Swift Standard Library Functions

This chapter describes the global functions defined in the Swift standard library.

Language

Swift

Symbols

Functions

```
func abs<T>(T)
```

Returns the absolute value of `x`.

```
func assert(() -> Bool, () -> String, file: StaticString, line: UInt)
```

Performs a traditional C-style assert with an optional message.

```
func assertionFailure(() -> String, file: StaticString, line: UInt)
```

Indicates that an internal sanity check failed.

```
func debugPrint(Any..., separator: String, terminator: String)
```

Writes the textual representations of the given items most suitable for debugging into the standard output.

```
func debugPrint<Target>(Any..., separator: String, terminator: String,
to: inout Target)
```

Writes the textual representations of the given items most suitable for debugging into the given output stream.

```
func dump<T>(T, name: String?, indent: Int, maxDepth: Int, maxItems: I
nt)
```

Dumps an object's contents using its mirror to standard output.

```
func dump<T, TargetStream>(T, to: inout TargetStream, name: String?, i
ndent: Int, maxDepth: Int, maxItems: Int)
```

Dumps an object's contents using its mirror to the specified output stream.

```
func fatalError(() -> String, file: StaticString, line: UInt)
```

Unconditionally prints a given message and stops execution.

```
func getVaList([CVarArg])
```

Returns a `CVaListPointer` built from `args` that's backed by autoreleased storage.

```
func isKnownUniquelyReferenced<T>(inout T?)
```

Returns a Boolean value indicating whether the given object is a class instance known to have a single strong reference.

strong reference.

```
func isKnownUniquelyReferenced<T>(inout T)
```

Returns a Boolean value indicating whether the given object is a class instance known to have a single strong reference.

```
func max<T>(T, T)
```

Returns the greater of two comparable values.

```
func max<T>(T, T, T, T...)
```

Returns the greatest argument passed.

```
func min<T>(T, T)
```

Returns the lesser of two comparable values.

```
func min<T>(T, T, T, T...)
```

Returns the least argument passed.

```
func numericCast<T, U>(T)
```

Convert `x` to type `U`, trapping on overflow in -Onone and -O builds.

```
func numericCast<T, U>(T)
```

Convert `x` to type `U`, trapping on overflow in -Onone and -O builds.

```
func numericCast<T, U>(T)
```

Convert `x` to type `U`, trapping on overflow in -Onone and -O builds.

```
func precondition(() -> Bool, () -> String, file: StaticString, line:
UInt)
```

Checks a necessary condition for making forward progress.

```
func preconditionFailure(() -> String, file: StaticString, line: UInt)
```

Indicates that a precondition was violated.

```
func print(Any..., separator: String, terminator: String)
```

Writes the textual representations of the given items into the standard output.

```
func print<Target>(Any..., separator: String, terminator: String, to:
inout Target)
```

Writes the textual representations of the given items into the given output stream.

```
func readLine(strippingNewline: Bool)
```

Returns a string read from standard input through the end of the current line or until EOF is reached.

```
func repeatElement<T>(T, count: Int)
```

Creates a collection containing the specified number of the given element.

```
func sequence<T>(first: T, next: @escaping (T) -> T?)
```

Returns a sequence formed from `first` and repeated lazy applications of `next`.

```
func sequence<T, State>(state: State, next: @escaping (inout State) ->
```

T?)

Returns a sequence formed from repeated lazy applications of `next` to a mutable `state`.

```
func stride<T>(from: T, through: T, by: T.Stride)
```

Returns the sequence of values (`self`, `self + stride`, `self + 2 * stride`, … *last*) where *last* is the last value in the progression less than or equal to `end`.

```
func stride<T>(from: T, to: T, by: T.Stride)
```

Returns the sequence of values (`self`, `self + stride`, `self + 2 * stride`, … *last*) where *last* is the last value in the progression that is less than `end`.

```
func swap<T>(inout T, inout T)
```

Exchange the values of `a` and `b`.

```
func transcode<Input, InputEncoding, OutputEncoding>(Input, from: Inpu
tEncoding.Type, to: OutputEncoding.Type, stoppingOnError: Bool, into:
(OutputEncoding.CodeUnit) -> Void)
```

Translates the given input from one Unicode encoding to another by calling the given closure.

```
func unsafeBitCast<T, U>(T, to: U.Type)
```

Returns the bits of `x`, interpreted as having type `U`.

```
func unsafeDowncast<T>(AnyObject, to: T.Type)
```

```
func withExtendedLifetime<T, Result>(T, () -> Result)
```

Evaluate `f()` and return its result, ensuring that `x` is not destroyed before f returns.

```
func withExtendedLifetime<T, Result>(T, (T) -> Result)
```

Evaluate `f(x)` and return its result, ensuring that `x` is not destroyed before f returns.

```
func withUnsafeBytes<T, Result>(of: inout T, (UnsafeRawBufferPointer)
-> Result)
```

Invokes `body` with an `UnsafeRawBufferPointer` argument and returns the result.

```
func withUnsafeMutableBytes<T, Result>(of: inout T, (UnsafeMutableRawB
ufferPointer) -> Result)
```

Invokes `body` with an `UnsafeMutableRawBufferPointer` argument and returns the result.

```
func withUnsafeMutablePointer<T, Result>(to: inout T, (UnsafeMutablePo
inter<T>) -> Result)
```

Invokes `body` with an `UnsafeMutablePointer` to `arg` and returns the result. Useful for calling Objective-C APIs that take “in/out” parameters (and default-constructible “out” parameters) by pointer.

```
func withUnsafePointer<T, Result>(to: inout T, (UnsafePointer<T>) -> R
esult)
```

Invokes `body` with an `UnsafePointer` to `arg` and returns the result. Useful for calling Objective-C APIs that take “in/out” parameters (and default-constructible “out” parameters) by pointer.

```
func withVaList<R>([CVarArg], (CVaListPointer) -> R)
```

Invoke `body` with a C `va_list` argument derived from `args`.

```
func zip<Sequence1, Sequence2>(Sequence1, Sequence2)
```

Creates a sequence of pairs built out of two underyling sequences.