# EsbDecimals for .NET

Free Assembly that has classes that supply Constants and Mathematical Routines for the Decimal Type supplied by the .NET Framework.

Made by ESB Consultancy with Doc-O-Matic.

# Table of Contents

Symbol Reference 1

EsbDecimals Namespace 1

CompareOps Class 1

CompareOps Fields 3

CompareOps.DecimalPrecisionDef Field 3

CompareOps.DoublePrecisionDef Field 3

CompareOps.FloatPrecisionDef Field 4

CompareOps Methods 4

CompareValue Method 4

GetPrecision Method 8

GreaterValue Method 11

LesserValue Method 15

SameValue Method 19

ValueIsNegative Method 22

ValueIsPositive Method 26

ValueIsZero Method 29

DecimalGlobals Class 33

DecimalGlobals Fields 35

DecimalGlobals.Cbrt10 Field 36

DecimalGlobals.Cbrt100 Field 36

DecimalGlobals.Cbrt2 Field 36

DecimalGlobals.Cbrt3 Field 36

DecimalGlobals.CbrtPi Field 36

DecimalGlobals.FivePiOver2 Field 37

DecimalGlobals.FourPiOver3 Field 37

DecimalGlobals.Gamma Field 37

DecimalGlobals.GoldenRatio Field 37

DecimalGlobals.InvCbrtPi Field 38

DecimalGlobals.InvPi Field 38

DecimalGlobals.InvSqrt2 Field 38

DecimalGlobals.InvSqrt3 Field 38

DecimalGlobals.InvSqrt5 Field 38

DecimalGlobals.InvSqrtPi Field 39

DecimalGlobals.Ln10 Field 39

DecimalGlobals.Ln2 Field 39

DecimalGlobals.LnOfPi Field 39

DecimalGlobals.LnSqrt2Pi Field 40

DecimalGlobals.Log10Base2 Field 40
DecimalGlobals.Log2 Field 40
DecimalGlobals.Log3 Field 40
DecimalGlobals.LogNaturalConstant Field 40
DecimalGlobals.LogPi Field 41
DecimalGlobals.NaturalConstant Field 41
DecimalGlobals.NaturalConstant2Pi Field 41
DecimalGlobals.NaturalConstant2PiOver2 Field 41
DecimalGlobals.NaturalConstant2PiOver4 Field 42
DecimalGlobals.OneDegree Field 42
DecimalGlobals.OneMinute Field 42
DecimalGlobals.OneRadian Field 42
DecimalGlobals.OneSecond Field 42
DecimalGlobals.Pi Field 43
DecimalGlobals.Pi2E Field 43
DecimalGlobals.PiOver12 Field 43
DecimalGlobals.PiOver2 Field 43
DecimalGlobals.PiOver3 Field 44
DecimalGlobals.PiOver4 Field 44
DecimalGlobals.PiOver5 Field 44
DecimalGlobals.PiOver6 Field 44
DecimalGlobals.SqrNaturalConstant Field 44
DecimalGlobals.SqrPi Field 45
DecimalGlobals.Sqrt10 Field 45
DecimalGlobals.Sqrt2 Field 45
DecimalGlobals.Sqrt3 Field 45
DecimalGlobals.Sqrt5 Field 46
DecimalGlobals.SqrtPi Field 46
DecimalGlobals.ThreePi Field 46
DecimalGlobals.ThreePiOver2 Field 46
DecimalGlobals.ThreePiOver4 Field 46
DecimalGlobals.TwoPi Field 47
DecimalGlobals.TwoToPower63 Field 47
DecimalGlobals.Version Field 47
DecimalGlobals.VersionDate Field 47
DecimalOps Class 48
DecimalOps Methods 51
Acos Method 51
AcosDeg Method 52
AdjustAngleDeg Method 53
AdjustAngleRad Method 56

Asin Method 59
AsinDeg Method 60
Atan Method 61
Atan2 Method 63
Atan2Deg Method 64
AtanDeg Method 66
Ceiling Method 67
Cos Method 68
CosDeg Method 70
Cosec Method 71
CosecDeg Method 73
Cot Method 74
CotDeg Method 76
Deg2Dms Method 78
Deg2Grad Method 79
Deg2Rad Method 80
Dms2Deg Method 82
Exp Method 83
FractionPart Method 85
Grad2Deg Method 86
Grad2Rad Method 87
IntPart Method 88
IntPow Method 89
Log Method 91
Log10 Method 92
Log2 Method 94
NthRoot Method 96
Pow Method 97
Rad2Deg Method 103
Rad2Grad Method 104
Sec Method 105
SecDeg Method 107
SignXY Method 109
Sin Method 111
SinCos Method 112
SinDeg Method 114
Sqrt Method 115
Tan Method 117
TanDeg Method 118
TenPow Method 120
TwoPow Method 121

IntOps Class 122
IntOps Methods 123
Ceiling Method 123
Ceiling64 Method 124
Floor Method 125
Floor64 Method 126
IntPart Method 127
IntPart64 Method 128
Odd Method 129

**Index a**

# 1 Symbol Reference

## 1.1 EsbDecimals Namespace

EsbDecimals - v2.1.0 - 10 June 2011.

Free Assembly that has classes that supply Constants and Mathematical Routines for the Decimal Type supplied by the .NET Framework.

Includes C# Source.

Developed by ESB Consultancy.

**Classes**

| | Name | Description |
|---|---|---|
| S | CompareOps (↗ see page 1) | Math Comparison Routines for EsbDecimals.<br>Equality, inequality, greater than and less than comparisons should not be done with the operators, rather use these methods where the Precision of the comparison can be controlled. |
| S | DecimalGlobals (↗ see page 33) | Global Decimal Constants for EsbDecimals. |
| S | DecimalOps (↗ see page 48) | Decimal Routines that are missing from the Framework including Sqrt (↗ see page 115), etc for EsbDecimals. |
| S | IntOps (↗ see page 122) | Integer Routines for EsbDecimals. |

### 1.1.1 CompareOps Class

**Inheritance Hierarchy**

EsbDecimals.CompareOps

**C++**

```
public: class CompareOps;
```

**C#**

```
public static class CompareOps;
```

**Visual Basic**

```
Public static Class CompareOps
```

**File**

EsbMath-Compare.cs

**Remarks**

Math Comparison Routines for EsbDecimals (↗ see page 1).

Equality, inequality, greater than and less than comparisons should not be done with the operators, rather use these methods where the Precision of the comparison can be controlled.

**Members**

**CompareOps Fields**

| | Name | Description |
|---|---|---|
| ◆ | DecimalPrecisionDef (↗ see page 3) | Default Precision when doing Floating Point Comparisons of type Decimal (1.0e-23m). |
| ◆ | DoublePrecisionDef (↗ see page 3) | Default Precision when doing Floating Point Comparisons of type Double (1.0e-12). |
| ◆ | FloatPrecisionDef (↗ see page 4) | Default Precision when doing Floating Point Comparisons of type Float (1.0e-5f). |

**CompareOps Methods**

| | Name | Description |
|---|---|---|
| ◆ S | CompareValue (↗ see page 4) | Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. DecimalPrecisionDef (↗ see page 3) defines the Precision. If you just want to know whether two values are the same then use SameValue(decimal,decimal) |
| ◆ S | GetPrecision (↗ see page 8) | Returns the Precision Value to use using the Default Precision. See DecimalPrecisionDef (↗ see page 3). |
| ◆ S | GreaterValue (↗ see page 11) | Returns True if Values are not within a "small" value of each other, and the First Value is greater. DecimalPrecisionDef (↗ see page 3) defines the Precision. |
| ◆ S | LesserValue (↗ see page 15) | Returns True if Values are not within a "small" value of each other, and the First Value is lesser. DecimalPrecisionDef (↗ see page 3) defines the Precision. |
| ◆ S | SameValue (↗ see page 19) | Returns True if the Values are within a "small" value of each other. If you want more control over the comparison then use CompareValue(decimal,decimal)<br>DecimalPrecisionDef (↗ see page 3) defines the Precision. |
| ◆ S | ValueIsNegative (↗ see page 23) | Returns True if Value is not within a "small" value of 0 and is Negative. DoublePrecisionDef (↗ see page 3) defines the Precision. |
| ◆ S | ValueIsPositive (↗ see page 26) | Returns True if Value is not within a "small" value of 0 and is Positive. DecimalPrecisionDef (↗ see page 3) defines the Precision. |
| ◆ S | ValueIsZero (↗ see page 29) | Returns True if Value is within a "small" value of 0. DecimalPrecisionDef (↗ see page 3) defines the Precision. |

**CompareOps Fields**

| | Name | Description |
|---|---|---|
| ◆ | DecimalPrecisionDef (↗ see page 3) | Default Precision when doing Floating Point Comparisons of type Decimal (1.0e-23m). |
| ◆ | DoublePrecisionDef (↗ see page 3) | Default Precision when doing Floating Point Comparisons of type Double (1.0e-12). |
| ◆ | FloatPrecisionDef (↗ see page 4) | Default Precision when doing Floating Point Comparisons of type Float (1.0e-5f). |

**CompareOps Methods**

| | Name | Description |
|---|---|---|
| ◆S | CompareValue (⧉ see page 4) | Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. DecimalPrecisionDef (⧉ see page 3) defines the Precision. If you just want to know whether two values are the same then use SameValue(decimal,decimal) |
| ◆S | GetPrecision (⧉ see page 8) | Returns the Precision Value to use using the Default Precision. See DecimalPrecisionDef (⧉ see page 3). |
| ◆S | GreaterValue (⧉ see page 11) | Returns True if Values are not within a "small" value of each other, and the First Value is greater. DecimalPrecisionDef (⧉ see page 3) defines the Precision. |
| ◆S | LesserValue (⧉ see page 15) | Returns True if Values are not within a "small" value of each other, and the First Value is lesser. DecimalPrecisionDef (⧉ see page 3) defines the Precision. |
| ◆S | SameValue (⧉ see page 19) | Returns True if the Values are within a "small" value of each other. If you want more control over the comparison then use CompareValue(decimal,decimal)<br>DecimalPrecisionDef (⧉ see page 3) defines the Precision. |
| ◆S | ValueIsNegative (⧉ see page 23) | Returns True if Value is not within a "small" value of 0 and is Negative. DoublePrecisionDef (⧉ see page 3) defines the Precision. |
| ◆S | ValueIsPositive (⧉ see page 26) | Returns True if Value is not within a "small" value of 0 and is Positive. DecimalPrecisionDef (⧉ see page 3) defines the Precision. |
| ◆S | ValueIsZero (⧉ see page 29) | Returns True if Value is within a "small" value of 0. DecimalPrecisionDef (⧉ see page 3) defines the Precision. |

## 1.1.1.1 CompareOps Fields

### 1.1.1.1.1 CompareOps.DecimalPrecisionDef Field

Default Precision when doing Floating Point Comparisons of type Decimal (1.0e-23m).

**C++**

```
public: decimal DecimalPrecisionDef = 1.0e-23m;
```

**C#**

```
public const decimal DecimalPrecisionDef = 1.0e-23m;
```

**Visual Basic**

```
Public Const DecimalPrecisionDef As decimal = 1.0e-23m
```

### 1.1.1.1.2 CompareOps.DoublePrecisionDef Field

Default Precision when doing Floating Point Comparisons of type Double (1.0e-12).

**C++**

```
public: double DoublePrecisionDef = 1.0e-12;
```

**C#**

```
public const double DoublePrecisionDef = 1.0e-12;
```

**Visual Basic**

```
Public Const DoublePrecisionDef As double = 1.0e-12
```

### 1.1.1.1.3 CompareOps.FloatPrecisionDef Field

Default Precision when doing Floating Point Comparisons of type Float (1.0e-5f).

**C++**

```
public: float FloatPrecisionDef = 1.0e-5f;
```

**C#**

```
public const float FloatPrecisionDef = 1.0e-5f;
```

**Visual Basic**

```
Public Const FloatPrecisionDef As float = 1.0e-5f
```

## 1.1.1.2 CompareOps Methods

### 1.1.1.2.1 CompareValue Method

#### 1.1.1.2.1.1 CompareOps.CompareValue Method (decimal, decimal)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. DecimalPrecisionDef (↗ see page 3) defines the Precision. If you just want to know whether two values are the same then use SameValue(decimal,decimal)

**C++**

```
public: int CompareValue(
    decimal X,
    decimal Y
);
```

**C#**

```
public static int CompareValue(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As decimal,
    Y As decimal
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.1.2 CompareOps.CompareValue Method (decimal, decimal, decimal)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. For Values with lots of significant figures, then Precision should be made smaller. If you just want to know whether two values are the same then use SameValue(decimal,decimal,decimal)

**C++**

```
public: int CompareValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**C#**

```
public static int CompareValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As decimal,
    Y As decimal,
    Precision As decimal
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (↗ see page 3). |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (↗ see page 3) for the Precision value.

### 1.1.1.2.1.3 CompareOps.CompareValue Method (double, double)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. DoublePrecisionDef (↗ see page 3) defines the Precision. If you just want to know whether two values are the same then use SameValue(double,double)

**C++**

```
public: int CompareValue(
    double X,
    double Y
);
```

**C#**

```
public static int CompareValue(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As double,
    Y As double
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.1.4 CompareOps.CompareValue Method (double, double, double)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. For Values with lots of significant figures, then Precision should be made smaller. If you just want to know whether two values are the same then use SameValue(double,double,double)

**C++**

```
public: int CompareValue(
    double X,
    double Y,
    double Precision
);
```

**C#**

```
public static int CompareValue(
    double X,
    double Y,
    double Precision
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As double,
    Y As double,
    Precision As double
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (↗ see page 3) for the Precision value.

## 1.1.1.2.1.5 CompareOps.CompareValue Method (float, float)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. FloatPrecisionDef (see page 4) defines the Precision. If you just want to know whether two values are the same then use SameValue(float,float)

**C++**

```
public: int CompareValue(
    float X,
    float Y
);
```

**C#**

```
public static int CompareValue(
    float X,
    float Y
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As float,
    Y As float
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.1.6 CompareOps.CompareValue Method (float, float, float)

Returns 0 if Values are within a "small" value computed from the supplied Precision, 1 if the first value is greater than Y and -1 if the first value is lesser. For Values with lots of significant figures, then Precision should be made smaller. If you just want to know whether two values are the same then use SameValue(float,float,float)

**C++**

```
public: int CompareValue(
    float X,
    float Y,
    float Precision
);
```

**C#**

```
public static int CompareValue(
    float X,
    float Y,
    float Precision
);
```

**Visual Basic**

```
Public static Function CompareValue(
    X As float,
    Y As float,
    Precision As float
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (see page 4). |

**Returns**

0 (Same), 1 (First Value Greater) or -1 (Second Value Greater).

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (see page 4) for the Precision value.

## 1.1.1.2.2 GetPrecision Method

### 1.1.1.2.2.1 CompareOps.GetPrecision Method (decimal, decimal)

Returns the Precision Value to use using the Default Precision. See DecimalPrecisionDef (see page 3).

**C++**

```
public: decimal GetPrecision(
    decimal X,
    decimal Y
);
```

**C#**

```
public static decimal GetPrecision(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function GetPrecision(
    X As decimal,
    Y As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.2.2 CompareOps.GetPrecision Method (decimal, decimal, decimal)

Returns the Precision Value to use using the specified Precision

**C++**

```
public: decimal GetPrecision(
    decimal X,
    decimal Y,
    decimal Precision
);
```

### C#

```
public static decimal GetPrecision(
    decimal X,
    decimal Y,
    decimal Precision
);
```

### Visual Basic

```
Public static Function GetPrecision(
    X As decimal,
    Y As decimal,
    Precision As decimal
) As decimal
```

### Parameters

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |
| decimal Precision | Precision Value to use - should be small and positive like DecimalPrecisionDef (⧉ see page 3). |

### Remarks

There is an overloaded version without the Precision that will use DecimalPrecisionDef (⧉ see page 3) for the Precision value.

## 1.1.1.2.2.3 CompareOps.GetPrecision Method (double, double)

Returns the Precision Value to use using the Default Precision. See DoublePrecisionDef (⧉ see page 3).

### C++

```
public: double GetPrecision(
    double X,
    double Y
);
```

### C#

```
public static double GetPrecision(
    double X,
    double Y
);
```

### Visual Basic

```
Public static Function GetPrecision(
    X As double,
    Y As double
) As double
```

### Parameters

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |

### Remarks

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.2.4 CompareOps.GetPrecision Method (double, double, double)

Returns the Precision Value to use using the specified Precision.

**C++**

```
public: double GetPrecision(
    double X,
    double Y,
    double Precision
);
```

**C#**

```
public static double GetPrecision(
    double X,
    double Y,
    double Precision
);
```

**Visual Basic**

```
Public static Function GetPrecision(
    X As double,
    Y As double,
    Precision As double
) As double
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |
| double Precision | Precision Value to use - should be small and positive like DoublePrecisionDef (see page 3). |

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (see page 3) for the Precision value.

### 1.1.1.2.2.5 CompareOps.GetPrecision Method (float, float)

Returns the Precision Value to use using the Default Precision. See FloatPrecisionDef (see page 4).

**C++**

```
public: float GetPrecision(
    float X,
    float Y
);
```

**C#**

```
public static float GetPrecision(
    float X,
    float Y
);
```

**Visual Basic**

```
Public static Function GetPrecision(
    X As float,
    Y As float
) As float
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.2.6 CompareOps.GetPrecision Method (float, float, float)

Returns the Precision Value to use using the specified Precision

**C++**

```
public: float GetPrecision(
    float X,
    float Y,
    float Precision
);
```

**C#**

```
public static float GetPrecision(
    float X,
    float Y,
    float Precision
);
```

**Visual Basic**

```
Public static Function GetPrecision(
    X As float,
    Y As float,
    Precision As float
) As float
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |
| float Precision | Precision Value to use - should be small and positive like FloatPrecisionDef (see page 4). |

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (see page 4) for the Precision value.

## 1.1.1.2.3 GreaterValue Method

### 1.1.1.2.3.1 CompareOps.GreaterValue Method (decimal, decimal)

Returns True if Values are not within a "small" value of each other, and the First Value is greater. DecimalPrecisionDef (see page 3) defines the Precision.

**C++**

```
public: bool GreaterValue(
    decimal X,
    decimal Y
);
```

**C#**

```
public static bool GreaterValue(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As decimal,
    Y As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.3.2 CompareOps.GreaterValue Method (decimal, decimal, decimal)

Returns True if Values are not within a "small" value of each other, and the First Value is greater.

**C++**

```
public: bool GreaterValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**C#**

```
public static bool GreaterValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As decimal,
    Y As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (⇗ see page 3). |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (⇗ see page 3) for the Precision value.

### 1.1.1.2.3.3 CompareOps.GreaterValue Method (double, double)

Returns True if Values are not within a "small" value of each other, and the First Value is greater. DoublePrecisionDef (⇗ see page 3) defines the Precision.

**C++**

```
public: bool GreaterValue(
    double X,
    double Y
);
```

**C#**

```
public static bool GreaterValue(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As double,
    Y As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.3.4 CompareOps.GreaterValue Method (double, double, double)

Returns True if Values are not within a "small" value of each other, and the First Value is greater.

**C++**

```
public: bool GreaterValue(
    double X,
    double Y,
    double Precision
);
```

**C#**

```
public static bool GreaterValue(
    double X,
    double Y,
    double Precision
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As double,
    Y As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (see page 3) for the Precision value.

### 1.1.1.2.3.5 CompareOps.GreaterValue Method (float, float)

Returns True if Values are not within a "small" value of each other, and the First Value is greater. FloatPrecisionDef (see page 4) defines the Precision.

**C++**

```
public: bool GreaterValue(
    float X,
    float Y
);
```

**C#**

```
public static bool GreaterValue(
    float X,
    float Y
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As float,
    Y As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.3.6 CompareOps.GreaterValue Method (float, float, float)

Returns True if Values are not within a "small" value of each other, and the First Value is greater.

**C++**

```
public: bool GreaterValue(
    float X,
    float Y,
    float Precision
);
```

**C#**

```
public static bool GreaterValue(
    float X,
    float Y,
    float Precision
);
```

**Visual Basic**

```
Public static Function GreaterValue(
    X As float,
    Y As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (see page 4). |

**Returns**

True if First Value is Greater.

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (see page 4) for the Precision value.

## 1.1.1.2.4 LesserValue Method

### 1.1.1.2.4.1 CompareOps.LesserValue Method (decimal, decimal)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser. DecimalPrecisionDef (see page 3) defines the Precision.

**C++**

```
public: bool LesserValue(
    decimal X,
    decimal Y
);
```

**C#**

```
public static bool LesserValue(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As decimal,
    Y As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.4.2 CompareOps.LesserValue Method (decimal, decimal, decimal)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser.

**C++**

```
public: bool LesserValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**C#**

```
public static bool LesserValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As decimal,
    Y As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (⧉ see page 3). |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (⧉ see page 3) for the Precision value.

## 1.1.1.2.4.3 CompareOps.LesserValue Method (double, double)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser. DoublePrecisionDef (⧉ see page 3) defines the Precision.

**C++**

```
public: bool LesserValue(
    double X,
    double Y
);
```

**C#**

```
public static bool LesserValue(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As double,
    Y As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.4.4 CompareOps.LesserValue Method (double, double, double)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser.

**C++**

```
public: bool LesserValue(
    double X,
    double Y,
    double Precision
);
```

**C#**

```
public static bool LesserValue(
    double X,
    double Y,
    double Precision
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As double,
    Y As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (⧉ see page 3). |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (⧉ see page 3) for the Precision value.

### 1.1.1.2.4.5 CompareOps.LesserValue Method (float, float)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser. FloatPrecisionDef (⧉ see page 4) defines the Precision.

**C++**

```
public: bool LesserValue(
    float X,
    float Y
);
```

**C#**

```
public static bool LesserValue(
    float X,
    float Y
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As float,
    Y As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.4.6 CompareOps.LesserValue Method (float, float, float)

Returns True if Values are not within a "small" value of each other, and the First Value is lesser.

**C++**

```
public: bool LesserValue(
    float X,
    float Y,
    float Precision
);
```

**C#**

```
public static bool LesserValue(
    float X,
    float Y,
    float Precision
);
```

**Visual Basic**

```
Public static Function LesserValue(
    X As float,
    Y As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (↗ see page 4). |

**Returns**

True if First Value is Lesser.

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (↗ see page 4) for the Precision value.

## 1.1.1.2.5 SameValue Method

### 1.1.1.2.5.1 CompareOps.SameValue Method (decimal, decimal)

Returns True if the Values are within a "small" value of each other. If you want more control over the comparison then use CompareValue(decimal,decimal)

DecimalPrecisionDef (↗ see page 3) defines the Precision.

**C++**

```
public: bool SameValue(
    decimal X,
    decimal Y
);
```

**C#**

```
public static bool SameValue(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As decimal,
    Y As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to compare. |
| decimal Y | Second Value to compare. |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.5.2 CompareOps.SameValue Method (decimal, decimal, decimal)

Returns True if the Values are within a "small" value computed from the supplied Precision. For Values with lots of significant figures, then Precision should be made smaller. If you want more control over the comparison then use CompareValue(decimal,decimal,decimal)

**C++**

```
public: bool SameValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**C#**

```
public static bool SameValue(
    decimal X,
    decimal Y,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As decimal,
    Y As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | First Value to Compare. |
| decimal Y | Second Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (see page 3). |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (see page 3) for the Precision value.

### 1.1.1.2.5.3 CompareOps.SameValue Method (double, double)

Returns True if the Values are within a "small" value of each other. DoublePrecisionDef (see page 3) defines the Precision. If you want more control over the comparison then use CompareValue(double,double)

**C++**

```
public: bool SameValue(
    double X,
    double Y
);
```

**C#**

```
public static bool SameValue(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As double,
    Y As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to compare. |
| double Y | Second Value to compare. |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.5.4 CompareOps.SameValue Method (double, double, double)

Returns True if the Values are within a "small" value computed from the supplied Precision. For Values with lots of significant figures, then Precision should be made smaller. If you want more control over the comparison then use CompareValue(double,double,double)

**C++**

```
public: bool SameValue(
    double X,
    double Y,
    double Precision
);
```

**C#**

```
public static bool SameValue(
    double X,
    double Y,
    double Precision
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As double,
    Y As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | First Value to Compare. |
| double Y | Second Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (↗ see page 3) for the Precision value.

### 1.1.1.2.5.5 CompareOps.SameValue Method (float, float)

Returns True if the Values are within a "small" value of each other. FloatPrecisionDef (↗ see page 4) defines the Precision. If you want more control over the comparison then use CompareValue(float,float)

**C++**

```
public: bool SameValue(
    float X,
    float Y
);
```

**C#**

```
public static bool SameValue(
    float X,
    float Y
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As float,
    Y As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to compare. |
| float Y | Second Value to compare. |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.5.6 CompareOps.SameValue Method (float, float, float)

Returns True if the Values are within a "small" value computed from the supplied Precision. For Values with lots of significant figures, then Precision should be made smaller. If you want more control over the comparison then use CompareValue(float,float,float)

**C++**

```
public: bool SameValue(
    float X,
    float Y,
    float Precision
);
```

**C#**

```
public static bool SameValue(
    float X,
    float Y,
    float Precision
);
```

**Visual Basic**

```
Public static Function SameValue(
    X As float,
    Y As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | First Value to Compare. |
| float Y | Second Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (↗ see page 4). |

**Returns**

True if values are within Precision of each other.

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (↗ see page 4) for the Precision value.

## 1.1.1.2.6 ValueIsNegative Method

## 1.1.1.2.6.1 CompareOps.ValueIsNegative Method (decimal)

Returns True if Value is not within a "small" value of 0 and is Negative. DoublePrecisionDef (↗ see page 3) defines the Precision.

**C++**

```
public: bool ValueIsNegative(
    decimal X
);
```

**C#**

```
public static bool ValueIsNegative(
    decimal X
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |

**Returns**

True if Value is Negative.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.6.2 CompareOps.ValueIsNegative Method (decimal, decimal)

Returns True if Value is not within a "small" value of 0 and is Negative.

**C++**

```
public: bool ValueIsNegative(
    decimal X,
    decimal Precision
);
```

**C#**

```
public static bool ValueIsNegative(
    decimal X,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

True if X is strictly Negative.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (see page 3) for the Precision value.

## 1.1.1.2.6.3 CompareOps.ValueIsNegative Method (double)

Returns True if Value is not within a "small" value of 0 and is Negative. DoublePrecisionDef (see page 3) defines the Precision.

**C++**

```
public: bool ValueIsNegative(
    double X
);
```

**C#**

```
public static bool ValueIsNegative(
    double X
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |

**Returns**

True if Value is Negative.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.6.4 CompareOps.ValueIsNegative Method (double, double)

Returns True if Value is not within a "small" value of 0 and is Negative.

**C++**

```
public: bool ValueIsNegative(
    double X,
    double Precision
);
```

**C#**

```
public static bool ValueIsNegative(
    double X,
    double Precision
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |

| | |
|---|---|
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (see page 3). |

**Returns**

True if X is strictly Negative.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (see page 3) for the Precision value.

### 1.1.1.2.6.5 CompareOps.ValueIsNegative Method (float)

Returns True if Value is not within a "small" value of 0 and is Negative. DoublePrecisionDef (see page 3) defines the Precision.

**C++**

```
public: bool ValueIsNegative(
    float X
);
```

**C#**

```
public static bool ValueIsNegative(
    float X
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |

**Returns**

True if Value is Negative.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.6.6 CompareOps.ValueIsNegative Method (float, float)

Returns True if Value is not within a "small" value of 0 and is Negative.

**C++**

```
public: bool ValueIsNegative(
    float X,
    float Precision
);
```

**C#**

```
public static bool ValueIsNegative(
    float X,
    float Precision
);
```

**Visual Basic**

```
Public static Function ValueIsNegative(
    X As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (see page 3). |

**Returns**

True if X is strictly Negative.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (see page 3) for the Precision value.

## 1.1.1.2.7 ValueIsPositive Method

### 1.1.1.2.7.1 CompareOps.ValueIsPositive Method (decimal)

Returns True if Value is not within a "small" value of 0 and is Positive. DecimalPrecisionDef (see page 3) defines the Precision.

**C++**

```
public: bool ValueIsPositive(
    decimal X
);
```

**C#**

```
public static bool ValueIsPositive(
    decimal X
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |

**Returns**

True if Value is Positive.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.7.2 CompareOps.ValueIsPositive Method (decimal, decimal)

Returns True if Value is not within a "small" value of 0 and is Positive.

**C++**

```
public: bool ValueIsPositive(
    decimal X,
    decimal Precision
);
```

**C#**

```
public static bool ValueIsPositive(
    decimal X,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (↗ see page 3). |

**Returns**

True if X is strictly Positive.

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (↗ see page 3) for the Precision value.

## 1.1.1.2.7.3 CompareOps.ValueIsPositive Method (double)

Returns True if Value is not within a "small" value of 0 and is Positive. DoublePrecisionDef (↗ see page 3) defines the Precision.

**C++**

```
public: bool ValueIsPositive(
    double X
);
```

**C#**

```
public static bool ValueIsPositive(
    double X
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |

**Returns**

True if Value is Positive.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.7.4 CompareOps.ValueIsPositive Method (double, double)

Returns True if Value is not within a "small" value of 0 and is Positive.

**C++**

```
public: bool ValueIsPositive(
    double X,
    double Precision
);
```

**C#**

```
public static bool ValueIsPositive(
    double X,
    double Precision
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

True if X is strictly Positive.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (↗ see page 3) for the Precision value.

## 1.1.1.2.7.5 CompareOps.ValueIsPositive Method (float)

Returns True if Value is not within a "small" value of 0 and is Positive. FloatPrecisionDef (↗ see page 4) defines the Precision.

**C++**

```
public: bool ValueIsPositive(
    float X
);
```

**C#**

```
public static bool ValueIsPositive(
    float X
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |

**Returns**

True if Value is Positive.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.7.6 CompareOps.ValueIsPositive Method (float, float)

Returns True if Value is not within a "small" value of 0 and is Positive.

**C++**

```
public: bool ValueIsPositive(
    float X,
    float Precision
);
```

**C#**

```
public static bool ValueIsPositive(
    float X,
    float Precision
);
```

**Visual Basic**

```
Public static Function ValueIsPositive(
    X As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (↗ see page 4). |

**Returns**

True if X is strictly Positive.

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (↗ see page 4) for the Precision value.

## 1.1.1.2.8 ValueIsZero Method

### 1.1.1.2.8.1 CompareOps.ValueIsZero Method (decimal)

Returns True if Value is within a "small" value of 0. DecimalPrecisionDef (↗ see page 3) defines the Precision.

**C++**

```
public: bool ValueIsZero(
    decimal X
);
```

**C#**

```
public static bool ValueIsZero(
    decimal X
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.8.2 CompareOps.ValueIsZero Method (decimal, decimal)

Returns True if Value is within a "small" value of 0.

**C++**

```
public: bool ValueIsZero(
    decimal X,
    decimal Precision
);
```

**C#**

```
public static bool ValueIsZero(
    decimal X,
    decimal Precision
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As decimal,
    Precision As decimal
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to Compare. |
| decimal Precision | Precision to used for comparison, should be positive and small, see DecimalPrecisionDef (⧉ see page 3). |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version without the Precision that will use DecimalPrecisionDef (⧉ see page 3) for the Precision value.

### 1.1.1.2.8.3 CompareOps.ValueIsZero Method (double)

Returns True if Value is within a "small" value of 0. DoublePrecisionDef (⧉ see page 3) defines the Precision.

**C++**

```
public: bool ValueIsZero(
    double X
);
```

**C#**

```
public static bool ValueIsZero(
    double X
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

## 1.1.1.2.8.4 CompareOps.ValueIsZero Method (double, double)

Returns True if Value is within a "small" value of 0.

**C++**

```
public: bool ValueIsZero(
    double X,
    double Precision
);
```

**C#**

```
public static bool ValueIsZero(
    double X,
    double Precision
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As double,
    Precision As double
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to Compare. |
| double Precision | Precision to used for comparison, should be positive and small, see DoublePrecisionDef (↗ see page 3). |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version without the Precision that will use DoublePrecisionDef (↗ see page 3) for the Precision value.

## 1.1.1.2.8.5 CompareOps.ValueIsZero Method (float)

Returns True if Value is within a "small" value of 0. FloatPrecisionDef (↗ see page 4) defines the Precision.

**C++**

```
public: bool ValueIsZero(
    float X
);
```

**C#**

```
public static bool ValueIsZero(
    float X
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version that allows you to specify the Precision of the comparison.

### 1.1.1.2.8.6 CompareOps.ValueIsZero Method (float, float)

Returns True if Value is within a "small" value of 0.

**C++**

```
public: bool ValueIsZero(
    float X,
    float Precision
);
```

**C#**

```
public static bool ValueIsZero(
    float X,
    float Precision
);
```

**Visual Basic**

```
Public static Function ValueIsZero(
    X As float,
    Precision As float
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| float X | Value to Compare. |
| float Precision | Precision to used for comparison, should be positive and small, see FloatPrecisionDef (↗ see page 4). |

**Returns**

True if Value is very close to Zero.

**Remarks**

There is an overloaded version without the Precision that will use FloatPrecisionDef (↗ see page 4) for the Precision value.

# 1.1.2 DecimalGlobals Class

**Inheritance Hierarchy**

EsbDecimals.DecimalGlobals

**C++**

```
public: class DecimalGlobals;
```

**C#**

```
public static class DecimalGlobals;
```

**Visual Basic**

```
Public static Class DecimalGlobals
```

**File**

EsbGlobals-Decimals.cs

**Remarks**

Global Decimal Constants for EsbDecimals (see page 1).

**Members**

**DecimalGlobals Fields**

| | Name | Description |
|---|---|---|
| ◆ | Cbrt10 (see page 36) | Decimal Cube Root of 10. |
| ◆ | Cbrt100 (see page 36) | Decimal Cube Root of 100. |
| ◆ | Cbrt2 (see page 36) | Decimal Cube Root of 2. |
| ◆ | Cbrt3 (see page 36) | Decimal Cube Root of 3. |
| ◆ | CbrtPi (see page 36) | Decimal Cube Root of Pi. |
| ◆ | FivePiOver2 (see page 37) | Decimal Five Halves of Pi. |
| ◆ | FourPiOver3 (see page 37) | Decimal Four Thirds of Pi. |
| ◆ | Gamma (see page 37) | Gamma Constant - also known as Euler's Constant or Mascheroni's Constant.<br>lim ( 1 + 1/2 + 1/3 + 1/4 + ... + 1/n - ln(n) ) as n -> infinity |
| ◆ | GoldenRatio (see page 37) | Decimal Golden Ratio Constant which is defined as (1 + Sqrt (5)) / 2. |
| ◆ | InvCbrtPi (see page 38) | Decimal Inverse of Cube Root of Pi. |
| ◆ | InvPi (see page 38) | Decimal Inverse of Pi. |
| ◆ | InvSqrt2 (see page 38) | Decimal Inverse of Square Root of 2. |
| ◆ | InvSqrt3 (see page 38) | Decimal Inverse of Square Root of 3. |
| ◆ | InvSqrt5 (see page 38) | Decimal Inverse of Square Root of 5. |
| ◆ | InvSqrtPi (see page 39) | Decimal Inverse of Square Root of Pi. |
| ◆ | Ln10 (see page 39) | Decimal Natural Log of 10. |
| ◆ | Ln2 (see page 39) | Decimal Natural Log of 2. |
| ◆ | LnOfPi (see page 39) | Decimal Natural Log of Pi. |
| ◆ | LnSqrt2Pi (see page 40) | Decimal Natural Log of the Square Root of (2 * Pi) |
| ◆ | Log10Base2 (see page 40) | Decimal Log to Base 2 of 10. |
| ◆ | Log2 (see page 40) | Decimal Log to Base 10 of 2. |
| ◆ | Log3 (see page 40) | Decimal Log to Base 10 of 3. |
| ◆ | LogNaturalConstant (see page 40) | Decimal Log to Base 10 of Natural Constant. |

| | | |
|---|---|---|
| ◆ | LogPi (↗ see page 41) | Decimal Log to Base 10 of Pi. |
| ◆ | NaturalConstant (↗ see page 41) | Decimal Natural Constant. |
| ◆ | NaturalConstant2Pi (↗ see page 41) | Decimal Natural Constant raised to Pi. |
| ◆ | NaturalConstant2PiOver2 (↗ see page 41) | Decimal Natural Constant raised to Pi/2. |
| ◆ | NaturalConstant2PiOver4 (↗ see page 42) | Decimal Natural Constant raised to Pi/4. |
| ◆ | OneDegree (↗ see page 42) | Decimal One Degree in Radians. |
| ◆ | OneMinute (↗ see page 42) | Decimal One Minute in Radians. |
| ◆ | OneRadian (↗ see page 42) | Decimal One Radian in Degrees. |
| ◆ | OneSecond (↗ see page 42) | Decimal One Second in Radians. |
| ◆ | Pi (↗ see page 43) | Decimal Accurate Pi Constant. |
| ◆ | Pi2E (↗ see page 43) | Decimal Pi raised to the Natural Constant. |
| ◆ | PiOver12 (↗ see page 43) | Decimal Twelfth of Pi. |
| ◆ | PiOver2 (↗ see page 43) | Decimal Half of Pi. |
| ◆ | PiOver3 (↗ see page 44) | Decimal Third of Pi. |
| ◆ | PiOver4 (↗ see page 44) | Decimal Quarter of Pi. |
| ◆ | PiOver5 (↗ see page 44) | Decimal Fifth of Pi. |
| ◆ | PiOver6 (↗ see page 44) | Decimal Sixth of Pi. |
| ◆ | SqrNaturalConstant (↗ see page 44) | Decimal Square of Natural Constant. |
| ◆ | SqrPi (↗ see page 45) | Decimal Square of Pi. |
| ◆ | Sqrt10 (↗ see page 45) | Decimal Square Root of 10. |
| ◆ | Sqrt2 (↗ see page 45) | Decimal Square Root of 2. |
| ◆ | Sqrt3 (↗ see page 45) | Decimal Square Root of 3. |
| ◆ | Sqrt5 (↗ see page 46) | Decimal Square Root of 5. |
| ◆ | SqrtPi (↗ see page 46) | Decimal Square Root of Pi. |
| ◆ | ThreePi (↗ see page 46) | Decimal 3 * Pi. |
| ◆ | ThreePiOver2 (↗ see page 46) | Decimal Three Halves of Pi. |
| ◆ | ThreePiOver4 (↗ see page 46) | Decimal Three Quarters of Pi. |
| ◆ | TwoPi (↗ see page 47) | Decimal 2 * Pi. |
| ◆ | TwoToPower63 (↗ see page 47) | Decimal 2^63. |
| ◆ | Version (↗ see page 47) | String representing the Version of ESBDecimals. |
| ◆ | VersionDate (↗ see page 47) | String representing the Date of the latest Version (↗ see page 47) of ESBDecimals. |

### DecimalGlobals Fields

| | Name | Description |
|---|---|---|
| ◆ | Cbrt10 (↗ see page 36) | Decimal Cube Root of 10. |
| ◆ | Cbrt100 (↗ see page 36) | Decimal Cube Root of 100. |
| ◆ | Cbrt2 (↗ see page 36) | Decimal Cube Root of 2. |
| ◆ | Cbrt3 (↗ see page 36) | Decimal Cube Root of 3. |
| ◆ | CbrtPi (↗ see page 36) | Decimal Cube Root of Pi. |
| ◆ | FivePiOver2 (↗ see page 37) | Decimal Five Halves of Pi. |
| ◆ | FourPiOver3 (↗ see page 37) | Decimal Four Thirds of Pi. |
| ◆ | Gamma (↗ see page 37) | Gamma Constant - also known as Euler's Constant or Mascheroni's Constant.<br>lim ( 1 + 1/2 + 1/3 + 1/4 + ... + 1/n - ln(n) ) as n -> infinity |
| ◆ | GoldenRatio (↗ see page 37) | Decimal Golden Ratio Constant which is defined as (1 + Sqrt (5)) / 2. |
| ◆ | InvCbrtPi (↗ see page 38) | Decimal Inverse of Cube Root of Pi. |
| ◆ | InvPi (↗ see page 38) | Decimal Inverse of Pi. |

|  |  |  |
|---|---|---|
| ◆ | InvSqrt2 (↗ see page 38) | Decimal Inverse of Square Root of 2. |
| ◆ | InvSqrt3 (↗ see page 38) | Decimal Inverse of Square Root of 3. |
| ◆ | InvSqrt5 (↗ see page 38) | Decimal Inverse of Square Root of 5. |
| ◆ | InvSqrtPi (↗ see page 39) | Decimal Inverse of Square Root of Pi. |
| ◆ | Ln10 (↗ see page 39) | Decimal Natural Log of 10. |
| ◆ | Ln2 (↗ see page 39) | Decimal Natural Log of 2. |
| ◆ | LnOfPi (↗ see page 39) | Decimal Natural Log of Pi. |
| ◆ | LnSqrt2Pi (↗ see page 40) | Decimal Natural Log of the Square Root of (2 * Pi) |
| ◆ | Log10Base2 (↗ see page 40) | Decimal Log to Base 2 of 10. |
| ◆ | Log2 (↗ see page 40) | Decimal Log to Base 10 of 2. |
| ◆ | Log3 (↗ see page 40) | Decimal Log to Base 10 of 3. |
| ◆ | LogNaturalConstant (↗ see page 40) | Decimal Log to Base 10 of Natural Constant. |
| ◆ | LogPi (↗ see page 41) | Decimal Log to Base 10 of Pi. |
| ◆ | NaturalConstant (↗ see page 41) | Decimal Natural Constant. |
| ◆ | NaturalConstant2Pi (↗ see page 41) | Decimal Natural Constant raised to Pi. |
| ◆ | NaturalConstant2PiOver2 (↗ see page 41) | Decimal Natural Constant raised to Pi/2. |
| ◆ | NaturalConstant2PiOver4 (↗ see page 42) | Decimal Natural Constant raised to Pi/4. |
| ◆ | OneDegree (↗ see page 42) | Decimal One Degree in Radians. |
| ◆ | OneMinute (↗ see page 42) | Decimal One Minute in Radians. |
| ◆ | OneRadian (↗ see page 42) | Decimal One Radian in Degrees. |
| ◆ | OneSecond (↗ see page 42) | Decimal One Second in Radians. |
| ◆ | Pi (↗ see page 43) | Decimal Accurate Pi Constant. |
| ◆ | Pi2E (↗ see page 43) | Decimal Pi raised to the Natural Constant. |
| ◆ | PiOver12 (↗ see page 43) | Decimal Twelfth of Pi. |
| ◆ | PiOver2 (↗ see page 43) | Decimal Half of Pi. |
| ◆ | PiOver3 (↗ see page 44) | Decimal Third of Pi. |
| ◆ | PiOver4 (↗ see page 44) | Decimal Quarter of Pi. |
| ◆ | PiOver5 (↗ see page 44) | Decimal Fifth of Pi. |
| ◆ | PiOver6 (↗ see page 44) | Decimal Sixth of Pi. |
| ◆ | SqrNaturalConstant (↗ see page 44) | Decimal Square of Natural Constant. |
| ◆ | SqrPi (↗ see page 45) | Decimal Square of Pi. |
| ◆ | Sqrt10 (↗ see page 45) | Decimal Square Root of 10. |
| ◆ | Sqrt2 (↗ see page 45) | Decimal Square Root of 2. |
| ◆ | Sqrt3 (↗ see page 45) | Decimal Square Root of 3. |
| ◆ | Sqrt5 (↗ see page 46) | Decimal Square Root of 5. |
| ◆ | SqrtPi (↗ see page 46) | Decimal Square Root of Pi. |
| ◆ | ThreePi (↗ see page 46) | Decimal 3 * Pi. |
| ◆ | ThreePiOver2 (↗ see page 46) | Decimal Three Halves of Pi. |
| ◆ | ThreePiOver4 (↗ see page 46) | Decimal Three Quarters of Pi. |
| ◆ | TwoPi (↗ see page 47) | Decimal 2 * Pi. |
| ◆ | TwoToPower63 (↗ see page 47) | Decimal 2^63. |
| ◆ | Version (↗ see page 47) | String representing the Version of ESBDecimals. |
| ◆ | VersionDate (↗ see page 47) | String representing the Date of the latest Version (↗ see page 47) of ESBDecimals. |

## 1.1.2.1 DecimalGlobals Fields

## 1.1.2.1.1 DecimalGlobals.Cbrt10 Field

Decimal Cube Root of 10.

**C++**

```
public: decimal Cbrt10 = 2.1544346900318837217592935665194m;
```

**C#**

```
public const decimal Cbrt10 = 2.1544346900318837217592935665194m;
```

**Visual Basic**

```
Public Const Cbrt10 As decimal = 2.1544346900318837217592935665194m
```

## 1.1.2.1.2 DecimalGlobals.Cbrt100 Field

Decimal Cube Root of 100.

**C++**

```
public: decimal Cbrt100 = 4.6415888336127788924100763509194m;
```

**C#**

```
public const decimal Cbrt100 = 4.6415888336127788924100763509194m;
```

**Visual Basic**

```
Public Const Cbrt100 As decimal = 4.6415888336127788924100763509194m
```

## 1.1.2.1.3 DecimalGlobals.Cbrt2 Field

Decimal Cube Root of 2.

**C++**

```
public: decimal Cbrt2 = 1.2599210498948731647672106072782m;
```

**C#**

```
public const decimal Cbrt2 = 1.2599210498948731647672106072782m;
```

**Visual Basic**

```
Public Const Cbrt2 As decimal = 1.2599210498948731647672106072782m
```

## 1.1.2.1.4 DecimalGlobals.Cbrt3 Field

Decimal Cube Root of 3.

**C++**

```
public: decimal Cbrt3 = 1.4422495703074083823216383107801m;
```

**C#**

```
public const decimal Cbrt3 = 1.4422495703074083823216383107801m;
```

**Visual Basic**

```
Public Const Cbrt3 As decimal = 1.4422495703074083823216383107801m
```

## 1.1.2.1.5 DecimalGlobals.CbrtPi Field

Decimal Cube Root of Pi.

**C++**

```
public: decimal CbrtPi = 1.4645918875615232630201425272638m;
```

**C#**

```
public const decimal CbrtPi = 1.4645918875615232630201425272638m;
```

**Visual Basic**

```
Public Const CbrtPi As decimal = 1.4645918875615232630201425272638m
```

### 1.1.2.1.6 DecimalGlobals.FivePiOver2 Field

Decimal Five Halves of Pi.

**C++**

```
public: decimal FivePiOver2 = 7.8539816339744830961566084581988m;
```

**C#**

```
public const decimal FivePiOver2 = 7.8539816339744830961566084581988m;
```

**Visual Basic**

```
Public Const FivePiOver2 As decimal = 7.8539816339744830961566084581988m
```

### 1.1.2.1.7 DecimalGlobals.FourPiOver3 Field

Decimal Four Thirds of Pi.

**C++**

```
public: decimal FourPiOver3 = 4.1887902047863909846168578443727m;
```

**C#**

```
public const decimal FourPiOver3 = 4.1887902047863909846168578443727m;
```

**Visual Basic**

```
Public Const FourPiOver3 As decimal = 4.1887902047863909846168578443727m
```

### 1.1.2.1.8 DecimalGlobals.Gamma Field

Gamma Constant - also known as Euler's Constant or Mascheroni's Constant.

lim ( 1 + 1/2 + 1/3 + 1/4 + ... + 1/n - ln(n) ) as n -> infinity

**C++**

```
public: decimal Gamma = 0.5772156649015328606065120900824m;
```

**C#**

```
public const decimal Gamma = 0.5772156649015328606065120900824m;
```

**Visual Basic**

```
Public Const Gamma As decimal = 0.5772156649015328606065120900824m
```

### 1.1.2.1.9 DecimalGlobals.GoldenRatio Field

Decimal Golden Ratio Constant which is defined as (1 + Sqrt (5)) / 2.

**C++**

```
public: decimal GoldenRatio = 1.6180339887498948482045868343656m;
```

**C#**

```
public const decimal GoldenRatio = 1.6180339887498948482045868343656m;
```

**Visual Basic**

```
Public Const GoldenRatio As decimal = 1.6180339887498948482045868343656m
```

## 1.1.2.1.10 DecimalGlobals.InvCbrtPi Field

Decimal Inverse of Cube Root of Pi.

**C++**

```
public: decimal InvCbrtPi = 0.68278406325529568146702083315816m;
```

**C#**

```
public const decimal InvCbrtPi = 0.68278406325529568146702083315816m;
```

**Visual Basic**

```
Public Const InvCbrtPi As decimal = 0.68278406325529568146702083315816m
```

## 1.1.2.1.11 DecimalGlobals.InvPi Field

Decimal Inverse of Pi.

**C++**

```
public: decimal InvPi = 0.31830988618379067153776752674503m;
```

**C#**

```
public const decimal InvPi = 0.31830988618379067153776752674503m;
```

**Visual Basic**

```
Public Const InvPi As decimal = 0.31830988618379067153776752674503m
```

## 1.1.2.1.12 DecimalGlobals.InvSqrt2 Field

Decimal Inverse of Square Root of 2.

**C++**

```
public: decimal InvSqrt2 = 0.70710678118654752440084436210485m;
```

**C#**

```
public const decimal InvSqrt2 = 0.70710678118654752440084436210485m;
```

**Visual Basic**

```
Public Const InvSqrt2 As decimal = 0.70710678118654752440084436210485m
```

## 1.1.2.1.13 DecimalGlobals.InvSqrt3 Field

Decimal Inverse of Square Root of 3.

**C++**

```
public: decimal InvSqrt3 = 0.57735026918962576450914878050196m;
```

**C#**

```
public const decimal InvSqrt3 = 0.57735026918962576450914878050196m;
```

**Visual Basic**

```
Public Const InvSqrt3 As decimal = 0.57735026918962576450914878050196m
```

## 1.1.2.1.14 DecimalGlobals.InvSqrt5 Field

Decimal Inverse of Square Root of 5.

**C++**

```
public: decimal InvSqrt5 = 0.44721359549995793928183473374626m;
```

**C#**

```
public const decimal InvSqrt5 = 0.44721359549995793928183473374626m;
```

**Visual Basic**

```
Public Const InvSqrt5 As decimal = 0.44721359549995793928183473374626m
```

## 1.1.2.1.15 DecimalGlobals.InvSqrtPi Field

Decimal Inverse of Square Root of Pi.

**C++**

```
public: decimal InvSqrtPi = 0.56418958354775628694807945156077m;
```

**C#**

```
public const decimal InvSqrtPi = 0.56418958354775628694807945156077m;
```

**Visual Basic**

```
Public Const InvSqrtPi As decimal = 0.56418958354775628694807945156077m
```

## 1.1.2.1.16 DecimalGlobals.Ln10 Field

Decimal Natural Log of 10.

**C++**

```
public: decimal Ln10 = 2.3025850929940456840179914546844m;
```

**C#**

```
public const decimal Ln10 = 2.3025850929940456840179914546844m;
```

**Visual Basic**

```
Public Const Ln10 As decimal = 2.3025850929940456840179914546844m
```

## 1.1.2.1.17 DecimalGlobals.Ln2 Field

Decimal Natural Log of 2.

**C++**

```
public: decimal Ln2 = 0.69314718055994530941723212145818m;
```

**C#**

```
public const decimal Ln2 = 0.69314718055994530941723212145818m;
```

**Visual Basic**

```
Public Const Ln2 As decimal = 0.69314718055994530941723212145818m
```

## 1.1.2.1.18 DecimalGlobals.LnOfPi Field

Decimal Natural Log of Pi.

**C++**

```
public: decimal LnOfPi = 1.1447298858494001741434273513531m;
```

**C#**

```
public const decimal LnOfPi = 1.1447298858494001741434273513531m;
```

**Visual Basic**

```
Public Const LnOfPi As decimal = 1.1447298858494001741434273513531m
```

## 1.1.2.1.19 DecimalGlobals.LnSqrt2Pi Field

Decimal Natural Log of the Square Root of (2 * Pi)

**C++**

```
public: decimal LnSqrt2Pi = 9.1893853320467274178032973640562e-1m;
```

**C#**

```
public const decimal LnSqrt2Pi = 9.1893853320467274178032973640562e-1m;
```

**Visual Basic**

```
Public Const LnSqrt2Pi As decimal = 9.1893853320467274178032973640562e-1m
```

## 1.1.2.1.20 DecimalGlobals.Log10Base2 Field

Decimal Log to Base 2 of 10.

**C++**

```
public: decimal Log10Base2 = 3.3219280948873623478703194294894m;
```

**C#**

```
public const decimal Log10Base2 = 3.3219280948873623478703194294894m;
```

**Visual Basic**

```
Public Const Log10Base2 As decimal = 3.3219280948873623478703194294894m
```

## 1.1.2.1.21 DecimalGlobals.Log2 Field

Decimal Log to Base 10 of 2.

**C++**

```
public: decimal Log2 = 0.30102999566398119521373889472449m;
```

**C#**

```
public const decimal Log2 = 0.30102999566398119521373889472449m;
```

**Visual Basic**

```
Public Const Log2 As decimal = 0.30102999566398119521373889472449m
```

## 1.1.2.1.22 DecimalGlobals.Log3 Field

Decimal Log to Base 10 of 3.

**C++**

```
public: decimal Log3 = 0.47712125471966243729502790325512m;
```

**C#**

```
public const decimal Log3 = 0.47712125471966243729502790325512m;
```

**Visual Basic**

```
Public Const Log3 As decimal = 0.47712125471966243729502790325512m
```

## 1.1.2.1.23 DecimalGlobals.LogNaturalConstant Field

Decimal Log to Base 10 of Natural Constant.

**C++**

```
public: decimal LogNaturalConstant = 0.43429448190325182765112891891661m;
```

**C#**

```
public const decimal LogNaturalConstant = 0.4971498726941338543512682882909m;
```

**Visual Basic**

```
Public Const LogNaturalConstant As decimal = 0.4971498726941338543512682882909m
```

### 1.1.2.1.24 DecimalGlobals.LogPi Field

Decimal Log to Base 10 of Pi.

**C++**

```
public: decimal LogPi = 0.4971498726941339m;
```

**C#**

```
public const decimal LogPi = 0.4971498726941339m;
```

**Visual Basic**

```
Public Const LogPi As decimal = 0.4971498726941339m
```

### 1.1.2.1.25 DecimalGlobals.NaturalConstant Field

Decimal Natural Constant.

**C++**

```
public: decimal NaturalConstant = 2.7182818284590452353602874713527m;
```

**C#**

```
public const decimal NaturalConstant = 2.7182818284590452353602874713527m;
```

**Visual Basic**

```
Public Const NaturalConstant As decimal = 2.7182818284590452353602874713527m
```

### 1.1.2.1.26 DecimalGlobals.NaturalConstant2Pi Field

Decimal Natural Constant raised to Pi.

**C++**

```
public: decimal NaturalConstant2Pi = 23.140692632779269005729086367949m;
```

**C#**

```
public const decimal NaturalConstant2Pi = 23.140692632779269005729086367949m;
```

**Visual Basic**

```
Public Const NaturalConstant2Pi As decimal = 23.140692632779269005729086367949m
```

### 1.1.2.1.27 DecimalGlobals.NaturalConstant2PiOver2 Field

Decimal Natural Constant raised to Pi/2.

**C++**

```
public: decimal NaturalConstant2PiOver2 = 4.8104773809653516554730356667038m;
```

**C#**

```
public const decimal NaturalConstant2PiOver2 = 4.8104773809653516554730356667038m;
```

**Visual Basic**

```
Public Const NaturalConstant2PiOver2 As decimal = 4.8104773809653516554730356667038m
```

## 1.1.2.1.28 DecimalGlobals.NaturalConstant2PiOver4 Field

Decimal Natural Constant raised to Pi/4.

**C++**

```
public: decimal NaturalConstant2PiOver4 = 2.1932800507380154565597696592787m;
```

**C#**

```
public const decimal NaturalConstant2PiOver4 = 2.1932800507380154565597696592787m;
```

**Visual Basic**

```
Public Const NaturalConstant2PiOver4 As decimal = 2.1932800507380154565597696592787m
```

## 1.1.2.1.29 DecimalGlobals.OneDegree Field

Decimal One Degree in Radians.

**C++**

```
public: decimal OneDegree = 1.7453292519943295769236907684886e-2m;
```

**C#**

```
public const decimal OneDegree = 1.7453292519943295769236907684886e-2m;
```

**Visual Basic**

```
Public Const OneDegree As decimal = 1.7453292519943295769236907684886e-2m
```

## 1.1.2.1.30 DecimalGlobals.OneMinute Field

Decimal One Minute in Radians.

**C++**

```
public: decimal OneMinute = 2.9088820866572159615394846141477e-4m;
```

**C#**

```
public const decimal OneMinute = 2.9088820866572159615394846141477e-4m;
```

**Visual Basic**

```
Public Const OneMinute As decimal = 2.9088820866572159615394846141477e-4m
```

## 1.1.2.1.31 DecimalGlobals.OneRadian Field

Decimal One Radian in Degrees.

**C++**

```
public: decimal OneRadian = 57.295779513082320876798154814105m;
```

**C#**

```
public const decimal OneRadian = 57.295779513082320876798154814105m;
```

**Visual Basic**

```
Public Const OneRadian As decimal = 57.295779513082320876798154814105m
```

## 1.1.2.1.32 DecimalGlobals.OneSecond Field

Decimal One Second in Radians.

**C++**

```
public: decimal OneSecond = 4.8481368110953599358991410235795e-6m;
```

**C#**

```
public const decimal OneSecond = 4.8481368110953599358991410235795e-6m;
```

**Visual Basic**

```
Public Const OneSecond As decimal = 4.8481368110953599358991410235795e-6m
```

## 1.1.2.1.33 DecimalGlobals.Pi Field

Decimal Accurate Pi Constant.

**C++**

```
public: decimal Pi = 3.1415926535897932384626433832795m;
```

**C#**

```
public const decimal Pi = 3.1415926535897932384626433832795m;
```

**Visual Basic**

```
Public Const Pi As decimal = 3.1415926535897932384626433832795m
```

## 1.1.2.1.34 DecimalGlobals.Pi2E Field

Decimal Pi raised to the Natural Constant.

**C++**

```
public: decimal Pi2E = 22.45915771836104547342715220454m;
```

**C#**

```
public const decimal Pi2E = 22.45915771836104547342715220454m;
```

**Visual Basic**

```
Public Const Pi2E As decimal = 22.45915771836104547342715220454m
```

## 1.1.2.1.35 DecimalGlobals.PiOver12 Field

Decimal Twelfth of Pi.

**C++**

```
public: decimal PiOver12 = 0.26179938779914943653855361527329m;
```

**C#**

```
public const decimal PiOver12 = 0.26179938779914943653855361527329m;
```

**Visual Basic**

```
Public Const PiOver12 As decimal = 0.26179938779914943653855361527329m
```

## 1.1.2.1.36 DecimalGlobals.PiOver2 Field

Decimal Half of Pi.

**C++**

```
public: decimal PiOver2 = 1.5707963267948966192313216916398m;
```

**C#**

```
public const decimal PiOver2 = 1.5707963267948966192313216916398m;
```

**Visual Basic**

```
Public Const PiOver2 As decimal = 1.5707963267948966192313216916398m
```

## 1.1.2.1.37 DecimalGlobals.PiOver3 Field

Decimal Third of Pi.

**C++**

```
public: decimal PiOver3 = 1.0471975511965977461542144610932m;
```

**C#**

```
public const decimal PiOver3 = 1.0471975511965977461542144610932m;
```

**Visual Basic**

```
Public Const PiOver3 As decimal = 1.0471975511965977461542144610932m
```

## 1.1.2.1.38 DecimalGlobals.PiOver4 Field

Decimal Quarter of Pi.

**C++**

```
public: decimal PiOver4 = 0.78539816339744830961566084581988m;
```

**C#**

```
public const decimal PiOver4 = 0.78539816339744830961566084581988m;
```

**Visual Basic**

```
Public Const PiOver4 As decimal = 0.78539816339744830961566084581988m
```

## 1.1.2.1.39 DecimalGlobals.PiOver5 Field

Decimal Fifth of Pi.

**C++**

```
public: decimal PiOver5 = 0.62831853071795864769252867655901m;
```

**C#**

```
public const decimal PiOver5 = 0.62831853071795864769252867655901m;
```

**Visual Basic**

```
Public Const PiOver5 As decimal = 0.62831853071795864769252867655901m
```

## 1.1.2.1.40 DecimalGlobals.PiOver6 Field

Decimal Sixth of Pi.

**C++**

```
public: decimal PiOver6 = 0.52359877559829887307710723054658m;
```

**C#**

```
public const decimal PiOver6 = 0.52359877559829887307710723054658m;
```

**Visual Basic**

```
Public Const PiOver6 As decimal = 0.52359877559829887307710723054658m
```

## 1.1.2.1.41 DecimalGlobals.SqrNaturalConstant Field

Decimal Square of Natural Constant.

**C++**

```
public: decimal SqrNaturalConstant = 7.3890560989306502272304274605750m;
```

**C#**

```
public const decimal SqrNaturalConstant = 7.389056098930650227230427460575m;
```

**Visual Basic**

```
Public Const SqrNaturalConstant As decimal = 7.389056098930650227230427460575m
```

## 1.1.2.1.42 DecimalGlobals.SqrPi Field

Decimal Square of Pi.

**C++**

```
public: decimal SqrPi = 9.8696044010893586188344909998762m;
```

**C#**

```
public const decimal SqrPi = 9.8696044010893586188344909998762m;
```

**Visual Basic**

```
Public Const SqrPi As decimal = 9.8696044010893586188344909998762m
```

## 1.1.2.1.43 DecimalGlobals.Sqrt10 Field

Decimal Square Root of 10.

**C++**

```
public: decimal Sqrt10 = 3.1622776601683793319988935444327m;
```

**C#**

```
public const decimal Sqrt10 = 3.1622776601683793319988935444327m;
```

**Visual Basic**

```
Public Const Sqrt10 As decimal = 3.1622776601683793319988935444327m
```

## 1.1.2.1.44 DecimalGlobals.Sqrt2 Field

Decimal Square Root of 2.

**C++**

```
public: decimal Sqrt2 = 1.4142135623730950488016887242097m;
```

**C#**

```
public const decimal Sqrt2 = 1.4142135623730950488016887242097m;
```

**Visual Basic**

```
Public Const Sqrt2 As decimal = 1.4142135623730950488016887242097m
```

## 1.1.2.1.45 DecimalGlobals.Sqrt3 Field

Decimal Square Root of 3.

**C++**

```
public: decimal Sqrt3 = 1.7320508075688772935274463415059m;
```

**C#**

```
public const decimal Sqrt3 = 1.7320508075688772935274463415059m;
```

**Visual Basic**

```
Public Const Sqrt3 As decimal = 1.7320508075688772935274463415059m
```

## 1.1.2.1.46 DecimalGlobals.Sqrt5 Field

Decimal Square Root of 5.

**C++**

```
public: decimal Sqrt5 = 2.2360679774997896964091736687313m;
```

**C#**

```
public const decimal Sqrt5 = 2.2360679774997896964091736687313m;
```

**Visual Basic**

```
Public Const Sqrt5 As decimal = 2.2360679774997896964091736687313m
```

## 1.1.2.1.47 DecimalGlobals.SqrtPi Field

Decimal Square Root of Pi.

**C++**

```
public: decimal SqrtPi = 1.7724538509055160272981674833411m;
```

**C#**

```
public const decimal SqrtPi = 1.7724538509055160272981674833411m;
```

**Visual Basic**

```
Public Const SqrtPi As decimal = 1.7724538509055160272981674833411m
```

## 1.1.2.1.48 DecimalGlobals.ThreePi Field

Decimal 3 * Pi.

**C++**

```
public: decimal ThreePi = 9.4247779607693797153879301498385m;
```

**C#**

```
public const decimal ThreePi = 9.4247779607693797153879301498385m;
```

**Visual Basic**

```
Public Const ThreePi As decimal = 9.4247779607693797153879301498385m
```

## 1.1.2.1.49 DecimalGlobals.ThreePiOver2 Field

Decimal Three Halves of Pi.

**C++**

```
public: decimal ThreePiOver2 = 4.7123889803846898576939650749193m;
```

**C#**

```
public const decimal ThreePiOver2 = 4.7123889803846898576939650749193m;
```

**Visual Basic**

```
Public Const ThreePiOver2 As decimal = 4.7123889803846898576939650749193m
```

## 1.1.2.1.50 DecimalGlobals.ThreePiOver4 Field

Decimal Three Quarters of Pi.

**C++**

```
public: decimal ThreePiOver4 = 2.3561944901923449288469825374596m;
```

**C#**

```
public const decimal ThreePiOver4 = 2.3561944901923449288469825374596m;
```

**Visual Basic**

```
Public Const ThreePiOver4 As decimal = 2.3561944901923449288469825374596m
```

## 1.1.2.1.51 DecimalGlobals.TwoPi Field

Decimal 2 * Pi.

**C++**

```
public: decimal TwoPi = 6.283185307179586476925286766559m;
```

**C#**

```
public const decimal TwoPi = 6.283185307179586476925286766559m;
```

**Visual Basic**

```
Public Const TwoPi As decimal = 6.283185307179586476925286766559m
```

## 1.1.2.1.52 DecimalGlobals.TwoToPower63 Field

Decimal 2^63.

**C++**

```
public: decimal TwoToPower63 = 9223372036854775808.0m;
```

**C#**

```
public const decimal TwoToPower63 = 9223372036854775808.0m;
```

**Visual Basic**

```
Public Const TwoToPower63 As decimal = 9223372036854775808.0m
```

## 1.1.2.1.53 DecimalGlobals.Version Field

String representing the Version of ESBDecimals.

**C++**

```
public: string Version = "2.1.0";
```

**C#**

```
public const string Version = "2.1.0";
```

**Visual Basic**

```
Public Const Version As string = "2.1.0"
```

## 1.1.2.1.54 DecimalGlobals.VersionDate Field

String representing the Date of the latest Version (↗ see page 47) of ESBDecimals.

**C++**

```
public: string VersionDate = "21 June 2011";
```

**C#**

```
public const string VersionDate = "21 June 2011";
```

**Visual Basic**

```
Public Const VersionDate As string = "21 June 2011"
```

## 1.1.3 DecimalOps Class

**Inheritance Hierarchy**

EsbDecimals.DecimalOps

**C++**

```
public: class DecimalOps;
```

**C#**

```
public static class DecimalOps;
```

**Visual Basic**

```
Public static Class DecimalOps
```

**File**

EsbMath-Decimals.cs

**Remarks**

Decimal Routines that are missing from the Framework including Sqrt (see page 115), etc for EsbDecimals (see page 1).

**Members**

**DecimalOps Methods**

| | Name | Description |
|---|---|---|
| S | Acos (see page 51) | Computes the angle (in Radians) that resulted in this Cosine Value using Asin(decimal). Thus computing the inverse cosine or arccos. |
| S | AcosDeg (see page 52) | Computes the Inverse Cosine returning the Angle in Degrees. Uses Acos(decimal). |
| S | AdjustAngleDeg (see page 53) | Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) Works in Degrees. |
| S | AdjustAngleRad (see page 56) | Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) Works in Radians. |
| S | Asin (see page 59) | Computes the angle (in Radians) that resulted in this Sine Value using Series Expansion. Thus computing the inverse sine or arcsin. |
| S | AsinDeg (see page 60) | Computes the Inverse Sine returning the Angle in Degrees. Uses Asin(decimal). |
| S | Atan (see page 61) | Computes the angle (in Radians) that resulted in this Tangent Value using Series Expansion. Thus computing the inverse tangent or arctan.<br>If you want to cover all quadrants then you may want to use Atan(decimal) |
| S | Atan2 (see page 63) | Computes the angle (in Radians) that resulted in this Tangent value of Y / X using Atan(decimal). Thus computing the inverse tangent or arctan. |
| S | Atan2Deg (see page 64) | Computes the Inverse Tangent, where the Tangent is Y /X, returning the Angle in Degrees. Uses Atan(decimal). |
| S | AtanDeg (see page 66) | Computes the Inverse Tangent returning the Angle in Degrees. Uses Atan(decimal). |
| S | Ceiling (see page 67) | Returns the Integral Value that is greater than or equal to the specified number. |
| S | Cos (see page 68) | Computes the Cosine of the given angle (in Radians) using Series Expansion. |

| | | |
|---|---|---|
| ◆ S | CosDeg (see page 70) | Computes the Cosine of the Angle, given the Angle in Degrees, using Cos(decimal). |
| ◆ S | Cosec (see page 71) | Computes the Cosecant of the given angle (in Radians) using . |
| ◆ S | CosecDeg (see page 73) | Computes the Cosecant of the given angle (in Degrees) using . |
| ◆ S | Cot (see page 75) | Computes the Cotangent of the given angle (in Radians). |
| ◆ S | CotDeg (see page 76) | Computes the Cotangent of the given angle (in Degrees). |
| ◆ S | Deg2Dms (see page 78) | Converts a Decimal Degree into Degrees/Minutes/Seconds. Returned values are positive with Sign containing the System.Math.Sign(decimal) of the original value. |
| ◆ S | Deg2Grad (see page 79) | Converts Degrees into Grads. |
| ◆ S | Deg2Rad (see page 80) | Converts Degrees into Radians. |
| ◆ S | Dms2Deg (see page 82) | Converts Degrees/Minutes/Seconds into Decimal Degrees. |
| ◆ S | Exp (see page 83) | Computes the Exponential Function using Series Expansion. |
| ◆ S | FractionPart (see page 85) | Returns the Fractional portion of the value supplied. |
| ◆ S | Grad2Deg (see page 86) | Converts Grads into Degrees. |
| ◆ S | Grad2Rad (see page 87) | Converts Grads into Radians. |
| ◆ S | IntPart (see page 88) | Returns the Integral portion of the value supplied. |
| ◆ S | IntPow (see page 89) | Computes the value of a Decimal raised to an Integer Power. |
| ◆ S | Log (see page 91) | Computes the Natural Logarithm using Series Expansion. |
| ◆ S | Log10 (see page 92) | Computes the Logarithm to Base 10 using Series Expansion. |
| ◆ S | Log2 (see page 94) | Computes the Logarithm to Base 2 using Series Expansion. |
| ◆ S | NthRoot (see page 96) | Computes the Decimal Nth Root using Newton's Method. |
| ◆ S | Pow (see page 97) | Computes the First Value raised to the power of the Second Value using series Expansion. |
| ◆ S | Rad2Deg (see page 103) | Converts Radians into Degrees. |
| ◆ S | Rad2Grad (see page 104) | Converts Radians into Grads. |
| ◆ S | Sec (see page 105) | Computes the Secant of the given angle (in Radians) using . |
| ◆ S | SecDeg (see page 107) | Computes the Secant of the given angle (in Degrees) using . |
| ◆ S | SignXY (see page 109) | Returns the FORTRAN type SIGN of the Values - basically it returns a value with the Magnitude of First Value and the Sign of the Second Value. |
| ◆ S | Sin (see page 111) | Computes the Sine of the given angle (in Radians) using Series Expansion. |
| ◆ S | SinCos (see page 112) | Computes the Sine and Cosine of the given angle (in Radians) using Series Expansion. |
| ◆ S | SinDeg (see page 114) | Computes the Sine of the Angle, given the Angle in Degrees, using Sin(decimal). |
| ◆ S | Sqrt (see page 115) | Computes the Decimal Square Root using Newton's Method. |
| ◆ S | Tan (see page 117) | Computes the Tangent of the given angle (in Radians). |
| ◆ S | TanDeg (see page 118) | Computes the Tangent of the Angle, given the Angle in Degrees, using Tan(decimal). |
| ◆ S | TenPow (see page 120) | Computes the Specified Power of 10 using series Expansion. |
| ◆ S | TwoPow (see page 121) | Computes the Specified Power of 2 using series Expansion. |

### DecimalOps Methods

| | Name | Description |
|---|---|---|
| ◆ S | Acos (↗ see page 51) | Computes the angle (in Radians) that resulted in this Cosine Value using Asin(decimal). Thus computing the inverse cosine or arccos. |
| ◆ S | AcosDeg (↗ see page 52) | Computes the Inverse Cosine returning the Angle in Degrees. Uses Acos(decimal). |
| ◆ S | AdjustAngleDeg (↗ see page 53) | Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) Works in Degrees. |
| ◆ S | AdjustAngleRad (↗ see page 56) | Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) Works in Radians. |
| ◆ S | Asin (↗ see page 59) | Computes the angle (in Radians) that resulted in this Sine Value using Series Expansion. Thus computing the inverse sine or arcsin. |
| ◆ S | AsinDeg (↗ see page 60) | Computes the Inverse Sine returning the Angle in Degrees. Uses Asin(decimal). |
| ◆ S | Atan (↗ see page 61) | Computes the angle (in Radians) that resulted in this Tangent Value using Series Expansion. Thus computing the inverse tangent or arctan.<br>If you want to cover all quadrants then you may want to use Atan(decimal) |
| ◆ S | Atan2 (↗ see page 63) | Computes the angle (in Radians) that resulted in this Tangent value of Y / X using Atan(decimal). Thus computing the inverse tangent or arctan. |
| ◆ S | Atan2Deg (↗ see page 64) | Computes the Inverse Tangent, where the Tangent is Y /X, returning the Angle in Degrees. Uses Atan(decimal). |
| ◆ S | AtanDeg (↗ see page 66) | Computes the Inverse Tangent returning the Angle in Degrees. Uses Atan(decimal). |
| ◆ S | Ceiling (↗ see page 67) | Returns the Integral Value that is greater than or equal to the specified number. |
| ◆ S | Cos (↗ see page 68) | Computes the Cosine of the given angle (in Radians) using Series Expansion. |
| ◆ S | CosDeg (↗ see page 70) | Computes the Cosine of the Angle, given the Angle in Degrees, using Cos(decimal). |
| ◆ S | Cosec (↗ see page 71) | Computes the Cosecant of the given angle (in Radians) using . |
| ◆ S | CosecDeg (↗ see page 73) | Computes the Cosecant of the given angle (in Degrees) using . |
| ◆ S | Cot (↗ see page 75) | Computes the Cotangent of the given angle (in Radians). |
| ◆ S | CotDeg (↗ see page 76) | Computes the Cotangent of the given angle (in Degrees). |
| ◆ S | Deg2Dms (↗ see page 78) | Converts a Decimal Degree into Degrees/Minutes/Seconds. Returned values are positive with Sign containing the System.Math.Sign(decimal) of the original value. |
| ◆ S | Deg2Grad (↗ see page 79) | Converts Degrees into Grads. |
| ◆ S | Deg2Rad (↗ see page 80) | Converts Degrees into Radians. |
| ◆ S | Dms2Deg (↗ see page 82) | Converts Degrees/Minutes/Seconds into Decimal Degrees. |
| ◆ S | Exp (↗ see page 83) | Computes the Exponential Function using Series Expansion. |
| ◆ S | FractionPart (↗ see page 85) | Returns the Fractional portion of the value supplied. |
| ◆ S | Grad2Deg (↗ see page 86) | Converts Grads into Degrees. |
| ◆ S | Grad2Rad (↗ see page 87) | Converts Grads into Radians. |
| ◆ S | IntPart (↗ see page 88) | Returns the Integral portion of the value supplied. |
| ◆ S | IntPow (↗ see page 89) | Computes the value of a Decimal raised to an Integer Power. |
| ◆ S | Log (↗ see page 91) | Computes the Natural Logarithm using Series Expansion. |

| | | |
|---|---|---|
| S | Log10 (↗ see page 92) | Computes the Logarithm to Base 10 using Series Expansion. |
| S | Log2 (↗ see page 94) | Computes the Logarithm to Base 2 using Series Expansion. |
| S | NthRoot (↗ see page 96) | Computes the Decimal Nth Root using Newton's Method. |
| S | Pow (↗ see page 97) | Computes the First Value raised to the power of the Second Value using series Expansion. |
| S | Rad2Deg (↗ see page 103) | Converts Radians into Degrees. |
| S | Rad2Grad (↗ see page 104) | Converts Radians into Grads. |
| S | Sec (↗ see page 105) | Computes the Secant of the given angle (in Radians) using . |
| S | SecDeg (↗ see page 107) | Computes the Secant of the given angle (in Degrees) using . |
| S | SignXY (↗ see page 109) | Returns the FORTRAN type SIGN of the Values - basically it returns a value with the Magnitude of First Value and the Sign of the Second Value. |
| S | Sin (↗ see page 111) | Computes the Sine of the given angle (in Radians) using Series Expansion. |
| S | SinCos (↗ see page 112) | Computes the Sine and Cosine of the given angle (in Radians) using Series Expansion. |
| S | SinDeg (↗ see page 114) | Computes the Sine of the Angle, given the Angle in Degrees, using Sin(decimal). |
| S | Sqrt (↗ see page 115) | Computes the Decimal Square Root using Newton's Method. |
| S | Tan (↗ see page 117) | Computes the Tangent of the given angle (in Radians). |
| S | TanDeg (↗ see page 118) | Computes the Tangent of the Angle, given the Angle in Degrees, using Tan(decimal). |
| S | TenPow (↗ see page 120) | Computes the Specified Power of 10 using series Expansion. |
| S | TwoPow (↗ see page 121) | Computes the Specified Power of 2 using series Expansion. |

# 1.1.3.1 DecimalOps Methods

## 1.1.3.1.1 Acos Method

### 1.1.3.1.1.1 DecimalOps.Acos Method (decimal)

Computes the angle (in Radians) that resulted in this Cosine Value using Asin(decimal). Thus computing the inverse cosine or arccos.

**C++**

```
public: decimal Acos(
    decimal X
);
```

**C#**

```
public static decimal Acos(
    decimal X
);
```

**Visual Basic**

```
Public static Function Acos(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Cosine of the Angle |

**Returns**

Angle in Radians in the range of 0 to Pi inclusive.

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when abs (Value) > 1. |

### 1.1.3.1.1.2 DecimalOps.Acos Method (double)

Computes the angle (in Radians) that resulted in this Cosine Value using Asin(decimal). Thus computing the inverse cosine or arccos.

**C++**

```
public: decimal Acos(
    double X
);
```

**C#**

```
public static decimal Acos(
    double X
);
```

**Visual Basic**

```
Public static Function Acos(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Cosine of the Angle |

**Returns**

Angle in Radians in the range of 0 to Pi inclusive.

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when abs (Value) > 1. |

## 1.1.3.1.2 AcosDeg Method

### 1.1.3.1.2.1 DecimalOps.AcosDeg Method (decimal)

Computes the Inverse Cosine returning the Angle in Degrees. Uses Acos(decimal).

**C++**

```
public: decimal AcosDeg(
    decimal X
);
```

**C#**

```
public static decimal AcosDeg(
    decimal X
);
```

**Visual Basic**

```
Public static Function AcosDeg(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Cosine of the Angle |

**Returns**

Angle in Degrees

### 1.1.3.1.2.2 DecimalOps.AcosDeg Method (double)

Computes the Inverse Cosine returning the Angle in Degrees. Uses Acos(decimal).

**C++**

```
public: decimal AcosDeg(
    double X
);
```

**C#**

```
public static decimal AcosDeg(
    double X
);
```

**Visual Basic**

```
Public static Function AcosDeg(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Cosine of the Angle |

**Returns**

Angle in Degrees

## 1.1.3.1.3 AdjustAngleDeg Method

### 1.1.3.1.3.1 DecimalOps.AdjustAngleDeg Method (decimal)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Adjusted Angle in Degrees

### 1.1.3.1.3.2 DecimalOps.AdjustAngleDeg Method (decimal, bool)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive). if PosAngle is False, or between 0 (inclusive) and 360 (exclusive) if PosAngle is True. Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    decimal AngleInDegrees,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    decimal AngleInDegrees,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As decimal,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |
| bool PosAngle | When true the Angle returned is Positive, ie between 0 and 360, when false then the Angle is between -180 and 180. |

**Returns**

Adjusted Angle in Degrees

### 1.1.3.1.3.3 DecimalOps.AdjustAngleDeg Method (double)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Adjusted Angle in Degrees

## 1.1.3.1.3.4 DecimalOps.AdjustAngleDeg Method (double, bool)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) if PosAngle is False, or between 0 (inclusive) and 360 (exclusive) if PosAngle is True. Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    double AngleInDegrees,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    double AngleInDegrees,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As double,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |
| bool PosAngle | When True the Angle returned is Positive, ie between 0 and 360, when False then the Angle is between -180 and 180. |

**Returns**

Adjusted Angle in Degrees

## 1.1.3.1.3.5 DecimalOps.AdjustAngleDeg Method (int)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Adjusted Angle in Degrees

### 1.1.3.1.3.6 DecimalOps.AdjustAngleDeg Method (int, bool)

Adjust the Angle so that it lies between -180 (exclusive) and 180 (inclusive) if PosAngle is False, or between 0 (inclusive) and 360 (exclusive) if PosAngle is True. Works in Degrees.

**C++**

```
public: decimal AdjustAngleDeg(
    int AngleInDegrees,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleDeg(
    int AngleInDegrees,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleDeg(
    AngleInDegrees As Integer,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |
| bool PosAngle | When True the Angle returned is Positive, ie between 0 and 360, when False then the Angle is between -180 and 180. |

**Returns**

Adjusted Angle in Degrees

## 1.1.3.1.4 AdjustAngleRad Method

### 1.1.3.1.4.1 DecimalOps.AdjustAngleRad Method (decimal)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal AdjustAngleRad(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Adjusted Angle in Radians

### 1.1.3.1.4.2 DecimalOps.AdjustAngleRad Method (decimal, bool)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) if PosAngle is False, or between 0 (inclusive) and 2 * Pi (exclusive) if PosAngle is True. Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    decimal AngleInRadians,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleRad(
    decimal AngleInRadians,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As decimal,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |
| bool PosAngle | When true the Angle returned is Positive, ie between 0 and 2Pi, when false then the Angle is between -Pi and Pi. |

**Returns**

Adjusted Angle in Radians

### 1.1.3.1.4.3 DecimalOps.AdjustAngleRad Method (double)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    double AngleInRadians
);
```

**C#**

```
public static decimal AdjustAngleRad(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Adjusted Angle in Radians

### 1.1.3.1.4.4 DecimalOps.AdjustAngleRad Method (double, bool)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) if PosAngle is False, or between 0 (inclusive) and 2 * Pi (exclusive) if PosAngle is True. Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    double AngleInRadians,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleRad(
    double AngleInRadians,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As double,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |
| bool PosAngle | When true the Angle returned is Positive, ie between 0 and 2Pi, when false then the Angle is between -Pi and Pi. |

**Returns**

Adjusted Angle in Radians

### 1.1.3.1.4.5 DecimalOps.AdjustAngleRad Method (int)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    int AngleInRadians
);
```

**C#**

```
public static decimal AdjustAngleRad(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Adjusted Angle in Radians

### 1.1.3.1.4.6 DecimalOps.AdjustAngleRad Method (int, bool)

Adjust the Angle so that it lies between -Pi (exclusive) and Pi (inclusive) if PosAngle is False, or between 0 (inclusive) and 2 * Pi (exclusive) if PosAngle is True. Works in Radians.

**C++**

```
public: decimal AdjustAngleRad(
    int AngleInRadians,
    bool PosAngle
);
```

**C#**

```
public static decimal AdjustAngleRad(
    int AngleInRadians,
    bool PosAngle
);
```

**Visual Basic**

```
Public static Function AdjustAngleRad(
    AngleInRadians As Integer,
    PosAngle As bool
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |
| bool PosAngle | When true the Angle returned is Positive, ie between 0 and 2Pi, when false then the Angle is between -Pi and Pi. |

**Returns**

Adjusted Angle in Radians

## 1.1.3.1.5 Asin Method

### 1.1.3.1.5.1 DecimalOps.Asin Method (decimal)

Computes the angle (in Radians) that resulted in this Sine Value using Series Expansion. Thus computing the inverse sine or arcsin.

**C++**

```
public: decimal Asin(
    decimal X
);
```

**C#**

```
public static decimal Asin(
    decimal X
);
```

**Visual Basic**

```
Public static Function Asin(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Sine of the Angle |

**Returns**

Angle in Radians in the range of -Pi/2 to Pi/2 inclusive.

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when abs (Value) > 1. |

### 1.1.3.1.5.2 DecimalOps.Asin Method (double)

Computes the angle (in Radians) that resulted in this Sine Value using Series Expansion. Thus computing the inverse sine or arcsin.

**C++**

```
public: decimal Asin(
    double X
);
```

**C#**

```
public static decimal Asin(
    double X
);
```

**Visual Basic**

```
Public static Function Asin(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Sine of the Angle |

**Returns**

Angle in Radians in the range of -Pi/2 to Pi/2 inclusive.

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when abs (Value) > 1. |

## 1.1.3.1.6 AsinDeg Method

### 1.1.3.1.6.1 DecimalOps.AsinDeg Method (decimal)

Computes the Inverse Sine returning the Angle in Degrees. Uses Asin(decimal).

**C++**

```
public: decimal AsinDeg(
    decimal X
);
```

**C#**

```
public static decimal AsinDeg(
    decimal X
);
```

**Visual Basic**

```
Public static Function AsinDeg(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Sine of the Angle |

**Returns**

Angle in Degrees

### 1.1.3.1.6.2 DecimalOps.AsinDeg Method (double)

Computes the Inverse Sine returning the Angle in Degrees. Uses Asin(decimal).

**C++**

```
public: decimal AsinDeg(
    double X
);
```

**C#**

```
public static decimal AsinDeg(
    double X
);
```

**Visual Basic**

```
Public static Function AsinDeg(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Sine of the Angle |

**Returns**

Angle in Degrees

## 1.1.3.1.7 Atan Method

### 1.1.3.1.7.1 DecimalOps.Atan Method (decimal)

Computes the angle (in Radians) that resulted in this Tangent Value using Series Expansion. Thus computing the inverse tangent or arctan.

If you want to cover all quadrants then you may want to use Atan(decimal)

**C++**

```
public: decimal Atan(
    decimal X
);
```

**C#**

```
public static decimal Atan(
    decimal X
);
```

**Visual Basic**

```
Public static Function Atan(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Tangent of the Angle |

**Returns**

Angle in Radians in the range of -Pi/2 to Pi/2 inclusive.

## 1.1.3.1.7.2 DecimalOps.Atan Method (double)

Computes the angle (in Radians) that resulted in this Tangent Value using Series Expansion. Thus computing the inverse tangent or arctan.

If you want to cover all quadrants then you may want to use Atan(decimal)

**C++**

```
public: decimal Atan(
    double X
);
```

**C#**

```
public static decimal Atan(
    double X
);
```

**Visual Basic**

```
Public static Function Atan(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Tangent of the Angle |

**Returns**

Angle in Radians in the range of -Pi/2 to Pi/2 inclusive.

## 1.1.3.1.7.3 DecimalOps.Atan Method (int)

Computes the angle (in Radians) that resulted in this Tangent Value using Series Expansion. Thus computing the inverse tangent or arctan.

If you want to cover all quadrants then you may want to use Atan(decimal)

**C++**

```
public: decimal Atan(
    int X
);
```

**C#**

```
public static decimal Atan(
    int X
);
```

**Visual Basic**

```
Public static Function Atan(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Tangent of the Angle |

**Returns**

Angle in Radians in the range of -Pi/2 to Pi/2 inclusive.

## 1.1.3.1.8 Atan2 Method

### 1.1.3.1.8.1 DecimalOps.Atan2 Method (decimal, decimal)

Computes the angle (in Radians) that resulted in this Tangent value of Y / X using Atan(decimal). Thus computing the inverse tangent or arctan.

**C++**

```
public: decimal Atan2(
    decimal Y,
    decimal X
);
```

**C#**

```
public static decimal Atan2(
    decimal Y,
    decimal X
);
```

**Visual Basic**

```
Public static Function Atan2(
    Y As decimal,
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal Y | Y Co-ordinate |
| decimal X | X Co-ordinate |

**Returns**

Angle in Radians in the range of -Pi (exclusive) to Pi (inclusive).

### 1.1.3.1.8.2 DecimalOps.Atan2 Method (double, double)

Computes the angle (in Radians) that resulted in this Tangent value of Y / X using Atan(decimal). Thus computing the inverse tangent or arctan.

**C++**

```
public: decimal Atan2(
    double X,
    double Y
);
```

**C#**

```
public static decimal Atan2(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function Atan2(
    X As double,
    Y As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | X Co-ordinate |
| double Y | Y Co-ordinate |

**Returns**

Angle in Radians in the range of -Pi (exclusive) to Pi (inclusive).

### 1.1.3.1.8.3 DecimalOps.Atan2 Method (int, int)

Computes the angle (in Radians) that resulted in this Tangent value of Y / X using Atan(decimal). Thus computing the inverse tangent or arctan.

**C++**

```
public: decimal Atan2(
    int X,
    int Y
);
```

**C#**

```
public static decimal Atan2(
    int X,
    int Y
);
```

**Visual Basic**

```
Public static Function Atan2(
    X As Integer,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | X Co-ordinate |
| int Y | Y Co-ordinate |

**Returns**

Angle in Radians in the range of -Pi (exclusive) to Pi (inclusive).

## 1.1.3.1.9 Atan2Deg Method

### 1.1.3.1.9.1 DecimalOps.Atan2Deg Method (decimal, decimal)

Computes the Inverse Tangent, where the Tangent is Y /X, returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal Atan2Deg(
    decimal Y,
    decimal X
);
```

**C#**

```
public static decimal Atan2Deg(
    decimal Y,
    decimal X
);
```

**Visual Basic**

```
Public static Function Atan2Deg(
    Y As decimal,
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal Y | Y Co-ordinate |
| decimal X | X Co-ordinate |

**Returns**

Angle in Degrees

## 1.1.3.1.9.2 DecimalOps.Atan2Deg Method (double, double)

Computes the Inverse Tangent, where the Tangent is Y /X, returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal Atan2Deg(
    double Y,
    double X
);
```

**C#**

```
public static decimal Atan2Deg(
    double Y,
    double X
);
```

**Visual Basic**

```
Public static Function Atan2Deg(
    Y As double,
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double Y | Y Co-ordinate |
| double X | X Co-ordinate |

**Returns**

Angle in Degrees

## 1.1.3.1.9.3 DecimalOps.Atan2Deg Method (int, int)

Computes the Inverse Tangent, where the Tangent is Y /X, returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal Atan2Deg(
    int Y,
    int X
);
```

**C#**

```
public static decimal Atan2Deg(
    int Y,
    int X
);
```

**Visual Basic**

```
Public static Function Atan2Deg(
    Y As Integer,
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int Y | Y Co-ordinate |
| int X | X Co-ordinate |

**Returns**

Angle in Degrees

# 1.1.3.1.10 AtanDeg Method

## 1.1.3.1.10.1 DecimalOps.AtanDeg Method (decimal)

Computes the Inverse Tangent returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal AtanDeg(
    decimal X
);
```

**C#**

```
public static decimal AtanDeg(
    decimal X
);
```

**Visual Basic**

```
Public static Function AtanDeg(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Tangent of the Angle |

**Returns**

Angle in Degrees

## 1.1.3.1.10.2 DecimalOps.AtanDeg Method (double)

Computes the Inverse Tangent returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal AtanDeg(
    double X
);
```

**C#**

```
public static decimal AtanDeg(
    double X
);
```

**Visual Basic**

```
Public static Function AtanDeg(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Tangent of the Angle |

**Returns**

Angle in Degrees

### 1.1.3.1.10.3 DecimalOps.AtanDeg Method (int)

Computes the Inverse Tangent returning the Angle in Degrees. Uses Atan(decimal).

**C++**

```
public: decimal AtanDeg(
    int X
);
```

**C#**

```
public static decimal AtanDeg(
    int X
);
```

**Visual Basic**

```
Public static Function AtanDeg(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Tangent of the Angle |

**Returns**

Angle in Degrees

## 1.1.3.1.11 Ceiling Method

### 1.1.3.1.11.1 DecimalOps.Ceiling Method (decimal)

Returns the Integral Value that is greater than or equal to the specified number.

**C++**

```
public: decimal Ceiling(
    decimal X
);
```

**C#**

```
public static decimal Ceiling(
    decimal X
);
```

**Visual Basic**

```
Public static Function Ceiling(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Number to be processed. |

**Returns**

Integral Value that is greater than or equal to the specified number.

### 1.1.3.1.11.2 DecimalOps.Ceiling Method (double)

Returns the Integral Value that is greater than or equal to the specified number.

**C++**

```
public: decimal Ceiling(
    double X
);
```

**C#**

```
public static decimal Ceiling(
    double X
);
```

**Visual Basic**

```
Public static Function Ceiling(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Number to be processed. |

**Returns**

Integral Value that is greater than or equal to the specified number.

## 1.1.3.1.12 Cos Method

### 1.1.3.1.12.1 DecimalOps.Cos Method (decimal)

Computes the Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Cos(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Cos(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cos(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Cosine of the Angle

### 1.1.3.1.12.2 DecimalOps.Cos Method (double)

Computes the Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Cos(
    double AngleInRadians
);
```

**C#**

```
public static decimal Cos(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cos(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Cosine of the Angle

### 1.1.3.1.12.3 DecimalOps.Cos Method (int)

Computes the Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Cos(
    int AngleInRadians
);
```

**C#**

```
public static decimal Cos(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cos(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Cosine of the Angle

## 1.1.3.1.13 CosDeg Method

### 1.1.3.1.13.1 DecimalOps.CosDeg Method (decimal)

Computes the Cosine of the Angle, given the Angle in Degrees, using Cos(decimal).

**C++**

```
public: decimal CosDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal CosDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Cosine of the Angle

### 1.1.3.1.13.2 DecimalOps.CosDeg Method (double)

Computes the Cosine of the Angle, given the Angle in Degrees, using Cos(decimal).

**C++**

```
public: decimal CosDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal CosDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Cosine of the Angle

### 1.1.3.1.13.3 DecimalOps.CosDeg Method (int)

Computes the Cosine of the Angle, given the Angle in Degrees, using Cos(decimal).

**C++**

```
public: decimal CosDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal CosDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Cosine of the Angle

## 1.1.3.1.14 Cosec Method

### 1.1.3.1.14.1 DecimalOps.Cosec Method (decimal)

Computes the Cosecant of the given angle (in Radians) using .

**C++**

```
public: decimal Cosec(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Cosec(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cosec(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | `DivideByZeroException`. |

**Related Topics**

Sin(decimal)

## 1.1.3.1.14.2 DecimalOps.Cosec Method (double)

Computes the Cosecant of the given angle (in Radians) using .

**C++**

```
public: decimal Cosec(
    double AngleInRadians
);
```

**C#**

```
public static decimal Cosec(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cosec(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

**Related Topics**

Sin(decimal)

## 1.1.3.1.14.3 DecimalOps.Cosec Method (int)

Computes the Cosecant of the given angle (in Radians) using .

**C++**

```
public: decimal Cosec(
    int AngleInRadians
);
```

**C#**

```
public static decimal Cosec(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cosec(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

**Related Topics**

Sin(decimal)

# 1.1.3.1.15 CosecDeg Method

## 1.1.3.1.15.1 DecimalOps.CosecDeg Method (decimal)

Computes the Cosecant of the given angle (in Degrees) using .

**C++**

```
public: decimal CosecDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal CosecDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosecDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

**Related Topics**

Sin(decimal)

## 1.1.3.1.15.2 DecimalOps.CosecDeg Method (double)

Computes the Cosecant of the given angle (in Degrees) using .

**C++**

```
public: decimal CosecDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal CosecDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosecDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

**Related Topics**

Sin(decimal)

### 1.1.3.1.15.3 DecimalOps.CosecDeg Method (int)

Computes the Cosecant of the given angle (in Degrees) using .

**C++**

```
public: decimal CosecDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal CosecDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CosecDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Cosecant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

**Related Topics**

Sin(decimal)

## 1.1.3.1.16 Cot Method

## 1.1.3.1.16.1 DecimalOps.Cot Method (decimal)

Computes the Cotangent of the given angle (in Radians).

**C++**

```
public: decimal Cot(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Cot(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cot(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

## 1.1.3.1.16.2 DecimalOps.Cot Method (double)

Computes the Cotangent of the given angle (in Radians).

**C++**

```
public: decimal Cot(
    double AngleInRadians
);
```

**C#**

```
public static decimal Cot(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cot(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

### 1.1.3.1.16.3 DecimalOps.Cot Method (int)

Computes the Cotangent of the given angle (in Radians).

**C++**

```
public: decimal Cot(
    int AngleInRadians
);
```

**C#**

```
public static decimal Cot(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Cot(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

## 1.1.3.1.17 CotDeg Method

### 1.1.3.1.17.1 DecimalOps.CotDeg Method (decimal)

Computes the Cotangent of the given angle (in Degrees).

**C++**

```
public: decimal CotDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal CotDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CotDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

### 1.1.3.1.17.2 DecimalOps.CotDeg Method (double)

Computes the Cotangent of the given angle (in Degrees).

**C++**

```
public: decimal CotDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal CotDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CotDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

### 1.1.3.1.17.3 DecimalOps.CotDeg Method (int)

Computes the Cotangent of the given angle (in Degrees).

**C++**

```
public: decimal CotDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal CotDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function CotDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Cotangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the 0 or Pi Axes. |

## 1.1.3.1.18 Deg2Dms Method

### 1.1.3.1.18.1 DecimalOps.Deg2Dms Method (decimal, out int, out int, out decimal, out int)

Converts a Decimal Degree into Degrees/Minutes/Seconds. Returned values are positive with Sign containing the System.Math.Sign(decimal) of the original value.

**C++**

```
public: void Deg2Dms(
    decimal decDegrees,
    out int varDegrees,
    out int varMinutes,
    out decimal varSeconds,
    out int varSign
);
```

**C#**

```
public static void Deg2Dms(
    decimal decDegrees,
    out int varDegrees,
    out int varMinutes,
    out decimal varSeconds,
    out int varSign
);
```

**Visual Basic**

```
Public static Function Deg2Dms(
    decDegrees As decimal,
    varDegrees As out int,
    varMinutes As out int,
    varSeconds As out decimal,
    varSign As out int
) As void
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal decDegrees | Degrees in decimal format. Can be Negative. |
| out int varDegrees | Whole Degrees, >= 0 |
| out int varMinutes | Whole Minutes, >= 0 |
| out decimal varSeconds | Seconds, including fractional portion, >= 0 |
| out int varSign | 1 if > 0, 0 if = 0, -1 if < 0 |

### 1.1.3.1.18.2 DecimalOps.Deg2Dms Method (double, out int, out int, out decimal, out int)

Converts a Decimal Degree into Degrees/Minutes/Seconds. Returned values are positive with Sign containing the System.Math.Sign(decimal) of the original value.

**C++**

```
public: void Deg2Dms(
    double decDegrees,
    out int varDegrees,
    out int varMinutes,
    out decimal varSeconds,
    out int varSign
);
```

**C#**

```
public static void Deg2Dms(
    double decDegrees,
    out int varDegrees,
    out int varMinutes,
    out decimal varSeconds,
    out int varSign
);
```

**Visual Basic**

```
Public static Function Deg2Dms(
    decDegrees As double,
    varDegrees As out int,
    varMinutes As out int,
    varSeconds As out decimal,
    varSign As out int
) As void
```

**Parameters**

| Parameters | Description |
|---|---|
| double decDegrees | Degrees in decimal format. Can be Negative. |
| out int varDegrees | Whole Degrees, >= 0 |
| out int varMinutes | Whole Minutes, >= 0 |
| out decimal varSeconds | Seconds, including fractional portion, >= 0 |
| out int varSign | 1 if > 0, 0 if = 0, -1 if < 0 |

## 1.1.3.1.19 Deg2Grad Method

### 1.1.3.1.19.1 DecimalOps.Deg2Grad Method (decimal)

Converts Degrees into Grads.

**C++**

```
public: decimal Deg2Grad(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Grad(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Grad(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Angle in Grads

### 1.1.3.1.19.2 DecimalOps.Deg2Grad Method (double)

Converts Degrees into Grads.

**C++**

```
public: decimal Deg2Grad(
    double AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Grad(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Grad(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Angle in Grads

### 1.1.3.1.19.3 DecimalOps.Deg2Grad Method (int)

Converts Degrees into Grads.

**C++**

```
public: decimal Deg2Grad(
    int AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Grad(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Grad(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Angle in Grads

## 1.1.3.1.20 Deg2Rad Method

### 1.1.3.1.20.1 DecimalOps.Deg2Rad Method (decimal)

Converts Degrees into Radians.

**C++**

```
public: decimal Deg2Rad(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Rad(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Rad(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Angle In Radians

### 1.1.3.1.20.2 DecimalOps.Deg2Rad Method (double)

Converts Degrees into Radians.

**C++**

```
public: decimal Deg2Rad(
    double AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Rad(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Rad(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Angle In Radians

### 1.1.3.1.20.3 DecimalOps.Deg2Rad Method (int)

Converts Degrees into Radians.

**C++**

```
public: decimal Deg2Rad(
    int AngleInDegrees
);
```

**C#**

```
public static decimal Deg2Rad(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function Deg2Rad(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Angle In Radians

## 1.1.3.1.21 Dms2Deg Method

### 1.1.3.1.21.1 DecimalOps.Dms2Deg Method (decimal, decimal, decimal)

Converts Degrees/Minutes/Seconds into Decimal Degrees.

**C++**

```
public: decimal Dms2Deg(
    decimal varDegrees,
    decimal varMinutes,
    decimal varSeconds
);
```

**C#**

```
public static decimal Dms2Deg(
    decimal varDegrees,
    decimal varMinutes,
    decimal varSeconds
);
```

**Visual Basic**

```
Public static Function Dms2Deg(
    varDegrees As decimal,
    varMinutes As decimal,
    varSeconds As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal varDegrees | Number of Degrees. |
| decimal varMinutes | Number of Minutes. |
| decimal varSeconds | Number of Seconds. |

### 1.1.3.1.21.2 DecimalOps.Dms2Deg Method (double, double, double)

Converts Degrees/Minutes/Seconds into Decimal Degrees.

**C++**

```
public: decimal Dms2Deg(
    double varDegrees,
    double varMinutes,
    double varSeconds
);
```

**C#**

```
public static decimal Dms2Deg(
    double varDegrees,
    double varMinutes,
    double varSeconds
);
```

**Visual Basic**

```
Public static Function Dms2Deg(
    varDegrees As double,
    varMinutes As double,
    varSeconds As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double varDegrees | Number of Degrees. |
| double varMinutes | Number of Minutes. |
| double varSeconds | Number of Seconds. |

### 1.1.3.1.21.3 DecimalOps.Dms2Deg Method (int, int, int)

Converts Degrees/Minutes/Seconds into Decimal Degrees.

**C++**

```
public: decimal Dms2Deg(
    int varDegrees,
    int varMinutes,
    int varSeconds
);
```

**C#**

```
public static decimal Dms2Deg(
    int varDegrees,
    int varMinutes,
    int varSeconds
);
```

**Visual Basic**

```
Public static Function Dms2Deg(
    varDegrees As Integer,
    varMinutes As Integer,
    varSeconds As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int varDegrees | Number of Degrees. |
| int varMinutes | Number of Minutes. |
| int varSeconds | Number of Seconds. |

## 1.1.3.1.22 Exp Method

### 1.1.3.1.22.1 DecimalOps.Exp Method (decimal)

Computes the Exponential Function using Series Expansion.

**C++**

```
public: decimal Exp(
    decimal X
);
```

**C#**

```
public static decimal Exp(
    decimal X
);
```

**Visual Basic**

```
Public static Function Exp(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Power to raise the Natural Constant, e, to |

**Returns**

e^X, Natural Constant raised to the power X

## 1.1.3.1.22.2 DecimalOps.Exp Method (double)

Computes the Exponential Function using Series Expansion.

**C++**

```
public: decimal Exp(
    double X
);
```

**C#**

```
public static decimal Exp(
    double X
);
```

**Visual Basic**

```
Public static Function Exp(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Power to raise the Natural Constant, e, to |

**Returns**

e^X, Natural Constant raised to the power X

## 1.1.3.1.22.3 DecimalOps.Exp Method (int)

Computes the Exponential Function using Series Expansion.

**C++**

```
public: decimal Exp(
    int X
);
```

**C#**

```
public static decimal Exp(
    int X
);
```

**Visual Basic**

```
Public static Function Exp(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Power to raise the Natural Constant, e, to |

**Returns**

e^X, Natural Constant raised to the power X

## 1.1.3.1.23 FractionPart Method

### 1.1.3.1.23.1 DecimalOps.FractionPart Method (decimal)

Returns the Fractional portion of the value supplied.

**C++**

```
public: decimal FractionPart(
    decimal X
);
```

**C#**

```
public static decimal FractionPart(
    decimal X
);
```

**Visual Basic**

```
Public static Function FractionPart(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Number to be processed. |

**Returns**

Fractional portion of the value supplied

### 1.1.3.1.23.2 DecimalOps.FractionPart Method (double)

Returns the Fractional portion of the value supplied.

**C++**

```
public: decimal FractionPart(
    double X
);
```

**C#**

```
public static decimal FractionPart(
    double X
);
```

**Visual Basic**

```
Public static Function FractionPart(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Number to be processed. |

**Returns**

Fractional portion of the value supplied

# 1.1.3.1.24 Grad2Deg Method

## 1.1.3.1.24.1 DecimalOps.Grad2Deg Method (decimal)

Converts Grads into Degrees.

**C++**

```
public: decimal Grad2Deg(
    decimal AngleInGrads
);
```

**C#**

```
public static decimal Grad2Deg(
    decimal AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Deg(
    AngleInGrads As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInGrads | Angle in Grads |

**Returns**

Angle In Degrees

## 1.1.3.1.24.2 DecimalOps.Grad2Deg Method (double)

Converts Grads into Degrees.

**C++**

```
public: decimal Grad2Deg(
    double AngleInGrads
);
```

**C#**

```
public static decimal Grad2Deg(
    double AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Deg(
    AngleInGrads As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInGrads | Angle in Grads |

**Returns**

Angle In Degrees

### 1.1.3.1.24.3 DecimalOps.Grad2Deg Method (int)

Converts Grads into Degrees.

**C++**

```
public: decimal Grad2Deg(
    int AngleInGrads
);
```

**C#**

```
public static decimal Grad2Deg(
    int AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Deg(
    AngleInGrads As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInGrads | Angle in Grads |

**Returns**

Angle In Degrees

## 1.1.3.1.25 Grad2Rad Method

### 1.1.3.1.25.1 DecimalOps.Grad2Rad Method (decimal)

Converts Grads into Radians.

**C++**

```
public: decimal Grad2Rad(
    decimal AngleInGrads
);
```

**C#**

```
public static decimal Grad2Rad(
    decimal AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Rad(
    AngleInGrads As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInGrads | Angle in Grads |

**Returns**

Angle In Radians

### 1.1.3.1.25.2 DecimalOps.Grad2Rad Method (double)

Converts Grads into Radians.

**C++**

```
public: decimal Grad2Rad(
    double AngleInGrads
);
```

**C#**

```
public static decimal Grad2Rad(
    double AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Rad(
    AngleInGrads As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInGrads | Angle in Grads |

**Returns**

Angle In Radians

### 1.1.3.1.25.3 DecimalOps.Grad2Rad Method (int)

Converts Grads into Radians.

**C++**

```
public: decimal Grad2Rad(
    int AngleInGrads
);
```

**C#**

```
public static decimal Grad2Rad(
    int AngleInGrads
);
```

**Visual Basic**

```
Public static Function Grad2Rad(
    AngleInGrads As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInGrads | Angle in Grads |

**Returns**

Angle In Radians

## 1.1.3.1.26 IntPart Method

### 1.1.3.1.26.1 DecimalOps.IntPart Method (decimal)

Returns the Integral portion of the value supplied.

**C++**

```
public: decimal IntPart(
    decimal X
);
```

**C#**

```
public static decimal IntPart(
    decimal X
);
```

**Visual Basic**

```
Public static Function IntPart(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Number to be processed. |

**Returns**

Integral portion of the value supplied

### 1.1.3.1.26.2 DecimalOps.IntPart Method (double)

Returns the Integral portion of the value supplied.

**C++**

```
public: decimal IntPart(
    double X
);
```

**C#**

```
public static decimal IntPart(
    double X
);
```

**Visual Basic**

```
Public static Function IntPart(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Number to be processed. |

**Returns**

Integral portion of the value supplied

## 1.1.3.1.27 IntPow Method

### 1.1.3.1.27.1 DecimalOps.IntPow Method (decimal, int)

Computes the value of a Decimal raised to an Integer Power.

**C++**

```
public: decimal IntPow(
    decimal X,
    int Y
);
```

**C#**

```
public static decimal IntPow(
    decimal X,
    int Y
);
```

**Visual Basic**

```
Public static Function IntPow(
    X As decimal,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | The Value to be raised to the specified power. |
| int Y | The Power to raise the specified value to. |

**Returns**

X ^ Y, ie X raised to the power of Y.

## 1.1.3.1.27.2 DecimalOps.IntPow Method (double, int)

Computes the value of a Decimal raised to an Integer Power.

**C++**

```
public: decimal IntPow(
    double X,
    int Y
);
```

**C#**

```
public static decimal IntPow(
    double X,
    int Y
);
```

**Visual Basic**

```
Public static Function IntPow(
    X As double,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | The Value to be raised to the specified power. |
| int Y | The Power to raise the specified value to. |

**Returns**

X ^ Y, ie X raised to the power of Y.

## 1.1.3.1.27.3 DecimalOps.IntPow Method (int, int)

Computes the value of a Decimal raised to an Integer Power.

**C++**

```
public: decimal IntPow(
    int X,
    int Y
);
```

**C#**

```
public static decimal IntPow(
    int X,
    int Y
);
```

**Visual Basic**

```
Public static Function IntPow(
    X As Integer,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | The Value to be raised to the specified power. |
| int Y | The Power to raise the specified value to. |

**Returns**

X ^ Y, ie X raised to the power of Y.

## 1.1.3.1.28 Log Method

### 1.1.3.1.28.1 DecimalOps.Log Method (decimal)

Computes the Natural Logarithm using Series Expansion.

**C++**

```
public: decimal Log(
    decimal X
);
```

**C#**

```
public static decimal Log(
    decimal X
);
```

**Visual Basic**

```
Public static Function Log(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | We desire the natural logarithm of this Value. Must be > 0 |

**Returns**

Natural Logarithm of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

### 1.1.3.1.28.2 DecimalOps.Log Method (double)

Computes the Natural Logarithm using Series Expansion.

**C++**

```
public: decimal Log(
    double X
);
```

**C#**

```
public static decimal Log(
    double X
);
```

**Visual Basic**

```
Public static Function Log(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | We desire the natural logarithm of this Value. Must be > 0 |

**Returns**

Natural Logarithm of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

### 1.1.3.1.28.3 DecimalOps.Log Method (int)

Computes the Natural Logarithm using Series Expansion.

**C++**

```
public: decimal Log(
    int X
);
```

**C#**

```
public static decimal Log(
    int X
);
```

**Visual Basic**

```
Public static Function Log(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | We desire the natural logarithm of this Value. Must be > 0 |

**Returns**

Natural Logarithm of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

## 1.1.3.1.29 Log10 Method

### 1.1.3.1.29.1 DecimalOps.Log10 Method (decimal)

Computes the Logarithm to Base 10 using Series Expansion.

**C++**

```
public: decimal Log10(
    decimal X
);
```

**C#**

```
public static decimal Log10(
    decimal X
);
```

**Visual Basic**

```
Public static Function Log10(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | We desire the logarithm to base 10 of this Value. Must be > 0 |

**Returns**

Logarithm to base 10 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

### 1.1.3.1.29.2 DecimalOps.Log10 Method (double)

Computes the Logarithm to Base 10 using Series Expansion.

**C++**

```
public: decimal Log10(
    double X
);
```

**C#**

```
public static decimal Log10(
    double X
);
```

**Visual Basic**

```
Public static Function Log10(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | We desire the logarithm to base 10 of this Value.Must be > 0 |

**Returns**

Logarithm to base 10 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

### 1.1.3.1.29.3 DecimalOps.Log10 Method (int)

Computes the Logarithm to Base 10 using Series Expansion.

**C++**

```
public: decimal Log10(
    int X
);
```

**C#**

```
public static decimal Log10(
    int X
);
```

**Visual Basic**

```
Public static Function Log10(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | We desire the logarithm to base 10 of this Value. Must be > 0 |

**Returns**

Logarithm to base 10 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

## 1.1.3.1.30 Log2 Method

### 1.1.3.1.30.1 DecimalOps.Log2 Method (decimal)

Computes the Logarithm to Base 2 using Series Expansion.

**C++**

```
public: decimal Log2(
    decimal X
);
```

**C#**

```
public static decimal Log2(
    decimal X
);
```

**Visual Basic**

```
Public static Function Log2(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | We desire the logarithm to base 2 of this Value. Must be > 0 |

**Returns**

Logarithm to base 2 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

## 1.1.3.1.30.2 DecimalOps.Log2 Method (double)

Computes the Logarithm to Base 2 using Series Expansion.

**C++**

```
public: decimal Log2(
    double X
);
```

**C#**

```
public static decimal Log2(
    double X
);
```

**Visual Basic**

```
Public static Function Log2(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | We desire the logarithm to base 2 of this Value. Must be > 0 |

**Returns**

Logarithm to base 2 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

## 1.1.3.1.30.3 DecimalOps.Log2 Method (int)

Computes the Logarithm to Base 2 using Series Expansion.

**C++**

```
public: decimal Log2(
    int X
);
```

**C#**

```
public static decimal Log2(
    int X
);
```

**Visual Basic**

```
Public static Function Log2(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | We desire the logarithm to base 2 of this Value. Must be > 0 |

**Returns**

Logarithm to base 2 of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a non-positive value is passed. |

## 1.1.3.1.31 NthRoot Method

### 1.1.3.1.31.1 DecimalOps.NthRoot Method (decimal, int)

Computes the Decimal Nth Root using Newton's Method.

**C++**

```
public: decimal NthRoot(
    decimal X,
    int N
);
```

**C#**

```
public static decimal NthRoot(
    decimal X,
    int N
);
```

**Visual Basic**

```
Public static Function NthRoot(
    X As decimal,
    N As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | We desire the Nth root of this Value. Must be >= 0 |
| int N | The Level of Root - ie Nth Root. |

**Returns**

Nth Root of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed or n < 2. `ArithmeticException`. |

### 1.1.3.1.31.2 DecimalOps.NthRoot Method (double, int)

Computes the Decimal Nth Root using Newton's Method.

**C++**

```
public: decimal NthRoot(
    double X,
    int N
);
```

**C#**

```
public static decimal NthRoot(
    double X,
    int N
);
```

**Visual Basic**

```
Public static Function NthRoot(
    X As double,
    N As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | We desire the Nth root of this Value. Must be >= 0 |
| int N | The Level of Root - ie Nth Root. |

**Returns**

Nth Root of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed or n < 2. |

### 1.1.3.1.31.3 DecimalOps.NthRoot Method (int, int)

Computes the Decimal Nth Root using Newton's Method.

**C++**

```
public: decimal NthRoot(
    int X,
    int N
);
```

**C#**

```
public static decimal NthRoot(
    int X,
    int N
);
```

**Visual Basic**

```
Public static Function NthRoot(
    X As Integer,
    N As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | We desire the Nth root of this Value. Must be >= 0 |
| int N | The Level of Root - ie Nth Root. |

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed or n < 2. |

## 1.1.3.1.32 Pow Method

### 1.1.3.1.32.1 DecimalOps.Pow Method (decimal, decimal)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    decimal X,
    decimal Y
);
```

**C#**

```
public static decimal Pow(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As decimal,
    Y As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to raise |
| decimal Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.2 DecimalOps.Pow Method (decimal, double)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    decimal X,
    double Y
);
```

**C#**

```
public static decimal Pow(
    decimal X,
    double Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As decimal,
    Y As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to raise |
| double Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.3 DecimalOps.Pow Method (decimal, int)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    decimal X,
    int Y
);
```

**C#**

```
public static decimal Pow(
    decimal X,
    int Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As decimal,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to raise |
| int Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.4 DecimalOps.Pow Method (double, decimal)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    double X,
    decimal Y
);
```

**C#**

```
public static decimal Pow(
    double X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As double,
    Y As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to raise |

| | |
|---|---|
| decimal Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

## 1.1.3.1.32.5 DecimalOps.Pow Method (double, double)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    double X,
    double Y
);
```

**C#**

```
public static decimal Pow(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As double,
    Y As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to raise |
| double Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

## 1.1.3.1.32.6 DecimalOps.Pow Method (double, int)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    double X,
    int Y
);
```

**C#**

```
public static decimal Pow(
    double X,
    int Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As double,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to raise |
| int Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.7 DecimalOps.Pow Method (int, decimal)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    int X,
    decimal Y
);
```

**C#**

```
public static decimal Pow(
    int X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As Integer,
    Y As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Value to raise |
| decimal Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.8 DecimalOps.Pow Method (int, double)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    int X,
    double Y
);
```

**C#**

```
public static decimal Pow(
    int X,
    double Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As Integer,
    Y As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Value to raise |
| double Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

### 1.1.3.1.32.9 DecimalOps.Pow Method (int, int)

Computes the First Value raised to the power of the Second Value using series Expansion.

**C++**

```
public: decimal Pow(
    int X,
    int Y
);
```

**C#**

```
public static decimal Pow(
    int X,
    int Y
);
```

**Visual Basic**

```
Public static Function Pow(
    X As Integer,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Value to raise |
| int Y | Power to use |

**Returns**

X ^ Y, X raised to the power Y

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when X is Zero and a Negative Power, or a Negative X is raised to a non-integral power. |

# 1.1.3.1.33 Rad2Deg Method

## 1.1.3.1.33.1 DecimalOps.Rad2Deg Method (decimal)

Converts Radians into Degrees.

**C++**

```
public: decimal Rad2Deg(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Rad2Deg(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Deg(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Angle in Degrees

## 1.1.3.1.33.2 DecimalOps.Rad2Deg Method (double)

Converts Radians into Degrees.

**C++**

```
public: decimal Rad2Deg(
    double AngleInRadians
);
```

**C#**

```
public static decimal Rad2Deg(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Deg(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Angle in Degrees

### 1.1.3.1.33.3 DecimalOps.Rad2Deg Method (int)

Converts Radians into Degrees.

**C++**

```
public: decimal Rad2Deg(
    int AngleInRadians
);
```

**C#**

```
public static decimal Rad2Deg(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Deg(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Angle in Degrees

## 1.1.3.1.34 Rad2Grad Method

### 1.1.3.1.34.1 DecimalOps.Rad2Grad Method (decimal)

Converts Radians into Grads.

**C++**

```
public: decimal Rad2Grad(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Rad2Grad(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Grad(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Angle in Grads

### 1.1.3.1.34.2 DecimalOps.Rad2Grad Method (double)

Converts Radians into Grads.

**C++**

```
public: decimal Rad2Grad(
    double AngleInRadians
);
```

**C#**

```
public static decimal Rad2Grad(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Grad(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Angle in Grads

### 1.1.3.1.34.3 DecimalOps.Rad2Grad Method (int)

Converts Radians into Grads.

**C++**

```
public: decimal Rad2Grad(
    int AngleInRadians
);
```

**C#**

```
public static decimal Rad2Grad(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Rad2Grad(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Angle in Grads

## 1.1.3.1.35 Sec Method

### 1.1.3.1.35.1 DecimalOps.Sec Method (decimal)

Computes the Secant of the given angle (in Radians) using .

**C++**

```
public: decimal Sec(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Sec(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sec(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

### 1.1.3.1.35.2 DecimalOps.Sec Method (double)

Computes the Secant of the given angle (in Radians) using .

**C++**

```
public: decimal Sec(
    double AngleInRadians
);
```

**C#**

```
public static decimal Sec(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sec(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

### 1.1.3.1.35.3 DecimalOps.Sec Method (int)

Computes the Secant of the given angle (in Radians) using .

**C++**

```
public: decimal Sec(
    int AngleInRadians
);
```

**C#**

```
public static decimal Sec(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sec(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

## 1.1.3.1.36 SecDeg Method

### 1.1.3.1.36.1 DecimalOps.SecDeg Method (decimal)

Computes the Secant of the given angle (in Degrees) using .

**C++**

```
public: decimal SecDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal SecDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SecDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

### 1.1.3.1.36.2 DecimalOps.SecDeg Method (double)

Computes the Secant of the given angle (in Degrees) using .

**C++**

```
public: decimal SecDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal SecDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SecDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

### 1.1.3.1.36.3 DecimalOps.SecDeg Method (int)

Computes the Secant of the given angle (in Degrees) using .

**C++**

```
public: decimal SecDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal SecDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SecDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Secant of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

**Related Topics**

Cos(decimal)

## 1.1.3.1.37 SignXY Method

### 1.1.3.1.37.1 DecimalOps.SignXY Method (decimal, decimal)

Returns the FORTRAN type SIGN of the Values - basically it returns a value with the Magnitude of First Value and the Sign of the Second Value.

**C++**

```
public: decimal SignXY(
    decimal X,
    decimal Y
);
```

**C#**

```
public static decimal SignXY(
    decimal X,
    decimal Y
);
```

**Visual Basic**

```
Public static Function SignXY(
    X As decimal,
    Y As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value whose magnitude to process. |
| decimal Y | Value whose sign to process. |

**Returns**

magnitude of X with sign of Y.

**Remarks**

If Y is negative then Returns -Abs(X) else Returns Abs(X)

### 1.1.3.1.37.2 DecimalOps.SignXY Method (double, double)

Returns the FORTRAN type SIGN of the Values - basically it returns a value with the Magnitude of First Value and the Sign of the Second Value.

**C++**

```
public: decimal SignXY(
    double X,
    double Y
);
```

**C#**

```
public static decimal SignXY(
    double X,
    double Y
);
```

**Visual Basic**

```
Public static Function SignXY(
    X As double,
    Y As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value whose magnitude to process. |
| double Y | Value whose sign to process. |

**Returns**

magnitude of X with sign of Y.

**Remarks**

If Y is negative then Returns -Abs(X) else Returns Abs(X)

### 1.1.3.1.37.3 DecimalOps.SignXY Method (int, int)

Returns the FORTRAN type SIGN of the Values - basically it returns a value with the Magnitude of First Value and the Sign of the Second Value.

**C++**

```
public: decimal SignXY(
    int X,
    int Y
);
```

**C#**

```
public static decimal SignXY(
    int X,
    int Y
);
```

**Visual Basic**

```
Public static Function SignXY(
    X As Integer,
    Y As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Value whose magnitude to process. |

| | |
|---|---|
| int Y | Value whose sign to process. |

**Returns**

magnitude of X with sign of Y.

**Remarks**

If Y is negative then Returns -Abs(X) else Returns Abs(X)

## 1.1.3.1.38 Sin Method

### 1.1.3.1.38.1 DecimalOps.Sin Method (decimal)

Computes the Sine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Sin(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Sin(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sin(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Sine of the Angle

### 1.1.3.1.38.2 DecimalOps.Sin Method (double)

Computes the Sine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Sin(
    double AngleInRadians
);
```

**C#**

```
public static decimal Sin(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sin(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Sine of the Angle

### 1.1.3.1.38.3 DecimalOps.Sin Method (int)

Computes the Sine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: decimal Sin(
    int AngleInRadians
);
```

**C#**

```
public static decimal Sin(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Sin(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Sine of the Angle

## 1.1.3.1.39 SinCos Method

### 1.1.3.1.39.1 DecimalOps.SinCos Method (decimal, out decimal, out decimal)

Computes the Sine and Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: void SinCos(
    decimal AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**C#**

```
public static void SinCos(
    decimal AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**Visual Basic**

```
Public static Function SinCos(
    AngleInRadians As decimal,
    SinX As out decimal,
    CosX As out decimal
) As void
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

| | |
|---|---|
| out decimal SinX | Sine of the Angle |
| out decimal CosX | Cosine of the Angle |

## 1.1.3.1.39.2 DecimalOps.SinCos Method (double, out decimal, out decimal)

Computes the Sine and Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: void SinCos(
    double AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**C#**

```
public static void SinCos(
    double AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**Visual Basic**

```
Public static Function SinCos(
    AngleInRadians As double,
    SinX As out decimal,
    CosX As out decimal
) As void
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |
| out decimal SinX | Sine of the Angle |
| out decimal CosX | Cosine of the Angle |

## 1.1.3.1.39.3 DecimalOps.SinCos Method (int, out decimal, out decimal)

Computes the Sine and Cosine of the given angle (in Radians) using Series Expansion.

**C++**

```
public: void SinCos(
    int AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**C#**

```
public static void SinCos(
    int AngleInRadians,
    out decimal SinX,
    out decimal CosX
);
```

**Visual Basic**

```
Public static Function SinCos(
    AngleInRadians As Integer,
    SinX As out decimal,
    CosX As out decimal
) As void
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |
| out decimal SinX | Sine of the Angle |
| out decimal CosX | Cosine of the Angle |

# 1.1.3.1.40 SinDeg Method

## 1.1.3.1.40.1 DecimalOps.SinDeg Method (decimal)

Computes the Sine of the Angle, given the Angle in Degrees, using Sin(decimal).

**C++**

```
public: decimal SinDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal SinDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SinDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Sine of the Angle

## 1.1.3.1.40.2 DecimalOps.SinDeg Method (double)

Computes the Sine of the Angle, given the Angle in Degrees, using Sin(decimal).

**C++**

```
public: decimal SinDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal SinDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SinDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Sine of the Angle

### 1.1.3.1.40.3 DecimalOps.SinDeg Method (int)

Computes the Sine of the Angle, given the Angle in Degrees, using Sin(decimal).

**C++**

```
public: decimal SinDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal SinDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function SinDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Sine of the Angle

## 1.1.3.1.41 Sqrt Method

### 1.1.3.1.41.1 DecimalOps.Sqrt Method (decimal)

Computes the Decimal Square Root using Newton's Method.

**C++**

```
public: decimal Sqrt(
    decimal X
);
```

**C#**

```
public static decimal Sqrt(
    decimal X
);
```

**Visual Basic**

```
Public static Function Sqrt(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | We desire the square root of this Value. Must be >= 0 |

**Returns**

Square Root of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed. `ArithmeticException`. |

## 1.1.3.1.41.2 DecimalOps.Sqrt Method (double)

Computes the Decimal Square Root using Newton's Method.

**C++**

```
public: decimal Sqrt(
    double X
);
```

**C#**

```
public static decimal Sqrt(
    double X
);
```

**Visual Basic**

```
Public static Function Sqrt(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | We desire the square root of this Value. Must be >= 0 |

**Returns**

Square Root of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed. |

## 1.1.3.1.41.3 DecimalOps.Sqrt Method (int)

Computes the Decimal Square Root using Newton's Method.

**C++**

```
public: decimal Sqrt(
    int X
);
```

**C#**

```
public static decimal Sqrt(
    int X
);
```

**Visual Basic**

```
Public static Function Sqrt(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | We desire the square root of this Value. Must be >= 0 |

**Returns**

Square Root of given Value

**Exceptions**

| Exceptions | Description |
|---|---|
| ArithmeticException | Thrown when a negative value is passed. |

## 1.1.3.1.42 Tan Method

### 1.1.3.1.42.1 DecimalOps.Tan Method (decimal)

Computes the Tangent of the given angle (in Radians).

**C++**

```
public: decimal Tan(
    decimal AngleInRadians
);
```

**C#**

```
public static decimal Tan(
    decimal AngleInRadians
);
```

**Visual Basic**

```
Public static Function Tan(
    AngleInRadians As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInRadians | Angle in Radians |

**Returns**

Tangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

### 1.1.3.1.42.2 DecimalOps.Tan Method (double)

Computes the Tangent of the given angle (in Radians).

**C++**

```
public: decimal Tan(
    double AngleInRadians
);
```

**C#**

```
public static decimal Tan(
    double AngleInRadians
);
```

**Visual Basic**

```
Public static Function Tan(
    AngleInRadians As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInRadians | Angle in Radians |

**Returns**

Tangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

### 1.1.3.1.42.3 DecimalOps.Tan Method (int)

Computes the Tangent of the given angle (in Radians).

**C++**

```
public: decimal Tan(
    int AngleInRadians
);
```

**C#**

```
public static decimal Tan(
    int AngleInRadians
);
```

**Visual Basic**

```
Public static Function Tan(
    AngleInRadians As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInRadians | Angle in Radians |

**Returns**

Tangent of the Angle

**Exceptions**

| Exceptions | Description |
|---|---|
| DivideByZeroException | Thrown when the angle falls on the Pi/2 or 3*Pi/2 Axes. |

## 1.1.3.1.43 TanDeg Method

### 1.1.3.1.43.1 DecimalOps.TanDeg Method (decimal)

Computes the Tangent of the Angle, given the Angle in Degrees, using Tan(decimal).

**C++**

```
public: decimal TanDeg(
    decimal AngleInDegrees
);
```

**C#**

```
public static decimal TanDeg(
    decimal AngleInDegrees
);
```

**Visual Basic**

```
Public static Function TanDeg(
    AngleInDegrees As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal AngleInDegrees | Angle in Degrees |

**Returns**

Tangent of the Angle

### 1.1.3.1.43.2 DecimalOps.TanDeg Method (double)

Computes the Tangent of the Angle, given the Angle in Degrees, using Tan(decimal).

**C++**

```
public: decimal TanDeg(
    double AngleInDegrees
);
```

**C#**

```
public static decimal TanDeg(
    double AngleInDegrees
);
```

**Visual Basic**

```
Public static Function TanDeg(
    AngleInDegrees As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double AngleInDegrees | Angle in Degrees |

**Returns**

Tangent of the Angle

### 1.1.3.1.43.3 DecimalOps.TanDeg Method (int)

Computes the Tangent of the Angle, given the Angle in Degrees, using Tan(decimal).

**C++**

```
public: decimal TanDeg(
    int AngleInDegrees
);
```

**C#**

```
public static decimal TanDeg(
    int AngleInDegrees
);
```

**Visual Basic**

```
Public static Function TanDeg(
    AngleInDegrees As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int AngleInDegrees | Angle in Degrees |

**Returns**

Tangent of the Angle

## 1.1.3.1.44 TenPow Method

### 1.1.3.1.44.1 DecimalOps.TenPow Method (decimal)

Computes the Specified Power of 10 using series Expansion.

**C++**

```
public: decimal TenPow(
    decimal X
);
```

**C#**

```
public static decimal TenPow(
    decimal X
);
```

**Visual Basic**

```
Public static Function TenPow(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Power to raise 10 to |

**Returns**

10 ^ X, 10 raised to the power X

### 1.1.3.1.44.2 DecimalOps.TenPow Method (double)

Computes the Specified Power of 10 using series Expansion.

**C++**

```
public: decimal TenPow(
    double X
);
```

**C#**

```
public static decimal TenPow(
    double X
);
```

**Visual Basic**

```
Public static Function TenPow(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Power to raise 10 to |

**Returns**

10 ^ X, 10 raised to the power X

### 1.1.3.1.44.3 DecimalOps.TenPow Method (int)

Computes the Specified Power of 10 using series Expansion.

**C++**

```
public: decimal TenPow(
    int X
);
```

**C#**

```
public static decimal TenPow(
    int X
);
```

**Visual Basic**

```
Public static Function TenPow(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Power to raise 10 to |

**Returns**

10 ^ X, 10 raised to the power X

## 1.1.3.1.45 TwoPow Method

### 1.1.3.1.45.1 DecimalOps.TwoPow Method (decimal)

Computes the Specified Power of 2 using series Expansion.

**C++**

```
public: decimal TwoPow(
    decimal X
);
```

**C#**

```
public static decimal TwoPow(
    decimal X
);
```

**Visual Basic**

```
Public static Function TwoPow(
    X As decimal
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Power to raise 2 to |

**Returns**

2 ^ X, 2 raised to the power X

### 1.1.3.1.45.2 DecimalOps.TwoPow Method (double)

Computes the Specified Power of 2 using series Expansion.

**C++**

```
public: decimal TwoPow(
    double X
);
```

**C#**

```
public static decimal TwoPow(
    double X
);
```

**Visual Basic**

```
Public static Function TwoPow(
    X As double
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Power to raise 2 to |

**Returns**

2 ^ X, 2 raised to the power X

### 1.1.3.1.45.3 DecimalOps.TwoPow Method (int)

Computes the Specified Power of 2 using series Expansion.

**C++**

```
public: decimal TwoPow(
    int X
);
```

**C#**

```
public static decimal TwoPow(
    int X
);
```

**Visual Basic**

```
Public static Function TwoPow(
    X As Integer
) As decimal
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Power to raise 2 to |

**Returns**

2 ^ X, 2 raised to the power X

## 1.1.4 IntOps Class

**Inheritance Hierarchy**

**C++**

```
public: class IntOps;
```

**C#**

```
public static class IntOps;
```

**Visual Basic**

```
Public static Class IntOps
```

**File**

EsbMath-Int.cs

**Remarks**

Integer Routines for EsbDecimals (↗ see page 1).

**Members**

**IntOps Methods**

| | Name | Description |
|---|---|---|
| S | Ceiling (↗ see page 124) | Returns the Integer Value that is greater than or equal to the specified number.<br>For "larger" values Ceiling64(decimal) |
| S | Ceiling64 (↗ see page 125) | Returns the Integer Value that is greater than or equal to the specified number. |
| S | Floor (↗ see page 125) | Returns the Integer Value that is less than or equal to the specified number.<br>For "larger" values Floor64(decimal) |
| S | Floor64 (↗ see page 126) | Returns the Integer Value that is less than or equal to the specified number. |
| S | IntPart (↗ see page 127) | Returns the Integral portion of the value supplied. |
| S | IntPart64 (↗ see page 128) | Returns the Integral portion of the value supplied. |
| S | Odd (↗ see page 129) | Determines whether an Integral value is Odd. |

**IntOps Methods**

| | Name | Description |
|---|---|---|
| S | Ceiling (↗ see page 124) | Returns the Integer Value that is greater than or equal to the specified number.<br>For "larger" values Ceiling64(decimal) |
| S | Ceiling64 (↗ see page 125) | Returns the Integer Value that is greater than or equal to the specified number. |
| S | Floor (↗ see page 125) | Returns the Integer Value that is less than or equal to the specified number.<br>For "larger" values Floor64(decimal) |
| S | Floor64 (↗ see page 126) | Returns the Integer Value that is less than or equal to the specified number. |
| S | IntPart (↗ see page 127) | Returns the Integral portion of the value supplied. |
| S | IntPart64 (↗ see page 128) | Returns the Integral portion of the value supplied. |
| S | Odd (↗ see page 129) | Determines whether an Integral value is Odd. |

## 1.1.4.1 IntOps Methods

### 1.1.4.1.1 Ceiling Method

### 1.1.4.1.1.1 IntOps.Ceiling Method (decimal)

Returns the Integer Value that is greater than or equal to the specified number.

For "larger" values Ceiling64(decimal)

**C++**

```
public: int Ceiling(
    decimal X
);
```

**C#**

```
public static int Ceiling(
    decimal X
);
```

**Visual Basic**

```
Public static Function Ceiling(
    X As decimal
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to be processed |

**Returns**

Integer Value that is greater than or equal to the specified number.

### 1.1.4.1.1.2 IntOps.Ceiling Method (double)

Returns the Integer Value that is greater than or equal to the specified number.

For "larger" values Ceiling64(decimal)

**C++**

```
public: int Ceiling(
    double X
);
```

**C#**

```
public static int Ceiling(
    double X
);
```

**Visual Basic**

```
Public static Function Ceiling(
    X As double
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to be processed |

**Returns**

Integer Value that is greater than or equal to the specified number.

## 1.1.4.1.2 Ceiling64 Method

## 1.1.4.1.2.1 IntOps.Ceiling64 Method (decimal)

Returns the Integer Value that is greater than or equal to the specified number.

**C++**

```
public: long Ceiling64(
    decimal X
);
```

**C#**

```
public static long Ceiling64(
    decimal X
);
```

**Visual Basic**

```
Public static Function Ceiling64(
    X As decimal
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to be processed |

**Returns**

Integer Value that is greater than or equal to the specified number.

## 1.1.4.1.2.2 IntOps.Ceiling64 Method (double)

Returns the Integer Value that is greater than or equal to the specified number.

**C++**

```
public: long Ceiling64(
    double X
);
```

**C#**

```
public static long Ceiling64(
    double X
);
```

**Visual Basic**

```
Public static Function Ceiling64(
    X As double
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to be processed |

**Returns**

Integer Value that is greater than or equal to the specified number.

# 1.1.4.1.3 Floor Method

## 1.1.4.1.3.1 IntOps.Floor Method (decimal)

Returns the Integer Value that is less than or equal to the specified number.

For "larger" values Floor64(decimal)

**C++**

```
public: int Floor(
    decimal X
);
```

**C#**

```
public static int Floor(
    decimal X
);
```

**Visual Basic**

```
Public static Function Floor(
    X As decimal
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to be processed |

**Returns**

Integer Value that is less than or equal to the specified number.

### 1.1.4.1.3.2 IntOps.Floor Method (double)

Returns the Integer Value that is less than or equal to the specified number.

For "larger" values Floor64(decimal)

**C++**

```
public: int Floor(
    double X
);
```

**C#**

```
public static int Floor(
    double X
);
```

**Visual Basic**

```
Public static Function Floor(
    X As double
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to be processed |

**Returns**

Integer Value that is less than or equal to the specified number.

## 1.1.4.1.4 Floor64 Method

### 1.1.4.1.4.1 IntOps.Floor64 Method (decimal)

Returns the Integer Value that is less than or equal to the specified number.

**C++**

```
public: long Floor64(
    decimal X
);
```

**C#**

```
public static long Floor64(
    decimal X
);
```

**Visual Basic**

```
Public static Function Floor64(
    X As decimal
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Value to be processed |

**Returns**

Integer Value that is less than or equal to the specified number.

### 1.1.4.1.4.2 IntOps.Floor64 Method (double)

Returns the Integer Value that is less than or equal to the specified number.

**C++**

```
public: long Floor64(
    double X
);
```

**C#**

```
public static long Floor64(
    double X
);
```

**Visual Basic**

```
Public static Function Floor64(
    X As double
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Value to be processed |

**Returns**

Integer Value that is less than or equal to the specified number.

## 1.1.4.1.5 IntPart Method

### 1.1.4.1.5.1 IntOps.IntPart Method (decimal)

Returns the Integral portion of the value supplied.

**C++**

```
public: int IntPart(
    decimal X
);
```

**C#**

```
public static int IntPart(
    decimal X
);
```

**Visual Basic**

```
Public static Function IntPart(
    X As decimal
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Number to be processed. |

**Returns**

Integral portion of the value supplied

### 1.1.4.1.5.2 IntOps.IntPart Method (double)

Returns the Integral portion of the value supplied.

**C++**

```
public: int IntPart(
    double X
);
```

**C#**

```
public static int IntPart(
    double X
);
```

**Visual Basic**

```
Public static Function IntPart(
    X As double
) As Integer
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Number to be processed. |

**Returns**

Integral portion of the value supplied

## 1.1.4.1.6 IntPart64 Method

### 1.1.4.1.6.1 IntOps.IntPart64 Method (decimal)

Returns the Integral portion of the value supplied.

**C++**

```
public: long IntPart64(
    decimal X
);
```

**C#**

```
public static long IntPart64(
    decimal X
);
```

**Visual Basic**

```
Public static Function IntPart64(
    X As decimal
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| decimal X | Number to be processed. |

**Returns**

Integral portion of the value supplied

### 1.1.4.1.6.2 IntOps.IntPart64 Method (double)

Returns the Integral portion of the value supplied.

**C++**

```
public: long IntPart64(
    double X
);
```

**C#**

```
public static long IntPart64(
    double X
);
```

**Visual Basic**

```
Public static Function IntPart64(
    X As double
) As long
```

**Parameters**

| Parameters | Description |
|---|---|
| double X | Number to be processed. |

**Returns**

Integral portion of the value supplied

## 1.1.4.1.7 Odd Method

### 1.1.4.1.7.1 IntOps.Odd Method (int)

Determines whether an Integral value is Odd.

**C++**

```
public: bool Odd(
    int X
);
```

**C#**

```
public static bool Odd(
    int X
);
```

**Visual Basic**

```
Public static Function Odd(
    X As Integer
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| int X | Value to Test. |

**Returns**

True if Odd, False if Even.

## 1.1.4.1.7.2 IntOps.Odd Method (long)

Determines whether an Integral value is Odd.

**C++**

```
public: bool Odd(
    long X
);
```

**C#**

```
public static bool Odd(
    long X
);
```

**Visual Basic**

```
Public static Function Odd(
    X As long
) As bool
```

**Parameters**

| Parameters | Description |
|---|---|
| long X | Value to Test. |

**Returns**

True if Odd, False if Even.

# Index

C

CompareOps class 1
  about CompareOps class 1
  CompareValue 4, 5, 6, 7
  DecimalPrecisionDef 3
  DoublePrecisionDef 3
  FloatPrecisionDef 4
  GetPrecision 8, 9, 10, 11
  GreaterValue 11, 12, 13, 14
  LesserValue 15, 16, 17, 18
  SameValue 19, 20, 21, 22
  ValueIsNegative 23, 24, 25
  ValueIsPositive 26, 27, 28, 29
  ValueIsZero 29, 30, 31, 32

D

DecimalGlobals class 33
  about DecimalGlobals class 33
  Cbrt10 36
  Cbrt100 36
  Cbrt2 36
  Cbrt3 36
  CbrtPi 36
  FivePiOver2 37
  FourPiOver3 37
  Gamma 37
  GoldenRatio 37
  InvCbrtPi 38
  InvPi 38
  InvSqrt2 38
  InvSqrt3 38
  InvSqrt5 38
  InvSqrtPi 39
  Ln10 39
  Ln2 39
  LnOfPi 39
  LnSqrt2Pi 40
  Log10Base2 40
  Log2 40
  Log3 40
  LogNaturalConstant 40
  LogPi 41
  NaturalConstant 41
  NaturalConstant2Pi 41
  NaturalConstant2PiOver2 41
  NaturalConstant2PiOver4 42
  OneDegree 42
  OneMinute 42
  OneRadian 42
  OneSecond 42
  Pi 43
  Pi2E 43
  PiOver12 43
  PiOver2 43
  PiOver3 44
  PiOver4 44
  PiOver5 44
  PiOver6 44
  SqrNaturalConstant 44
  SqrPi 45
  Sqrt10 45
  Sqrt2 45
  Sqrt3 45
  Sqrt5 46
  SqrtPi 46
  ThreePi 46
  ThreePiOver2 46
  ThreePiOver4 46
  TwoPi 47
  TwoToPower63 47
  Version 47
  VersionDate 47
DecimalOps class 48
  about DecimalOps class 48
  Acos 51, 52
  AcosDeg 52, 53
  AdjustAngleDeg 53, 54, 55, 56
  AdjustAngleRad 56, 57, 58, 59
  Asin 59, 60

AsinDeg 60, 61

Atan 61, 62

Atan2 63, 64

Atan2Deg 64, 65

AtanDeg 66, 67

Ceiling 67, 68

Cos 68, 69

CosDeg 70

Cosec 71, 72

CosecDeg 73, 74

Cot 75, 76

CotDeg 76, 77

Deg2Dms 78

Deg2Grad 79, 80

Deg2Rad 80, 81

Dms2Deg 82, 83

Exp 83, 84

FractionPart 85

Grad2Deg 86, 87

Grad2Rad 87, 88

IntPart 88, 89

IntPow 89, 90

Log 91, 92

Log10 92, 93

Log2 94, 95

NthRoot 96, 97

Pow 97, 98, 99, 100, 101, 102

Rad2Deg 103, 104

Rad2Grad 104, 105

Sec 105, 106, 107

SecDeg 107, 108

SignXY 109, 110

Sin 111, 112

SinCos 112, 113

SinDeg 114, 115

Sqrt 115, 116

Tan 117, 118

TanDeg 118, 119

TenPow 120, 121

TwoPow 121, 122

E

EsbDecimals 1

EsbDecimals namespace 1

EsbDecimals.CompareOps 1

EsbDecimals.CompareOps.CompareValue 4, 5, 6, 7

EsbDecimals.CompareOps.DecimalPrecisionDef 3

EsbDecimals.CompareOps.DoublePrecisionDef 3

EsbDecimals.CompareOps.FloatPrecisionDef 4

EsbDecimals.CompareOps.GetPrecision 8, 9, 10, 11

EsbDecimals.CompareOps.GreaterValue 11, 12, 13, 14

EsbDecimals.CompareOps.LesserValue 15, 16, 17, 18

EsbDecimals.CompareOps.SameValue 19, 20, 21, 22

EsbDecimals.CompareOps.ValueIsNegative 23, 24, 25

EsbDecimals.CompareOps.ValueIsPositive 26, 27, 28, 29

EsbDecimals.CompareOps.ValueIsZero 29, 30, 31, 32

EsbDecimals.DecimalGlobals 33

EsbDecimals.DecimalGlobals.Cbrt10 36

EsbDecimals.DecimalGlobals.Cbrt100 36

EsbDecimals.DecimalGlobals.Cbrt2 36

EsbDecimals.DecimalGlobals.Cbrt3 36

EsbDecimals.DecimalGlobals.CbrtPi 36

EsbDecimals.DecimalGlobals.FivePiOver2 37

EsbDecimals.DecimalGlobals.FourPiOver3 37

EsbDecimals.DecimalGlobals.Gamma 37

EsbDecimals.DecimalGlobals.GoldenRatio 37

EsbDecimals.DecimalGlobals.InvCbrtPi 38

EsbDecimals.DecimalGlobals.InvPi 38

EsbDecimals.DecimalGlobals.InvSqrt2 38

EsbDecimals.DecimalGlobals.InvSqrt3 38

EsbDecimals.DecimalGlobals.InvSqrt5 38

EsbDecimals.DecimalGlobals.InvSqrtPi 39

EsbDecimals.DecimalGlobals.Ln10 39

EsbDecimals.DecimalGlobals.Ln2 39

EsbDecimals.DecimalGlobals.LnOfPi 39

EsbDecimals.DecimalGlobals.LnSqrt2Pi 40

EsbDecimals.DecimalGlobals.Log10Base2 40

EsbDecimals.DecimalGlobals.Log2 40

EsbDecimals.DecimalGlobals.Log3 40

EsbDecimals.DecimalGlobals.LogNaturalConstant 40

EsbDecimals.DecimalGlobals.LogPi 41

EsbDecimals.DecimalGlobals.NaturalConstant 41
EsbDecimals.DecimalGlobals.NaturalConstant2Pi 41
EsbDecimals.DecimalGlobals.NaturalConstant2PiOver2 41
EsbDecimals.DecimalGlobals.NaturalConstant2PiOver4 42
EsbDecimals.DecimalGlobals.OneDegree 42
EsbDecimals.DecimalGlobals.OneMinute 42
EsbDecimals.DecimalGlobals.OneRadian 42
EsbDecimals.DecimalGlobals.OneSecond 42
EsbDecimals.DecimalGlobals.Pi 43
EsbDecimals.DecimalGlobals.Pi2E 43
EsbDecimals.DecimalGlobals.PiOver12 43
EsbDecimals.DecimalGlobals.PiOver2 43
EsbDecimals.DecimalGlobals.PiOver3 44
EsbDecimals.DecimalGlobals.PiOver4 44
EsbDecimals.DecimalGlobals.PiOver5 44
EsbDecimals.DecimalGlobals.PiOver6 44
EsbDecimals.DecimalGlobals.SqrNaturalConstant 44
EsbDecimals.DecimalGlobals.SqrPi 45
EsbDecimals.DecimalGlobals.Sqrt10 45
EsbDecimals.DecimalGlobals.Sqrt2 45
EsbDecimals.DecimalGlobals.Sqrt3 45
EsbDecimals.DecimalGlobals.Sqrt5 46
EsbDecimals.DecimalGlobals.SqrtPi 46
EsbDecimals.DecimalGlobals.ThreePi 46
EsbDecimals.DecimalGlobals.ThreePiOver2 46
EsbDecimals.DecimalGlobals.ThreePiOver4 46
EsbDecimals.DecimalGlobals.TwoPi 47
EsbDecimals.DecimalGlobals.TwoToPower63 47
EsbDecimals.DecimalGlobals.Version 47
EsbDecimals.DecimalGlobals.VersionDate 47
EsbDecimals.DecimalOps 48
EsbDecimals.DecimalOps.Acos 51, 52
EsbDecimals.DecimalOps.AcosDeg 52, 53
EsbDecimals.DecimalOps.AdjustAngleDeg 53, 54, 55, 56
EsbDecimals.DecimalOps.AdjustAngleRad 56, 57, 58, 59
EsbDecimals.DecimalOps.Asin 59, 60
EsbDecimals.DecimalOps.AsinDeg 60, 61
EsbDecimals.DecimalOps.Atan 61, 62
EsbDecimals.DecimalOps.Atan2 63, 64
EsbDecimals.DecimalOps.Atan2Deg 64, 65
EsbDecimals.DecimalOps.AtanDeg 66, 67
EsbDecimals.DecimalOps.Ceiling 67, 68
EsbDecimals.DecimalOps.Cos 68, 69
EsbDecimals.DecimalOps.CosDeg 70
EsbDecimals.DecimalOps.Cosec 71, 72
EsbDecimals.DecimalOps.CosecDeg 73, 74
EsbDecimals.DecimalOps.Cot 75, 76
EsbDecimals.DecimalOps.CotDeg 76, 77
EsbDecimals.DecimalOps.Deg2Dms 78
EsbDecimals.DecimalOps.Deg2Grad 79, 80
EsbDecimals.DecimalOps.Deg2Rad 80, 81
EsbDecimals.DecimalOps.Dms2Deg 82, 83
EsbDecimals.DecimalOps.Exp 83, 84
EsbDecimals.DecimalOps.FractionPart 85
EsbDecimals.DecimalOps.Grad2Deg 86, 87
EsbDecimals.DecimalOps.Grad2Rad 87, 88
EsbDecimals.DecimalOps.IntPart 88, 89
EsbDecimals.DecimalOps.IntPow 89, 90
EsbDecimals.DecimalOps.Log 91, 92
EsbDecimals.DecimalOps.Log10 92, 93
EsbDecimals.DecimalOps.Log2 94, 95
EsbDecimals.DecimalOps.NthRoot 96, 97
EsbDecimals.DecimalOps.Pow 97, 98, 99, 100, 101, 102
EsbDecimals.DecimalOps.Rad2Deg 103, 104
EsbDecimals.DecimalOps.Rad2Grad 104, 105
EsbDecimals.DecimalOps.Sec 105, 106, 107
EsbDecimals.DecimalOps.SecDeg 107, 108
EsbDecimals.DecimalOps.SignXY 109, 110
EsbDecimals.DecimalOps.Sin 111, 112
EsbDecimals.DecimalOps.SinCos 112, 113
EsbDecimals.DecimalOps.SinDeg 114, 115
EsbDecimals.DecimalOps.Sqrt 115, 116
EsbDecimals.DecimalOps.Tan 117, 118
EsbDecimals.DecimalOps.TanDeg 118, 119
EsbDecimals.DecimalOps.TenPow 120, 121
EsbDecimals.DecimalOps.TwoPow 121, 122
EsbDecimals.IntOps 122
EsbDecimals.IntOps.Ceiling 124
EsbDecimals.IntOps.Ceiling64 125
EsbDecimals.IntOps.Floor 125, 126
EsbDecimals.IntOps.Floor64 126, 127
EsbDecimals.IntOps.IntPart 127, 128

EsbDecimals.IntOps.IntPart64 128, 129

EsbDecimals.IntOps.Odd 129, 130

I

IntOps class 122
    about IntOps class 122
    Ceiling 124
    Ceiling64 125
    Floor 125, 126
    Floor64 126, 127
    IntPart 127, 128
    IntPart64 128, 129
    Odd 129, 130