NSDecimalNumber

Inherits From:	NSNumber : NSValue : NSObject
Conforms To:	NSDecimalNumberBehavior NSObject (NSObject)
Declared In:	Foundation/NSDecimalNumber.h

Class Description

NSDecimalNumber, an immutable subclass of NSNumber, provides an object-oriented wrapper for doing base-10 arithmetic. An instance can represent any number that can be expressed as *mantissa* **x** $10^{exponent}$ where *mantissa* is a decimal integer up to 38 digits long, and *exponent* is an integer between -128 and 127.

In the course of doing arithmetic, a method may produce calculation errors, such as division by zero. It may also meet circumstances where it has a choice of ways to round a number off. The way the method acts on such occasions is called its "behavior."

Behavior is set by methods in the NSDecimalNumberBehaviors protocol. Every NSDecimalNumber argument called *behavior* requires an object that conforms to this protocol. For more on behaviors, see the specifications for the NSDecimalNumberBehaviors protocol and the NSDecimalNumberHandler class. Also see the **defaultBehavior** method description, below.

C Interface to Decimal Numbers

The arithmetic and rounding methods of NSDecimalNumber are also accessible through group of ordinary C functions, defined in NSDecimal.h. You might consider the C interface if you don't need to treat NSDecimalNumbers as objects—that is, if you don't need to store them in an object-oriented collection like an NSArray or NSDictionary.

You might also consider the C interface if you need maximum efficiency. The C interface is faster and uses less memory than the NSDecimalNumberClass.

If you need mutability, you can combine the two interfaces. Use functions from the C interface and convert their results to NSDecimalNumbers.

The C functions—**NSDecimalCompact()**, **NSDecimalCompare()**, **NSDecimalRound()**, **NSDecimalNormalize()**, **NSDecimalAdd()**, **NSDecimalSubtract()**, **NSDecimalMultiply()**, **NSDecimalDivide()**, **NSDecimalPower()**, **NSDecimalMultiplyByPowerOf10()**, **NSDecimalString()**—are all documented in the "Functions" chapter of the *Foundation Framework Reference*.

Method Types

Creating an NSDecimalNumber	+ decimalNumberWithDecimal: + decimalNumberWithMantissa:exponent:isNegative: + decimalNumberWithString: + decimalNumberWithString:locale: + one + zero + notANumber
Initializing an NSDecimalNumber	– initWithDecimal: – initWithMantissa:exponent:isNegative: – initWithString: – initWithString:locale:
Doing arithmetic	– decimalNumberByAdding: – decimalNumberBySubtracting: – decimalNumberByMultiplyingBy: – decimalNumberByDividingBy: – decimalNumberByRaisingToPower: – decimalNumberByMultiplyingByPowerOf10: – decimalNumberByAdding:withBehavior: – decimalNumberBySubtracting:withBehavior: – decimalNumberByMultiplyingBy:withBehavior: – decimalNumberByDividingBy:withBehavior: – decimalNumberByRaisingToPower:withBehavior: – decimalNumberByMultiplyingByPowerOf10:withBehavior:
Rounding off	– decimalNumberByRoundingAccordingToBehavior:
Getting the value in other formats	– decimalValue
Asking and changing the behavior	+ defaultBehavior + setDefaultBehavior:

Class Methods

decimalNumberWithDecimal:

+ (NSDecimalNumber *)**decimalNumberWithDecimal:**(NSDecimal)*decimal*

Creates and returns an NSDecimalNumber equivalent to *decimal*.

decimal is an NSDecimal struct, which you can initialize by hand, or generate using the **scanDecimal:** method from the NSDecimalNumberScanning category of NSScanner, defined in “NSDecimalNumber.h”.

decimalNumberWithMantissa:exponent:isNegative:

+ (NSDecimalNumber *)**decimalNumberWithMantissa:**(unsigned long long)*mantissa*
exponent:(short)*exponent* **isNegative:**(BOOL)*isNegative*

Creates and returns an NSDecimalNumber equivalent to the number specified by the arguments.

The arguments express a number in a kind of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is 1.23, it is expressed as $123x10^{-2}$—*mantissa* is 123, *exponent* is -2, and *isNegative*, which refers to the sign of the mantissa, is NO.

decimalNumberWithString:

+ (NSDecimalNumber *)**decimalNumberWithString:**(NSString *)*numericString*

Creates and returns an NSDecimalNumber equivalent to *numericString.* Besides digits, *numericString* can include an initial “+” or “-,” a single “E” or “e”, to indicate the exponent of a number in scientific notation, and a single NSDecimalSeparator to divide the fractional from the integral part of the number.

Whether the NSDecimalSeparator is a period (as in the United States) or a comma (as in France) depends on the default locale.

See also: – decimalNumberWithString:locale:

decimalNumberWithString:locale:

+ (NSDecimalNumber *)**decimalNumberWithString:**(NSString *)*numericString*
locale:(NSDictionary *)*locale*

Creates and returns an NSDecimalNumber equivalent to *numericString.* Besides digits, *numericString* can include an initial “+” or “-,” a single “E” or “e”, to indicate the exponent of a number in scientific notation, and a single NSDecimalSeparator to divide the fractional from the integral part of the number.

locale determines whether the NSDecimalSeparator is a period (as in the United States) or a comma (as in France).

The following strings are acceptable values for *numericString*:

- “2500.6” (or “2500,6”, depending on *locale*)
- “-2500.6” (or “-2500.6”)
- “-2.5006e3” (or “-2,5006e3”)
- “-2.5006E3” (or “-2,5006E3”)

The following are unacceptable:

- “2,500.6”
- “2500 3/5”
- “2.5006x10e3”

- “two thousand five hundred and six tenths”

See also: **– decimalNumberWithString:**

defaultBehavior

+ (id <NSDecimalNumberBehaviors>)**defaultBehavior**

Returns the way that arithmetic methods, like **decimalNumberByAdding:**, round off and handle error conditions.

By default, the arithmetic methods do not round numbers off. They assume that your need for precision does not exceed 38 significant digits. And they raise exceptions when they try to divide by zero, or when they produce a number that is too big or small to be represented.

If this default behavior doesn’t suit your application, you should use methods that let you specify the behavior, like **decimalNumberByAdding:withBehavior.** If you find yourself using a particular behavior consistently, you can specify a different default behavior with **setDefaultBehavior:**.

notANumber

+ (NSDecimalNumber *)**notANumber**

Creates and returns an NSDecimalNumber that specifies no number. Any arithmetic method receiving **notANumber** as an argument returns **notANumber**.

This value can be a useful way of handling non-numeric data in an input file. It can also be a useful response to calculation errors. For more information on calculation errors, see the **exceptionDuringOperation:error:leftOperand:rightOperand:** method description in the NSDecimalNumberBehaviors protocol specification.

one

+ (NSDecimalNumber *)**one**

Creates and returns an NSDecimalNumber equivalent to the number 1.0.

See also: + **zero**

setDefaultBehavior:

+ (void)**setDefaultBehavior:**(id <NSDecimalNumberBehaviors>)*behavior*

Specifies the way that arithmetic methods, like **decimalNumberByAdding:**, round off and handle error conditions. *behavior* must conform to the NSDecimalNumberBehaviors protocol.

zero

+ (NSDecimalNumber *)**zero**

Returns a newly allocated NSDecimalNumber equivalent to the number 0.0.

See also: + **one**

Instance Methods

decimalNumberByAdding:

– (NSDecimalNumber *)**decimalNumberByAdding:**(NSDecimalNumber *)*decimalNumber*

Adds *decimalNumber* to the receiver, and returns the sum, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and rounding.

See also: **– decimalNumberByAdding:withBehavior:**, + **defaultBehavior**

decimalNumberByAdding:withBehavior:

– (NSDecimalNumber *)**decimalNumberByAdding:**(NSDecimalNumber *)*decimalNumber*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Adds *decimalNumber* to the receiver, and returns the sum, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalNumberByDividingBy:

– (NSDecimalNumber *)**decimalNumberByDividingBy:**(NSDecimalNumber *)*decimalNumber*

Divides the receiver by *decimalNumber*, and returns the quotient, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and rounding.

See also: – **decimalNumberByDividingBy:withBehavior:**, + **defaultBehavior**

decimalNumberByDividingBy:withBehavior:

– (NSDecimalNumber *)**decimalNumberByDividingBy:**(NSDecimalNumber *)*decimalNumber*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Divides the receiver by *decimalNumber*, and returns the quotient, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalNumberByMultiplyingBy:

– (NSDecimalNumber *)**decimalNumberByMultiplyingBy:**(NSDecimalNumber *)*decimalNumber*

Multiplies the receiver by *decimalNumber*, and returns the product, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and when rounding.

See also: – **decimalNumberByMultiplyingBy:withBehavior:**, + **defaultBehavior**

decimalNumberByMultiplyingBy:withBehavior:

– (NSDecimalNumber *)**decimalNumberByMultiplyingBy:**(NSDecimalNumber *)*decimalNumber*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Multiplies the receiver by *decimalNumber*, and returns the product, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalNumberByMultiplyingByPowerOf10:

– (NSDecimalNumber *)**decimalNumberByMultiplyingByPowerOf10:**(short)*power*

Multiplies the receiver by 10^{power}, and returns the product, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and when rounding.

See also: – **decimalNumberByMultiplyingByPowerOf10:withBehavior:**, + **defaultBehavior**

decimalNumberByMultiplyingByPowerOf10:withBehavior:

– (NSDecimalNumber *)**decimalNumberByMultiplyingByPowerOf10:**(short)*power*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Multiplies the receiver by 10^{power}, and returns the product, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalNumberByRaisingToPower:

– (NSDecimalNumber *)**decimalNumberByRaisingToPower:**(unsigned)*power*

Raises the receiver to *power*, and returns the result, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and when rounding.

See also: **– decimalNumberByRaisingToPower:withBehavior:**, + **defaultBehavior**

decimalNumberByRaisingToPower:withBehavior:

– (NSDecimalNumber *)**decimalNumberByRaisingToPower:**(unsigned)*power*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Raises the receiver to *power*, and returns the result, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalNumberByRoundingAccordingToBehavior:

– (NSDecimalNumber *)**decimalNumberByRoundingAccordingToBehavior:**(id <NSDecimalNumberBehaviors>)*behavior*

Rounds the receiver off in the way specified by *behavior,* and returns the result, a newly created NSDecimalNumber. For a description of the different ways of rounding, see the **roundingMode** method in the NSDecimalNumberHandler class specification.

decimalNumberBySubtracting:

– (NSDecimalNumber *)**decimalNumberBySubtracting:**(NSDecimalNumber *)*decimalNumber*

Subtracts *decimalNumber* from the receiver, and returns the difference, a newly created NSDecimalNumber. This method uses the default behavior when handling calculation errors and when rounding.

See also: – **decimalNumberBySubtracting:withBehavior:**, + **defaultBehavior**.

decimalNumberBySubtracting:withBehavior:

– (NSDecimalNumber *)**decimalNumberBySubtracting:**(NSDecimalNumber *)*decimalNumber*
withBehavior:(id <NSDecimalNumberBehaviors>)*behavior*

Subtracts *decimalNumber* from the receiver, and returns the difference, a newly created NSDecimalNumber. *behavior* specifies the handling of calculation errors and rounding.

decimalValue

– (NSDecimal)**decimalValue**

Returns the receiver's value, expressed as an NSDecimal struct.

initWithDecimal:

– (NSDecimalNumber *)**initWithDecimal:**(NSDecimal)*decimal*

Returns an NSDecimalNumber initialized to represent *decimal*. This method is NSDecimalNumber's designated initializer.

initWithMantissa:exponent:isNegative:

– (NSDecimalNumber *)**initWithMantissa:**(unsigned long long)*mantissa* **exponent:**(short)*exponent* **isNegative:**(BOOL)*isNegative*

Creates and returns an NSDecimalNumber equivalent to the number specified by the arguments.

The arguments express a number in a type of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is 1.23, it is expressed as $123x10^{-2}$—*mantissa* is 123, *exponent* is -2, and *isNegative*, which refers to the sign of the mantissa, is NO.

initWithString:

– (NSDecimalNumber *)**initWithString:**(NSString *)*numericString*

Returns an NSDecimalNumber equivalent to *numericString. numericString* must be a simple string of digits, possibly including a decimal separator. For a listing of acceptable and unacceptable strings, see the class method **decimalNumberWithString:**.

initWithString:locale:

– (NSDecimalNumber *)**initWithString:**(NSString *)*numericString* **locale:**(NSDictionary *)*locale*

Returns a newly created NSDecimalNumber equivalent to *numericString.* The interpretation of the numeric string depends on *locale*.

See also: + **decimalNumberWithString:locale:**