orphan:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\swift-main\docs\ABI\[swift-main][docs][ABI]OldMangling.rst, line 5)

Unknown directive type "highlight".

```
.. highlight:: none
```

Mangling

This file documents ONLY the old mangling scheme in use before Swift 4.0, which is still used for the Objective-C class names of Swift classes.

```
mangled-name ::= '_T' global
```

All Swift-mangled names begin with this prefix.

Globals

```
global ::= 't' type                    // standalone type (for DWARF)
global ::= 'M' type                    // type metadata (address point)
                                       // -- type starts with [BCOSTV]
global ::= 'Mf' type                   // 'full' type metadata (start of object)
global ::= 'MP' type                   // type metadata pattern
global ::= 'Ma' type                   // type metadata access function
global ::= 'ML' type                   // type metadata lazy cache variable
global ::= 'Mm' type                   // class metaclass
global ::= 'Mn' nominal-type           // nominal type descriptor
global ::= 'Mp' protocol               // protocol descriptor
global ::= 'MR' remote-reflection-record // metadata for remote mirrors
global ::= 'PA' .*                     // partial application forwarder
global ::= 'PAo' .*                    // ObjC partial application forwarder
global ::= 'w' value-witness-kind type // value witness
global ::= 'Wa' protocol-conformance   // protocol witness table accessor
global ::= 'WG' protocol-conformance   // generic protocol witness table
global ::= 'WI' protocol-conformance   // generic protocol witness table instantiation function
global ::= 'Wl' type protocol-conformance // lazy protocol witness table accessor
global ::= 'WL' protocol-conformance   // lazy protocol witness table cache variable
global ::= 'Wo' entity                 // witness table offset
global ::= 'WP' protocol-conformance   // protocol witness table
global ::= 'Wt' protocol-conformance identifier // associated type metadata accessor
global ::= 'WT' protocol-conformance identifier nominal-type // associated type witness table accessor
global ::= 'Wv' directness entity      // field offset
global ::= 'WV' type                   // value witness table
global ::= entity                      // some identifiable thing
global ::= 'TO' global                 // ObjC-as-swift thunk
global ::= 'To' global                 // swift-as-ObjC thunk
global ::= 'TD' global                 // dynamic dispatch thunk
global ::= 'Td' global                 // direct method reference thunk
global ::= 'TR' reabstract-signature   // reabstraction thunk helper function
global ::= 'Tr' reabstract-signature   // reabstraction thunk

global ::= 'TS' specializationinfo '_' mangled-name
specializationinfo ::= 'g' passid (type protocol-conformance* '_')+           // Generic specialization info.
specializationinfo ::= 'f' passid (funcspecializationarginfo '_')+            // Function signature specialization kind
passid ::= integer                                                           // The id of the pass that generated this sp
funcsigspecializationarginfo ::= 'cl' closurename type*                       // Closure specialized with closed over type
funcsigspecializationarginfo ::= 'n'                                         // Unmodified argument
funcsigspecializationarginfo ::= 'cp' funcsigspecializationconstantproppayload // Constant propagated argument
funcsigspecializationarginfo ::= 'd'                                         // Dead argument
funcsigspecializationarginfo ::= 'g' 's'?                                    // Owned => Guaranteed and Exploded if 's' p
funcsigspecializationarginfo ::= 's'                                         // Exploded
funcsigspecializationarginfo ::= 'k'                                         // Exploded
funcsigspecializationconstantpropinfo ::= 'fr' mangled-name
funcsigspecializationconstantpropinfo ::= 'g' mangled-name
funcsigspecializationconstantpropinfo ::= 'i' 64-bit-integer
funcsigspecializationconstantpropinfo ::= 'fl' float-as-64-bit-integer
funcsigspecializationconstantpropinfo ::= 'se' stringencoding 'v' md5hash

global ::= 'TV' global                 // vtable override thunk
global ::= 'TW' protocol-conformance entity
                                       // protocol witness thunk
global ::= 'TB' identifier context identifier
                                       // property behavior initializer thunk
global ::= 'Tb' identifier context identifier
                                       // property behavior setter thunk

entity ::= nominal-type                // named type declaration
entity ::= static? entity-kind context entity-name
entity-kind ::= 'F'                    // function (ctor, accessor, etc.)
entity-kind ::= 'v'                    // variable (let/var)
entity-kind ::= 'i'                    // subscript ('i'ndex) itself (not the individual accessors)
entity-kind ::= 'I'                    // initializer
entity-name ::= decl-name type         // named declaration
entity-name ::= 'A' index              // default argument generator
entity-name ::= 'a' addressor-kind decl-name type     // mutable addressor
entity-name ::= 'C' type               // allocating constructor
entity-name ::= 'c' type               // non-allocating constructor
entity-name ::= 'D'                    // deallocating destructor; untyped
entity-name ::= 'd'                    // non-deallocating destructor; untyped
entity-name ::= 'g' decl-name type     // getter
entity-name ::= 'i'                    // non-local variable initializer
entity-name ::= 'l' addressor-kind decl-name type     // non-mutable addressor
entity-name ::= 'm' decl-name type     // materializeForSet
entity-name ::= 's' decl-name type     // setter
entity-name ::= 'U' index type         // explicit anonymous closure expression
entity-name ::= 'u' index type         // implicit anonymous closure
entity-name ::= 'w' decl-name type     // willSet
entity-name ::= 'W' decl-name type     // didSet
static ::= 'Z'                         // entity is a static member of a type
decl-name ::= identifier
decl-name ::= local-decl-name
decl-name ::= private-decl-name
local-decl-name ::= 'L' index identifier  // locally-discriminated declaration
private-decl-name ::= 'P' identifier identifier  // file-discriminated declaration
reabstract-signature ::= ('G' generic-signature)? type type
addressor-kind ::= 'u'                 // unsafe addressor (no owner)
```

```
addressor-kind ::= 'O'                // owning addressor (non-native owner)
addressor-kind ::= 'o'                // owning addressor (native owner)
addressor-kind ::= 'p'                // pinning addressor (native owner)

remote-reflection-record ::= 'f' type                 // field descriptor
remote-reflection-record ::= 'a' protocol-conformance  // associated type descriptor
remote-reflection-record ::= 'b' type                 // builtin type descriptor
```

An `entity` starts with a `nominal-type-kind` (`[COPV]`), a substitution (`[Ss]`) of a nominal type, or an `entity-kind` (`[FIiv]`).

An `entity-name` starts with `[AaCcDggis]` or a `decl-name`. A `decl-name` starts with `[LP]` or an `identifier` (`[0-9oX]`).

A `context` starts with either an `entity`, an `extension` (which starts with `[Ee]`), or a `module`, which might be an `identifier` (`[0-9oX]`) or a substitution of a module (`[Ss]`).

A global mangling starts with an `entity` or `[MTWw]`.

If a partial application forwarder is for a static symbol, its name will start with the sequence `_TPA_` followed by the mangled symbol name of the forwarder's destination.

A generic specialization mangling consists of a header, specifying the types and conformances used to specialize the generic function, followed by the full mangled name of the original unspecialized generic symbol.

The first identifier in a `<private-decl-name>` is a string that represents the file the original declaration came from. It should be considered unique within the enclosing module. The second identifier is the name of the entity.

Not all declarations marked `private` declarations will use the `<private-decl-name>` mangling; if the entity's context is enough to uniquely identify the entity, the simple `identifier` form is preferred.

The types in a `<reabstract-signature>` are always non-polymorphic `<impl-function-type>` types.

Direct and Indirect Symbols

```
directness ::= 'd'                        // direct
directness ::= 'i'                        // indirect
```

A direct symbol resolves directly to the address of an object. An indirect symbol resolves to the address of a pointer to the object. They are distinct manglings to make a certain class of bugs immediately obvious.

The terminology is slightly overloaded when discussing offsets. A direct offset resolves to a variable holding the true offset. An indirect offset resolves to a variable holding an offset to be applied to type metadata to get the address of the true offset. (Offset variables are required when the object being accessed lies within a resilient structure. When the layout of the object may depend on generic arguments, these offsets must be kept in metadata. Indirect field offsets are therefore required when accessing fields in generic types where the metadata itself has unknown layout.)

Declaration Contexts

```
context ::= module
context ::= extension
context ::= entity
module ::= substitution                   // other substitution
module ::= identifier                     // module name
module ::= known-module                   // abbreviation
extension ::= 'E' module entity
extension ::= 'e' module generic-signature entity
```

These manglings identify the enclosing context in which an entity was declared, such as its enclosing module, function, or nominal type.

An `extension` mangling is used whenever an entity's declaration context is an extension *and* the entity being extended is in a different module. In this case the extension's module is mangled first, followed by the entity being extended. If the extension and the extended entity are in the same module, the plain `entity` mangling is preferred. If the extension is constrained, the constraints on the extension are mangled in its generic signature.

When mangling the context of a local entity within a constructor or destructor, the non-allocating or non-deallocating variant is used.

Types

```
type ::= 'Bb'                             // Builtin.BridgeObject
type ::= 'BB'                             // Builtin.UnsafeValueBuffer
type ::= 'Bf' natural '_'                 // Builtin.Float<n>
type ::= 'Bi' natural '_'                 // Builtin.Int<n>
type ::= 'BO'                             // Builtin.UnknownObject
type ::= 'Bo'                             // Builtin.NativeObject
type ::= 'Bp'                             // Builtin.RawPointer
type ::= 'Bv' natural type                // Builtin.Vec<n>x<type>
type ::= 'Bw'                             // Builtin.Word
type ::= nominal-type
type ::= associated-type
type ::= 'a' context identifier           // Type alias (DWARF only)
type ::= 'b' type type                    // objc block function type
type ::= 'c' type type                    // C function pointer type
type ::= 'F' throws-annotation? type type  // function type
type ::= 'f' throws-annotation? type type  // uncurried function type
type ::= 'G' type <type>+ '_'             // generic type application
type ::= 'K' type type                    // @auto_closure function type
type ::= 'M' type                         // metatype without representation
type ::= 'XM' metatype-repr type          // metatype with representation
type ::= 'P' protocol-list '_'            // protocol type
type ::= 'PM' type                        // existential metatype without representation
type ::= 'XPM' metatype-repr type         // existential metatype with representation
type ::= archetype
type ::= 'R' type                         // inout
type ::= 'T' tuple-element* '_'           // tuple
type ::= 't' tuple-element* '_'           // variadic tuple
type ::= 'Xo' type                        // @unowned type
type ::= 'Xu' type                        // @unowned(unsafe) type
type ::= 'Xw' type                        // @weak type
type ::= 'XF' impl-function-type          // function implementation type
type ::= 'Xf' type type                   // @thin function type
type ::= 'Xb' type                        // SIL @box type
nominal-type ::= known-nominal-type
nominal-type ::= substitution
nominal-type ::= nominal-type-kind declaration-name
nominal-type-kind ::= 'C'                 // class
nominal-type-kind ::= 'O'                 // enum
nominal-type-kind ::= 'V'                 // struct
declaration-name ::= context decl-name
```

```
archetype ::= 'Q' index                    // archetype with depth=0, idx=N
archetype ::= 'Qd' index index             // archetype with depth=M+1, idx=N
archetype ::= associated-type
archetype ::= qualified-archetype
associated-type ::= substitution
associated-type ::= 'Q' protocol-context     // self type of protocol
associated-type ::= 'Q' archetype identifier // associated type
qualified-archetype ::= 'Qq' index context   // archetype+context (DWARF only)
protocol-context ::= 'P' protocol
tuple-element ::= identifier? type
metatype-repr ::= 't'                      // Thin metatype representation
metatype-repr ::= 'T'                      // Thick metatype representation
metatype-repr ::= 'o'                      // ObjC metatype representation
throws-annotation ::= 'z'                  // 'throws' annotation on function types


type ::= 'u' generic-signature type        // generic type
type ::= 'x'                               // generic param, depth=0, idx=0
type ::= 'q' generic-param-index           // dependent generic parameter
type ::= 'q' type assoc-type-name          // associated type of non-generic param
type ::= 'w' generic-param-index assoc-type-name // associated type
type ::= 'W' generic-param-index assoc-type-name+ '_' // associated type at depth

generic-param-index ::= 'x'                // depth = 0,   idx = 0
generic-param-index ::= index              // depth = 0,   idx = N+1
generic-param-index ::= 'd' index index    // depth = M+1, idx = N
```

`<type>` never begins or ends with a number. `<type>` never begins with an underscore. `<type>` never begins with `d`. `<type>` never begins with `z`.

Note that protocols mangle differently as types and as contexts. A protocol context always consists of a single protocol name and so mangles without a trailing underscore. A protocol type can have zero, one, or many protocol bounds which are juxtaposed and terminated with a trailing underscore.

```
assoc-type-name ::= ('P' protocol-name)? identifier
assoc-type-name ::= substitution
```

Associated types use an abbreviated mangling when the base generic parameter or associated type is constrained by a single protocol requirement. The associated type in this case can be referenced unambiguously by name alone. If the base has multiple conformance constraints, then the protocol name is mangled in to disambiguate.

```
impl-function-type ::=
  impl-callee-convention impl-function-attribute* generic-signature? '_'
  impl-parameter* '_' impl-result* '_'
impl-callee-convention ::= 't'              // thin
impl-callee-convention ::= impl-convention  // thick, callee transferred with given convention
impl-convention ::= 'a'                     // direct, autoreleased
impl-convention ::= 'd'                     // direct, no ownership transfer
impl-convention ::= 'D'                     // direct, no ownership transfer,
                                            // dependent on 'self' parameter
impl-convention ::= 'g'                     // direct, guaranteed
impl-convention ::= 'e'                     // direct, deallocating
impl-convention ::= 'i'                     // indirect, ownership transfer
impl-convention ::= 'l'                     // indirect, inout
impl-convention ::= 'G'                     // indirect, guaranteed
impl-convention ::= 'o'                     // direct, ownership transfer
impl-convention ::= 'z' impl-convention     // error result
impl-function-attribute ::= 'Cb'            // compatible with C block invocation function
impl-function-attribute ::= 'Cc'            // compatible with C global function
impl-function-attribute ::= 'Cm'            // compatible with Swift method
impl-function-attribute ::= 'CO'            // compatible with ObjC method
impl-function-attribute ::= 'Cw'            // compatible with protocol witness
impl-function-attribute ::= 'G'             // generic
impl-function-attribute ::= 'g'             // pseudogeneric
impl-parameter ::= impl-convention type
impl-result ::= impl-convention type
```

For the most part, manglings follow the structure of formal language types. However, in some cases it is more useful to encode the exact implementation details of a function type.

Any `<impl-function-attribute>` productions must appear in the order in which they are specified above: e.g. a pseudogeneric C function is mangled with `Ccg`. `g` and `G` are exclusive and mark the presence of a generic signature immediately following.

Note that the convention and function-attribute productions do not need to be disambiguated from the start of a `<type>`.

Generics

```
protocol-conformance ::= ('u' generic-signature)? type protocol module
```

`<protocol-conformance>` refers to a type's conformance to a protocol. The named module is the one containing the extension or type declaration that declared the conformance.

```
// Property behavior conformance
protocol-conformance ::= ('u' generic-signature)?
                         'b' identifier context identifier protocol
```

Property behaviors are implemented using private protocol conformances.

```
generic-signature ::= (generic-param-count+)? ('R' requirement*)? 'r'
generic-param-count ::= 'z'       // zero parameters
generic-param-count ::= index     // N+1 parameters
requirement ::= type-param protocol-name // protocol requirement
requirement ::= type-param type          // base class requirement
                                         // type starts with [CS]
requirement ::= type-param 'z' type      // 'z'ame-type requirement

// Special type mangling for type params that saves the initial 'q' on
// generic params
type-param ::= generic-param-index       // generic parameter
type-param ::= 'w' generic-param-index assoc-type-name // associated type
type-param ::= 'W' generic-param-index assoc-type-name+ '_'
```

A generic signature begins by describing the number of generic parameters at each depth of the signature, followed by the requirements. As a special case, no `generic-param-count` values indicates a single generic parameter at the outermost depth:

```
urFq_q_                          // <T_0_0> T_0_0 -> T_0_0
u_0_rFq_qd_0_                    // <T_0_0><T_1_0, T_1_1> T_0_0 -> T_1_1
```

Value Witnesses

TODO: document these

```
value-witness-kind ::= 'al'           // allocateBuffer
value-witness-kind ::= 'ca'           // assignWithCopy
value-witness-kind ::= 'ta'           // assignWithTake
value-witness-kind ::= 'de'           // deallocateBuffer
value-witness-kind ::= 'xx'           // destroy
value-witness-kind ::= 'XX'           // destroyBuffer
value-witness-kind ::= 'Xx'           // destroyArray
value-witness-kind ::= 'CP'           // initializeBufferWithCopyOfBuffer
value-witness-kind ::= 'Cp'           // initializeBufferWithCopy
value-witness-kind ::= 'cp'           // initializeWithCopy
value-witness-kind ::= 'TK'           // initializeBufferWithTakeOfBuffer
value-witness-kind ::= 'Tk'           // initializeBufferWithTake
value-witness-kind ::= 'tk'           // initializeWithTake
value-witness-kind ::= 'pr'           // projectBuffer
value-witness-kind ::= 'xs'           // storeExtraInhabitant
value-witness-kind ::= 'xg'           // getExtraInhabitantIndex
value-witness-kind ::= 'Cc'           // initializeArrayWithCopy
value-witness-kind ::= 'Tt'           // initializeArrayWithTakeFrontToBack
value-witness-kind ::= 'tT'           // initializeArrayWithTakeBackToFront
value-witness-kind ::= 'ug'           // getEnumTag
value-witness-kind ::= 'up'           // destructiveProjectEnumData
value-witness-kind ::= 'ui'           // destructiveInjectEnumTag
```

`<value-witness-kind>` differentiates the kinds of value witness functions for a type.

Identifiers

```
identifier ::= natural identifier-start-char identifier-char*
identifier ::= 'o' operator-fixity natural operator-char+

operator-fixity ::= 'p'                   // prefix operator
operator-fixity ::= 'P'                   // postfix operator
operator-fixity ::= 'i'                   // infix operator

operator-char ::= 'a'                     // & 'and'
operator-char ::= 'c'                     // @ 'commercial at'
operator-char ::= 'd'                     // / 'divide'
operator-char ::= 'e'                     // = 'equals'
operator-char ::= 'g'                     // > 'greater'
operator-char ::= 'l'                     // < 'less'
operator-char ::= 'm'                     // * 'multiply'
operator-char ::= 'n'                     // ! 'not'
operator-char ::= 'o'                     // | 'or'
operator-char ::= 'p'                     // + 'plus'
operator-char ::= 'q'                     // ? 'question'
operator-char ::= 'r'                     // % 'remainder'
operator-char ::= 's'                     // - 'subtract'
operator-char ::= 't'                     // ~ 'tilde'
operator-char ::= 'x'                     // ^ 'xor'
operator-char ::= 'z'                     // . 'zperiod'
```

`<identifier>` is run-length encoded: the natural indicates how many characters follow. Operator characters are mapped to letter characters as given. In neither case can an identifier start with a digit, so there's no ambiguity with the run-length.

```
identifier ::= 'X' natural identifier-start-char identifier-char*
identifier ::= 'X' 'o' operator-fixity natural identifier-char*
```

Identifiers that contain non-ASCII characters are encoded using the Punycode algorithm specified in RFC 3492, with the modifications that _ is used as the encoding delimiter, and uppercase letters A through J are used in place of digits 0 through 9 in the encoding character set. The mangling then consists of an `X` followed by the run length of the encoded string and the encoded string itself. For example, the identifier `vergÃ¾enza` is mangled to `X12vergenza_JFa`. (The encoding in standard Punycode would be `vergenza-95a`)

Operators that contain non-ASCII characters are mangled by first mapping the ASCII operator characters to letters as for pure ASCII operator names, then Punycode-encoding the substituted string. The mangling then consists of `Xo` followed by the fixity, run length of the encoded string, and the encoded string itself. For example, the infix operator `Â«+Â»` is mangled to `Xoi7p_qcaDc` (`p_qcaDc` being the encoding of the substituted string `Â«pÂ»`).

Substitutions

```
substitution ::= 'S' index
```

`<substitution>` is a back-reference to a previously mangled entity. The mangling algorithm maintains a mapping of entities to substitution indices as it runs. When an entity that can be represented by a substitution (a module, nominal type, or protocol) is mangled, a substitution is first looked for in the substitution map, and if it is present, the entity is mangled using the associated substitution index. Otherwise, the entity is mangled normally, and it is then added to the substitution map and associated with the next available substitution index.

For example, in mangling a function type `(zim.zang.zung, zim.zang.zung, zim.zippity) -> zim.zang.zoo` (with module `zim` and class `zim.zang`), the recurring contexts `zim`, `zim.zang`, and `zim.zang.zung` will be mangled using substitutions after being mangled for the first time. The first argument type will mangle in long form, `CC3zim4zang4zung`, and in doing so, `zim` will acquire substitution `S_`, `zim.zang` will acquire substitution `S0_`, and `zim.zang.zung` will acquire `S1_`. The second argument is the same as the first and will mangle using its substitution, `S1_`. The third argument type will mangle using the substitution for `zim`, `CS_7zippity`. (It also acquires substitution `S2_` which would be used if it mangled again.) The result type will mangle using the substitution for `zim.zang`, `CS0_3zoo` (and acquire substitution `S3_`). The full function type thus mangles as `fTCC3zim4zang4zungS1_CS_7zippity_CS0_3zoo`.

```
substitution ::= 's'
```

The special substitution `s` is used for the `Swift` standard library module.

Predefined Substitutions

```
known-module ::= 's'                      // Swift
known-module ::= 'SC'                     // C
known-module ::= 'So'                     // Objective-C
known-nominal-type ::= 'Sa'               // Swift.Array
known-nominal-type ::= 'Sb'               // Swift.Bool
known-nominal-type ::= 'Sc'               // Swift.UnicodeScalar
known-nominal-type ::= 'Sd'               // Swift.Float64
known-nominal-type ::= 'Sf'               // Swift.Float32
known-nominal-type ::= 'Si'               // Swift.Int
known-nominal-type ::= 'SV'               // Swift.UnsafeRawPointer
```

```
known-nominal-type ::= 'Sv'               // Swift.UnsafeMutableRawPointer
known-nominal-type ::= 'SP'               // Swift.UnsafePointer
known-nominal-type ::= 'Sp'               // Swift.UnsafeMutablePointer
known-nominal-type ::= 'SQ'               // Swift.ImplicitlyUnwrappedOptional
known-nominal-type ::= 'Sq'               // Swift.Optional
known-nominal-type ::= 'SR'               // Swift.UnsafeBufferPointer
known-nominal-type ::= 'Sr'               // Swift.UnsafeMutableBufferPointer
known-nominal-type ::= 'SS'               // Swift.String
known-nominal-type ::= 'Su'               // Swift.UInt
```

<known-module> and <known-nominal-type> are built-in substitutions for certain common entities. Like any other substitution, they all start with 'S'.

The Objective-C module is used as the context for mangling Objective-C classes as <type>s.

Indexes

```
index ::= '_'                             // 0
index ::= natural '_'                     // N+1
natural ::= [0-9]+
```

<index> is a production for encoding numbers in contexts that can't end in a digit; it's optimized for encoding smaller numbers.