLine("The value of number is {0}.", number);
// The example displays the following output:
//       The value of number is 1.
```

Rational(Boolean,Byte[],Byte[])

> ⚠重要
> この API は CLS 準拠ではありません。

バイト配列の値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(bool sign,byte[] numerator,byte[] denominator) {
```

パラメーター
sign　Boolean
符号を示す値（false＝プラス、true=マイナス）。
numerator　Byte[]
分子を表すリトルエンディアン順に格納されたバイト値の配列。
denominator　Byte[]
分母を表すリトルエンディアン順に格納されたバイト値の配列。

例外
numerator、又は、denominator は null です。

例

次の例では{1}の 1 要素からなる Byte 配列を分母、{5,4,3,2,1}の 5 要素からなる Byte 配列を分子の正の値として、Rational オブジェクトを初期化しています。後に Rational オブジェクトの値をコンソールに 10 進数で表示しています。このオーバーロードで作成した Rational オブジェクトの値は 4328719365 になります。

```
bool sign=false;
byte[] numerator = { 5, 4, 3, 2, 1 };
byte[] denominator = { 1 };
Rational number = new Rational(sign, numerator, denominator);
Console.WriteLine("The value of number is {0}.", number);
// The example displays the following output:
//       The value of number is 4328719365.
```

次の例では負の値で Rational オブジェクトを初期化しています。正負の違いは符号フラグとして示され分母、及び、分子を示す Byte 配列は全く同じである事に注意してください。

```
bool sign=true;
byte[] numerator = { 5, 4, 3, 2, 1 };
byte[] denominator = { 1 };
Rational number = new Rational(sign, numerator, denominator);
Console.WriteLine("The value of number is {0}.", number);
// The example displays the following output:
//       The value of number is -4328719365.
```

注釈

numerator パラメーター、denominator パラメーターは最上位バイトに数値の最下位を並べるリトルエンディアン順でなければなりません。たとえば、数値 1,000,000,000,000 は、次の表に示すように表されます。

数値の 16 進数文字列	E8D4A51000
バイト配列（前方のインデックスが最も低い）	00 10 A5 D4 E8 00

BitConverter.GetBytes、BigInteger.ToByteArray 等、殆どの Byte 配列に変換するメソッドは、リトルエンディアン順で値を格納して Byte 配列を返します。ただし、numerator パラメーター、denominator パラメーター共に、最上位ビットと常に符号無し整数として扱います。その結果、負数が正数として解釈される場合があります。通常、numerator パラメーター、denominator パラメーターの Byte 配列は次の方法で生成されます。

- Rational.ToByteArray メソッドの呼び出しにより生成する。このメソッドでは符号フラグ、分子配列、分母配列、無限大フラグのタプル型として生成し、分子配列、分母配列共に常に正の値を持っており負数が正数として解釈する事はありません。
- BitConverter.GetBytes メソッドにパラメーターとして符号付き整数を渡して生成する。符号付整数では最上位ビットを符号ビットとして負数を 2 の補数で表現しますが、numerator パラメーター、denominator パラメーターでは正数として解釈します。Rational(Int32)、Rational(Int64)コンストラクターを使用する事で回避可能です。
- BitConverter.GetBytes メソッドにパラメーターとして符号なし整数を渡して生成する。符号なし整数では負数が存在しない為、numerator パラメーター、denominator パラメーターは正しく解釈することができます。
- 動的、又は、静的に生成した Byte 配列を生成する。この場合、正しく解釈されるには numerator パラメーター、denominator パラメーター共に絶対値を設定し、符号は sign パラメーターに設定しなければなりません。
- BigInteger.ToByteArray メソッドの呼び出しにより生成する。BigInteger では最上位ビットを符号ビットとして負数を 2 の補数で表現しますが、numerator パラメーター、denominator パラメーターでは正数として解釈します。符号は sign パラメーターに設定して、絶対値を取得してから ToByteArray メソッドを呼び出して Byte 配列を生成してください。

numerator パラメーターが空配列の場合、Rational.Zero として初期化します。denominator パラメーターが空配列、又は、0 のみで構成される配列の場合、Rational.NaN として初期化します。numerator パラメーター、denominator パラメーターが null の場合は、ArgumentNullException が発生します。

> ⚠注意
> denominator パラメーターが空配列、又は、0 のみで構成される配列の場合、Rational.NaN に初期化する仕様については、将来的に変更する可能性があります。NaN 値が必要な場合は、Rational.NaN プロパティを使用してください。

また、ToByteArray()も参照してください。

Rational(Decimal)

Decimal 値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(decimal value);
```

パラメーター

value　Decimal

10 進数値。

例

次の例では、Rational オブジェクトを Rational(Decimal)コンストラクターを初期化しています。配列に定義した Decimal 値をそれぞれ Rational(Decimal)コンストラクターに渡しています。

```
decimal[] decimalValues = { -1790.533m, -15.1514m, 18903.79m,
9180098.003m };
foreach (decimal decimalValue in decimalValues)
{
    Rational number = new Rational(decimalValue);
    Console.WriteLine("Instantiated Rational value {0} from the Decimal value
{1}.",
                      number, decimalValue);
}
// The example displays the following output:
//       Instantiated Rational value -1790.533 from the Decimal value -1790.533.
//       Instantiated Rational value -15.1514. from the Decimal value -15.1514.
//       Instantiated Rational value 18903.79 from the Decimal value 18903.79.
//       Instantiated Rational value 9180098.003 from the Decimal value
9180098.003.
```

注釈

このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に Decimal 値を割り当てる事は等価です。

Rational(Double)

倍精度浮動小数点値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(double value);
```

パラメーター

value Double

倍精度浮動小数点数値。

例

Double 値を使用して、Rational 構造体の新しいインスタンスを初期化します。Double 型には大きな値が割り当てられている為、足し算を行った際に、情報落ちが起こり、変更が反映されていませんが、Rational 型では変更が反映されています。

```
// Create a Rational from a large double value.
double doubleValue = -6e20;
Rational rationalValue = new Rational (doubleValue);
Console.WriteLine("Original Double value: {0:N0}", doubleValue);
Console.WriteLine("Original Rational value: {0:N0}", rationalValue);
// Increment and then display both values.
doubleValue++;
rationalValue += Rational.One;
Console.WriteLine("Incremented Double value: {0:N0}", doubleValue);
Console.WriteLine("Incremented Rational value: {0:N0}", rationalValue);
// The example displays the following output:
//       Original Double value: -600,000,000,000,000,000,000
//       Original Rational value: -600,000,000,000,000,000,000
//       Incremented Double value: -600,000,000,000,000,000,000
//       Incremented Rational value: -599,999,999,999,999,999,999
```

注釈

Rational 型には有効桁数の上限が事実上無限の為、Double 型と違いデータの損失を起こさずに演算することができます。このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に Double 値を割り当てる事は等価です。

Rational(Int32)

32 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(int value);
```

パラメーター

value　Int32

32 ビット符号付き整数値。

例

32 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。32 ビット符号付き整数から Rational 変数に代入し暗黙的な変換をした結果と比較しています。その結果は明示的なインスタンス化と暗黙的な変換の結果は等価です。

```
int[] integers = { Int32.MinValue, -10534, -189, 0, 17, 113439,
                   Int32.MaxValue };
Rational constructed, assigned;

foreach (int number in integers)
{
    constructed = new Rational(number);
    assigned = number;
    Console.WriteLine("{0} = {1}: {2}", constructed, assigned,
                      constructed.Equals(assigned));
}
// The example displays the following output:
//       -2147483648 = -2147483648: True
//       -10534 = -10534: True
//       -189 = -189: True
//       0 = 0: True
//       17 = 17: True
//       113439 = 113439: True
//       2147483647 = 2147483647: True
```

注釈

このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に Int32 値を割り当てる事は等価です。Byte、Int16、SByte、UInt16 のコンストラクターはありません。ただし、Int32 型の 8 ビット、16 ビットの符号付き、符号無し整数の 32 ビット符号付整数への暗黙的な変換を利用して、このコンストラクターが呼び出されます。その為このコンストラクターに渡されるデータ型は Byte、Int16、SByte、UInt16、Int32 のいずれかです。

Rational(Int64)

64 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(long value);
```

パラメーター

value　Int64

64 ビット符号付き整数値。

例

64 ビット符号付き整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。64 ビット符号付き整数から Rational 変数に代入し暗黙的な変換をした結果と比較しています。その結果は明示的なインスタンス化と暗黙的な変換の結果は等価です。

```
long[] longs = { Int64.MinValue, -10534, -189, 0, 17, 113439,
                 Int64.MaxValue };
Rational constructed, assigned;

foreach (long number in longs)
{
    constructed = new Rational(number);
    assigned = number;
    Console.WriteLine("{0} = {1}: {2}", constructed, assigned,
                      constructed.Equals(assigned));
}
// The example displays the following output:
//       - 9223372036854775808 = - 9223372036854775808: True
//       -10534 = -10534: True
//       -189 = -189: True
//       0 = 0: True
//       17 = 17: True
//       113439 = 113439: True
//       9223372036854775807 = 9223372036854775807: True
```

注釈

このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に Int64 値を割り当てる事は等価です。

Rational(Single)

単精度浮動小数点値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(float value);
```

パラメーター

value　Single

単精度浮動小数点数値。

例

Single 値を使用して、Rational 構造体の新しいインスタンスを初期化します。Single 型には大きな値が割り当てられている為、足し算を行った際に、情報落ちが起こり、変更が反映されていませんが、Rational 型では変更が反映されています。

```
// Create a BigInteger from a large negative Single value
float negativeSingle = Single.MinValue;
Rational negativeNumber = new Rational (negativeSingle);

Console.WriteLine(negativeSingle.ToString("N0"));
Console.WriteLine(negativeNumber.ToString("N0"));

negativeSingle++;
negativeNumber++;

Console.WriteLine(negativeSingle.ToString("N0"));
Console.WriteLine(negativeNumber.ToString("N0"));
// The example displays the following output:
//       -340,282,300,000,000,000,000,000,000,000,000,000,000
//       -340,282,346,638,528,859,811,704,183,484,516,925,440
//       -340,282,300,000,000,000,000,000,000,000,000,000,000
//       -340,282,346,638,528,859,811,704,183,484,516,925,439
```

注釈

Rational 型には有効桁数の上限が事実上無限の為、Single 型と違いデータの損失を起こさずに演算することができます。このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に Single 値を割り当てる事は等価です。

Rational(UInt32)

> ⚠重要
> この API は CLS 準拠ではありません。
> CLS 準拠の代替：WS.Theia.ExtremelyPrecise.Rational.Rational(Int64)

32 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(uint value);
```

パラメーター
value　UInt32
32 ビットの符号なし整数値。

例

32 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。32 ビット符号なし整数から Rational 変数に代入し暗黙的な変換をした結果と比較しています。その結果は明示的なインスタンス化と暗黙的な変換の結果は等価です。

```
uint[] unsignedValues = { 0, 16704, 199365, UInt32.MaxValue };
foreach (uint unsignedValue in unsignedValues)
{
    Rational constructedNumber = new Rational (unsignedValue);
    Rational assignedNumber = unsignedValue;
    if (constructedNumber.Equals(assignedNumber))
        Console.WriteLine("Both methods create a Rational whose value is
{0:N0}.",
                          constructedNumber);
    else
        Console.WriteLine("{0:N0} ≠ {1:N0}", constructedNumber,
assignedNumber);

}
// The example displays the following output:
//       Both methods create a Rational whose value is 0.
//       Both methods create a Rational r whose value is 16,704.
//       Both methods create a Rational whose value is 199,365.
//       Both methods create a Rational whose value is 4,294,967,295.
```

注釈

このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に UInt32 値を割り当てる事は等価です。

Rational(UInt64)

> ⚠重要
> この API は CLS 準拠ではありません。
> CLS 準拠の代替：WS.Theia.ExtremelyPrecise.Rational.Rational(Double)

64 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。

```
public Rational(uint value);
```

パラメーター
value　UInt64
64 ビットの符号なし整数値。

例

64 ビット符号なし整数値を使用して、Rational 構造体の新しいインスタンスを初期化します。次の例では UInt64.MaxValue を設定しています。

```
ulong unsignedValue = UInt64.MaxValue;
Rational number = new Rational (unsignedValue);
Console.WriteLine(number.ToString("N0"));
// The example displays the following output:
//           18,446,744,073,709,551,615
```

注釈

このコンストラクターを明示的に呼び出して割り当てる事と、Rational 変数に UInt64 値を割り当てる事は等価です。

適用対象

.NET Core

2.0

.NET Framework

4.6.1 2

.NET Standard

2.0

UWP

10.0.16299

Xamarin.Android

8.0

Xamarin.iOS

10.14

Xamarin.Mac

3.8