Structure

Double

A double-precision, floating-point value type.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 13.0+ | macOS 10.10+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
@frozen
struct Double
```

Topics

Converting Integers

`init<Source>(Source)`

Creates a new value, rounded to the closest possible representation.

`init(Int)`

Creates a new value, rounded to the closest possible representation.

Converting Strings

`init?<S>(S)`

Creates a new instance from the given string.

`init?(Substring)`

Converting Floating-Point Values

`init<Source>(Source)`

Creates a new instance from the given value, rounded to the closest possible representation.

`init(Double)`

Creates a new instance initialized to the given value.

`init(Float)`

Creates a new instance that approximates the given value.

`init(Float16)`

Creates a new instance that approximates the given value.

`init(Float80)`

Creates a new instance that approximates the given value.

`init(CGFloat)`

`init(sign: FloatingPointSign, exponent: Int, significand: Double)`

Creates a new value from the given sign, exponent, and significand.

`init(signOf: Double, magnitudeOf: Double)`

Creates a new floating-point value using the sign of one value and the magnitude of another.

`init<Source>(Source)`

Creates a new value, rounded to the closest possible representation.

`init(truncating: NSNumber)`

Converting with No Loss of Precision

These initializers result in `nil` if the value passed can't be represented without any loss of precision.

`init?<Source>(exactly: Source)`

Creates a new instance from the given value, if it can be represented exactly.

`init?<Source>(exactly: Source)`

Creates a new value, if the given integer can be represented exactly.

`init?<Source>(exactly: Source)`

Creates a new value, if the given integer can be represented exactly.

`init?(exactly: Double)`

Creates a new instance initialized to the given value, if it can be represented without rounding.

`init?(exactly: Float)`

Creates a new instance initialized to the given value, if it can be represented without rounding.

`init?(exactly: Float16)`

Creates a new instance initialized to the given value, if it can be represented without rounding.

`init?(exactly: Float80)`

Creates a new instance initialized to the given value, if it can be represented without rounding.

`init?(exactly: NSNumber)`

Creating a Random Value

`static func random(in: Range<Self>) -> Self`

Returns a random value within the specified range.

`static func random<T>(in: Range<Self>, using: inout T) -> Self`

Returns a random value within the specified range, using the given generator as a source for randomness.

`static func random(in: ClosedRange<Self>) -> Self`

Returns a random value within the specified range.

`static func random<T>(in: ClosedRange<Self>, using: inout T) -> Self`

Returns a random value within the specified range, using the given generator as a source for randomness.

Performing Calculations

Floating-Point Operators for Double

Perform arithmetic and bitwise operations or compare values.

`func addingProduct(Self, Self) -> Self`

Returns the result of adding the product of the two given values to this value, computed without intermediate rounding.

`func addProduct(Double, Double)`

Adds the product of the two given values to this value in place, computed without intermediate rounding.

`func squareRoot() -> Self`

Returns the square root of the value, rounded to a representable value.

`func formSquareRoot()`

Replaces this value with its square root, rounded to a representable value.

`func remainder(dividingBy: Self) -> Self`

Returns the remainder of this value divided by the given value.

`func formRemainder(dividingBy: Double)`

Replaces this value with the remainder of itself divided by the given value.

`func truncatingRemainder(dividingBy: Self) -> Self`

Returns the remainder of this value divided by the given value using truncating division.

`func formTruncatingRemainder(dividingBy: Double)`

Replaces this value with the remainder of itself divided by the given value using truncating division.

`func negate()`

Replaces this value with its additive inverse.

Rounding Values

`func rounded() -> Self`

`func rounded(FloatingPointRoundingRule) -> Self`

Returns this value rounded to an integral value using the specified rounding rule.

`func round()`

`func round(FloatingPointRoundingRule)`

Rounds the value to an integral value using the specified rounding rule.

Comparing Values

Floating-Point Operators for Double

Perform arithmetic and bitwise operations or compare values.

`func isEqual(to: Double) -> Bool`

Returns a Boolean value indicating whether this instance is equal to the given value.

`func isLess(than: Double) -> Bool`

Returns a Boolean value indicating whether this instance is less than the given value.

`func isLessThanOrEqualTo(Double) -> Bool`

Returns a Boolean value indicating whether this instance is less than or equal to the given value.

`func isTotallyOrdered(belowOrEqualTo: Self) -> Bool`

Returns a Boolean value indicating whether this instance should precede or tie positions with the given value in an ascending sort.

`static func minimum(Self, Self) -> Self`

Returns the lesser of the two given values.

`static func minimumMagnitude(Self, Self) -> Self`

Returns the value with lesser magnitude.

`static func maximum(Self, Self) -> Self`

Returns the greater of the two given values.

`static func maximumMagnitude(Self, Self) -> Self`

Returns the value with greater magnitude.

Finding the Sign and Magnitude

`var magnitude: Double`

The magnitude of this value.

`var sign: FloatingPointSign`

The sign of the floating-point value.

`typealias Magnitude`

A type that can represent the absolute value of any possible value of the conforming type.

Querying a Double

`var ulp: Double`

The unit in the last place of this value.

`var significand: Double`

The significand of the floating-point value.

`var exponent: Int`

The exponent of the floating-point value.

`var nextUp: Double`

The least representable value that compares greater than this value.

`var nextDown: Self`

The greatest representable value that compares less than this value.

`var binade: Double`

The floating-point value with the same sign and exponent as this value, but with a significand of 1.0.

Accessing Numeric Constants

`static var pi: Double`

The mathematical constant pi (π), approximately equal to 3.14159.

`static var infinity: Double`

Positive infinity.

`static var greatestFiniteMagnitude: Double`

The greatest finite number representable by this type.

`static var nan: Double`

A quiet NaN (“not a number”).

`static var signalingNaN: Double`

A signaling NaN (“not a number”).

`static var ulpOfOne: Double`

The unit in the last place of 1.0.

`static var leastNonzeroMagnitude: Double`

The least positive number.

`static var leastNormalMagnitude: Double`

The least positive normal number.

`static var zero: Self`

The zero value.

Working with Binary Representation

`var bitPattern: UInt64`

The bit pattern of the value's encoding.

`var significandBitPattern: UInt64`

The raw encoding of the value's significand field.

`var significandWidth: Int`

The number of bits required to represent the value's significand.

`var exponentBitPattern: UInt`

The raw encoding of the value's exponent field.

`static var significandBitCount: Int`

The available number of fractional significand bits.

`static var exponentBitCount: Int`

The number of bits used to represent the type's exponent.

`static var radix: Int`

The radix, or base of exponentiation, for a floating-point type.

`init(bitPattern: UInt64)`

Creates a new value with the given bit pattern.

`init(sign: FloatingPointSign, exponentBitPattern: UInt, significandBitPattern: UInt64)`

Creates a new instance from the specified sign and bit patterns.

`init(nan: Double.RawSignificand, signaling: Bool)`

Creates a NaN ("not a number") value with the specified payload.

`typealias Exponent`

A type that can represent any written exponent.

`typealias RawSignificand`

A type that represents the encoded significand of a value.

`typealias RawExponent`

A type that represents the encoded exponent of a value.

Querying a Double’s State

`var isZero: Bool`

A Boolean value indicating whether the instance is equal to zero.

`var isFinite: Bool`

A Boolean value indicating whether this instance is finite.

`var isInfinite: Bool`

A Boolean value indicating whether the instance is infinite.

`var isNaN: Bool`

A Boolean value indicating whether the instance is NaN (“not a number”).

`var isSignalingNaN: Bool`

A Boolean value indicating whether the instance is a signaling NaN.

`var isNormal: Bool`

A Boolean value indicating whether this instance is normal.

`var isSubnormal: Bool`

A Boolean value indicating whether the instance is subnormal.

`var isCanonical: Bool`

A Boolean value indicating whether the instance’s representation is in its canonical form.

`var floatingPointClass: FloatingPointClassification`

The classification of this value.

Encoding and Decoding Values

`func encode(to: any Encoder) throws`

Encodes this value into the given encoder.

`init(from: any Decoder) throws`

Creates a new instance by decoding from the given decoder.

Creating a Range

`static func ... (Self, Self) -> ClosedRange<Self>`

Returns a closed range that contains both of its bounds.

Describing a Double

`var description: String`

A textual representation of the value.

`var debugDescription: String`

A textual representation of the value, suitable for debugging.

`var customMirror: Mirror`

A mirror that reflects the `Double` instance.

`func hash(into: inout Hasher)`

Hashes the essential components of this value by feeding them into the given hasher.

Infrequently Used Functionality

`init()`

`init(floatLiteral: Double)`

Creates an instance initialized to the specified floating-point value.

`init(integerLiteral: Int64)`

Creates an instance initialized to the specified integer value.

`init(integerLiteral: Self)`

`typealias FloatLiteralType`

A type that represents a floating-point literal.

`typealias IntegerLiteralType`

A type that represents an integer literal.

`func advanced(by: Double) -> Double`

Returns a value that is offset the specified distance from this value.

`func distance(to: Double) -> Double`

Returns the distance from this value to the given value, expressed as a stride.

`typealias Stride`

A type that represents the distance between two values.

`func write<Target>(to: inout Target)`

Writes a textual representation of this instance into the given output stream.

`var hashValue: Int`

The hash value.

SIMD-Supporting Types

`typealias SIMDMaskScalar`

`struct SIMD2Storage`

Storage for a vector of two floating-point values.

`struct SIMD4Storage`

Storage for a vector of four floating-point values.

`struct SIMD8Storage`

Storage for a vector of eight floating-point values.

`struct SIMD16Storage`

Storage for a vector of 16 floating-point values.

`struct SIMD32Storage`

Storage for a vector of 32 floating-point values.

`struct SIMD64Storage`

Storage for a vector of 64 floating-point values.

Deprecated

~~`var customPlaygroundQuickLook: _PlaygroundQuickLook`~~

A custom playground Quick Look for the `Double` instance.

Deprecated

`init(NSNumber)`

Type Aliases

```
typealias Specification
```

```
typealias UnwrappedType
```

```
typealias ValueType
```

Type Properties

```
static var defaultResolverSpecification: some ResolverSpecification
```

```
static var mlMultiArrayDataType: MLMultiArrayDataType
```

Default Implementations

- AdditiveArithmetic Implementations
- AtomicRepresentable Implementations
- BinaryFloatingPoint Implementations
- Comparable Implementations
- CustomDebugStringConvertible Implementations
- CustomReflectable Implementations
- CustomStringConvertible Implementations
- Decodable Implementations
- Encodable Implementations
- Equatable Implementations
- ExpressibleByFloatLiteral Implementations
- ExpressibleByIntegerLiteral Implementations
- FloatingPoint Implementations
- Hashable Implementations
- LosslessStringConvertible Implementations
- Numeric Implementations
- SIMDScalar Implementations

- SignedNumeric Implementations
- Strideable Implementations
- TextOutputStreamable Implementations

Relationships

Conforms To

`AdditiveArithmetic`
`Animatable`
`AnimatableData`
`AtomicRepresentable`
`BinaryFloatingPoint`
`BindableData`
`BitwiseCopyable`
`CKRecordValueProtocol`
`CVarArg`
`Comparable`
`ConvertibleFromGeneratedContent`
`ConvertibleToGeneratedContent`
`Copyable`
`CustomDebugStringConvertible`
`CustomReflectable`
`CustomStringConvertible`
`Decodable`
`Encodable`
`Equatable`
`ExpressibleByFloatLiteral`
`ExpressibleByIntegerLiteral`
`FloatingPoint`
`Generable`
`Hashable`
`InstructionsRepresentable`
`LosslessStringConvertible`
`MLDataValueConvertible`
`MLShapedArrayScalar`
`Numeric`
`Plottable`
`PrimitivePlottableProtocol`

`PromptRepresentable`
`RangeComparableProperty`
`SIMDScalar`
`Sendable`
`SendableMetatype`
`SignedNumeric`
`Strideable`
`TextOutputStreamable`
`VectorArithmetic`
`vDSP_DiscreteFourierTransformable`
`vDSP_FloatingPointBiquadFilterable`
`vDSP_FloatingPointConvertable`
`vDSP_FloatingPointDiscreteFourierTransformable`
`vDSP_FloatingPointGeneratable`

See Also

Standard Library

`struct Int`

A signed integer value type.

`struct String`

A Unicode string value that is a collection of characters.

`struct Array`

An ordered, random-access collection.

`struct Dictionary`

A collection whose elements are key-value pairs.

Swift Standard Library

Solve complex problems and write high-performance, readable code.