Adding Validation

The key-value coding protocol defines methods for validating properties by key or key path. The default implementation of these methods in turn rely on you to define methods following naming patterns similar to those used for accessor methods. Specifically, you provide a `validate<Key>:error:` method for any property with the name `key` that you want to validate. The default implementation searches for this in response to a key-coded `validateValue:forKey:error:` message.

If you don't supply a validation method for a property, the default implementation of the protocol assumes validation succeeds for that property, regardless of the value. This means that you opt in to validation on a property-by-property basis.

> NOTE
>
> You typically use the validation described here only in Objective-C. In Swift, property validation is more idiomatically handled by relying on compiler support for optionals and strong type checking, while using the built-in `willSet` and `didSet` property observers to test any run-time API contracts, as described in the Property Observers section of *The Swift Programming Language (Swift 3.0.1)*.

Implementing a Validation Method

When you do provide a validation method for a property, that method receives two parameters by reference: the value object to validate and the `NSError` used to return error information. As a result, your validation method can take one of three actions:

- When the value object is valid, return `YES` without altering the value object or the error.
- When the value object isn't valid, and you either can't or don't want to provide a valid alternative, set the error parameter to an `NSError` object that indicates the reason for failure and return `NO`.

> IMPORTANT
>
> Always test that an error reference is not `NULL` before trying to set it.

- When the value object isn't valid, but you know of a valid alternative, create the valid object, assign the value reference to the new object, and return `YES` without modifying the error reference. If you provide another value, always return a new object rather than modifying the one being validated, even if the original object is mutable.

Listing 11-1 demonstrates a validation method for a `name` string property that ensures that the value object is not `nil` and that the name is a minimum length. This method does not substitute another value if validation fails.

Listing 11-1 Validation method for the name property

On This Page

```
- (BOOL
    if ((*ioValue == nil) || ([(NSString *)*ioValue length] < 2)) {
        if (outError != NULL) {
            *outError = [NSError errorWithDomain:PersonErrorDomain
                                            code:PersonInvalidNameCode
                                        userInfo:@{ NSLocalizedDescriptionKey
                                                    : @"Name too short" }];
        }
        return NO;
    }
    return YES;
}
```

Validation of Scalar Values

Validation methods expect the value parameter to be an object, and as a result, values for non-object properties are wrapped in an `NSValue` or `NSNumber` object, as discussed in Representing Non-Object Values. The example in Listing 11-2 demonstrates a validation method for the scalar property `age`. In this case, one potential invalid condition, namely a `nil` `age` value, is handled by creating a valid value set to zero, and returning `YES`. You might also handle this particular condition in your `setNilValueForKey:` override, because a user of your class might not invoke the validation method.

Listing 11-2 Validation method for a scalar property

```
- (BOOL)validateAge:(id *)ioValue error:(NSError * __autoreleasing *)outError {
    if (*ioValue == nil) {
        // Value is nil: Might also handle in setNilValueForKey
        *ioValue = @(0);
    } else if ([*ioValue floatValue] < 0.0) {
        if (outError != NULL) {
            *outError = [NSError errorWithDomain:PersonErrorDomain
                                            code:PersonInvalidAgeCode
                                        userInfo:@{ NSLocalizedDescriptionKey
                                                    : @"Age cannot be negative" }];
        }
        return NO;
    }
    return YES;
}
```


On This Page