Objective-C Mapping for Identifiers

Objective-C identifiers are derived from Slice identifiers. The exact Objective-C identifier that is generated depends on the context. For types that are nested in modules (and hence have global visibility in Objective-C), the generated Objective-C identifiers are prefixed with their module prefix. Slice identifiers that do not have global visibility (such as operation names and structure members) do not use the module prefix and are preserved without change. For example, consider the following Slice definition:

Slice

```
["objc:prefix:EX"]
module Example {
    struct Point {
        double x;
        double y;
    };
};
```

This maps to the following Objective-C definition:

Objective-C

Slice

```
@interface EXPoint : NSObject <NSCopying>
{
    @private
        ICEDouble x;
        ICEDouble y;
}
@property(nonatomic, assign) ICEDouble x;
@property(nonatomic, assign) ICEDouble y;
// More definitions here...
@end
```

If a Slice identifier is the same as an Objective-C keyword, the corresponding Objective-C identifier has an underscore suffix. For example, Slice while maps to Objective-C while_.

In some cases, the Objective-C mapping generates more than one identifier for a given Slice construct. For example, an interface Intf generates the identifiers EXIntf and EXIntfPrx. If a Slice identifier happens to be an Objective-C keyword, the underscore suffix applies only where necessary, so an interface while generates EXWhile and EXWhilePrx.

Note that Slice operation and member names can clash with the name of an inherited method, property, or instance variable. For example:

```
exception Failed {
   string reason; // Clashes with NSException
};
```

This is a legal Slice definition. However, the generated exception class derives from *NSException*, which defines a reason method. To avoid hiding the method in the base class, the generated exception class maps the Slice reason member to the Objective-C property reason_, just as it would for a keyword.

This escape mechanism applies to all generated classes that directly or indirectly derive from NSObject or NSException.

Internal Identifiers in Objective-C

Any methods that contain two or more adjacent underscores (such as read_ and op____) are internal to the Objective-C mapping implementation and are not for use by application code.

See Also

- Objective-C Mapping for Modules
 Objective-C Mapping for Built-In Types
 Objective-C Mapping for Enumerations
 Objective-C Mapping for Structures
 Objective-C Mapping for Dictionaries
 Objective-C Mapping for Constants
 Objective-C Mapping for Exceptions
 Objective-C Mapping for Interfaces

2