Parameter Passing in Objective-C

This page shows how to implement parameters for Slice operations in Objective-C.

On this page:

Implementing Parameters for Slice Operations in Objective-C

For each parameter of a Slice operation, the Objective-C mapping generates a corresponding parameter for the method in the skeleton. In addition, every method has an additional, trailing parameter of type `ICECurrent`. For example, the `name` operation of the `Node` interface has no parameters, but the `name` method of the `Node` skeleton protocol has a single parameter of type `ICECurrent`. We will ignore this parameter for now.

Parameter passing on the server side follows the rules for the client side (with one exception):

- In-parameters and the return value are passed by value or by pointer, depending on the parameter type.
- Out-parameters are passed by pointer-to-pointer.

The exception to the client-side rules concerns types that come in mutable and immutable variants (strings, sequences, and dictionaries). For these, the server-side mapping passes the mutable variant where the client-side passes the immutable variant, and vice versa.

To illustrate the rules, consider the following interface that passes string parameters in all possible directions:

Slice

```
interface Intf {
    string op(string sin, out string sout);
};
```

The generated skeleton protocol for this interface looks as follows:

Objective-C

```
@protocol EXIntf <ICEObject>
-(NSString *) op:(NSMutableString *)sin
                 sout:(NSString **)sout
                 current:(ICECurrent *)current;
@end
```

As you can see, the in-parameter `sin` is of type `NSMutableString`, and the out parameter and return value are passed as `NSString` (the opposite of the client-side mapping). This means that in-parameters are passed to the servant as their mutable variant, and it is safe for you to modify such in-parameters. This is useful, for example, if a client passes a sequence to the operation, and the operation returns the sequence with a few minor changes. In that case, there is no need for the operation implementation to copy the sequence. Instead, you can simply modify the passed sequence as necessary and return the modified sequence to the client.

Here is an example implementation of the operation:

Objective-C

```
-(NSString *) op:(NSMutableString *)sin
                 sout:(NSString **)sout
                 current:(ICECurrent *)current
{
    printf("%s\n", [sin UTF8String]); // In-params are initialized
    *sout = [sin appendString:@"appended"]; // Assign out-param
    return @"Done";                         // Return a string
}
```

Memory Management for Operations in Objective-C

To avoid leaking memory, you must be aware of how the Ice run time manages memory for operation implementations:

- In-parameters are passed to the servant already autoreleased.
- Out-parameters and return values must be returned by the servant as autoreleased values.

This follows the usual Objective-C convention: the allocator of a value is responsible for releasing it. This is what the Ice run time does for in-parameters, and what you are expected to do for out-parameters and return values. These rules also mean that it is safe to return an in-parameter as an out-parameter or return value. For example:

Objective-C

```
-(NSString *) op:(NSMutableString *)sin
                 sout:(NSString **)sout
                 current:(ICECurrent *)current
{
    *sout = sin; // Works fine.
    return sin;  // Works fine.
}
```

The Ice run time creates and releases a separate autorelease pool for each invocation. This means that the memory for parameters is reclaimed as soon as the run time has marshaled the operation results back to the client.

See Also

- Server-Side Objective-C Mapping for Interfaces
- Raising Exceptions in Objective-C
- The Current Object