Dealing with Errors

Almost every app encounters errors. Some of these errors will be outside of your control, such as running out of disk space or losing network connectivity. Some of these errors will be recoverable, such as invalid user input. And, while all developers strive for perfection, the occasional programmer error may also occur.

If you're coming from other platforms and languages, you may be used to working with exceptions for the majority of error handling. When you're writing code with Objective-C, exceptions are used solely for programmer errors, like out-of-bounds array access or invalid method arguments. These are the problems that you should find and fix during testing before you ship your app.

All other errors are represented by instances of the `NSError` class. This chapter gives a brief introduction to using `NSError` objects, including how to work with framework methods that may fail and return errors. For further information, see *Error Handling Programming Guide*.

Use NSError for Most Errors

Errors are an unavoidable part of any app's lifecycle. If you need to request data from a remote web service, for example, there are a variety of potential problems that may arise, including:

- No network connectivity
- The remote web service may be inaccessible
- The remote web service may not be able to serve the information you request
- The data you receive may not match what you were expecting

Sadly, it's not possible to build contingency plans and solutions for every conceivable problem. Instead, you must plan for errors and know how to deal with them to give the best possible user experience.

Some Delegate Methods Alert You to Errors

If you're implementing a delegate object for use with a framework class that performs a certain task, like downloading information from a remote web service, you'll typically find that you need to implement at least one error-related method. The `NSURLConnectionDelegate` protocol, for example, includes a `connection:didFailWithError:` method:

```
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error;
```

If an error occurs, this delegate method will be called to provide you with an `NSError` object to describe the problem.

An `NSError` object contains a numeric error code, domain and description, as well as other relevant information packaged in a user info dictionary.

Rather than making the requirement that every possible error have a unique numeric code, Cocoa and Cocoa Touch errors are divided into domains. If an error occurs in an `NSURLConnection`, for example, the `connection:didFailWithError:` method above will provide an error from `NSURLErrorDomain`.

The error object also includes a localized description, such as "A server with the specified hostname could not be found."

Some Methods Pass Errors by Reference

Some Cocoa and Cocoa Touch API pass back errors by reference. As an example, you might decide to store the data that you receive from a web service by writing it to disk, using the `NSData` method `writeToURL:options:error:`. The last parameter to this method is a reference to an `NSError` pointer:

```
- (BOOL)writeToURL:(NSURL *)aURL
           options:(NSDataWritingOptions)mask
             error:(NSError **)errorPtr;
```

Before you call this method, you'll need to create a suitable pointer so that you can pass its address:

```
    NSError *anyError;
    BOOL success = [receivedData writeToURL:someLocalFileURL
                                    options:0
                                      error:&anyError];
    if (!success) {
        NSLog(@"Write failed with error: %@", anyError);
        // present error to user
    }
```

If an error occurs, the `writeToURL:...` method will return `NO`, and update your `anyError` pointer to point to an error object describing the problem.

When dealing with errors passed by reference, it's important to test the return value of the method to see whether an error occurred, as shown above. Don't just test to see whether the error pointer was set to point to an error.

Tip: If you're not interested in the error object, just pass `NULL` for the `error:` parameter.

Recover if Possible or Display the Error to the User

The best user experience is for your app to recover transparently from an error. If you're making a remote web request, for example, you might try making the request again with a different server. Alternatively, you might need to request additional information from the user such as valid username or password credentials before trying again.

If it's not possible to recover from an error, you should alert the user. If you're developing with Cocoa Touch for iOS, you'll need to create and configure a `UIAlertView` to display the error. If you're developing with Cocoa for OS X, you can call `presentError:` on any `NSResponder` object (like a view, window or even the application object itself) and the error will propagate up the responder chain for further configuration or recovery. When it reaches the application object, the application presents the error to the user through an alert panel.

For more information on presenting errors to the user, see Displaying Information From Error Objects.

Generating Your Own Errors

In order to create your own `NSError` objects you'll need to define your own error domain, which should be of the form:

```
com.companyName.appOrFrameworkName.ErrorDomain
```

You'll also need to pick a unique error code for each error that may occur in your domain, along with a suitable description, which is stored in the user info dictionary for the error, like this:

```
    NSString *domain = @"com.MyCompany.MyApplication.ErrorDomain";
    NSString *desc = NSLocalizedString(@"Unable to…", @"");
    NSDictionary *userInfo = @{ NSLocalizedDescriptionKey : desc };

    NSError *error = [NSError errorWithDomain:domain
                                         code:-101
                                     userInfo:userInfo];
```

This example uses the `NSLocalizedString` function to look up a localized version of the error description from a `Localizable.strings` file, as described in Localizing String Resources.

If you need to pass back an error by reference as described earlier, your method signature should include a parameter for a pointer to a pointer to an `NSError` object. You should also use the return value to indicate success or failure, like this:

```
- (BOOL)doSomethingThatMayGenerateAnError:(NSError **)errorPtr;
```

If an error occurs, you should start by checking whether a non-`NULL` pointer was provided for the error parameter before you attempt to dereference it to set the error, before returning `NO` to indicate failure, like this:

```
- (BOOL)doSomethingThatMayGenerateAnError:(NSError **)errorPtr {
    ...
    // error occurred
    if (errorPtr) {
        *errorPtr = [NSError errorWithDomain:...
                                        code:...
                                    userInfo:...];
    }
    return NO;
}
```

Exceptions Are Used for Programmer Errors

Objective-C supports exceptions in much the same way as other programming languages, with a similar syntax to Java or C++. As with `NSError`, exceptions in Cocoa and Cocoa Touch are objects, represented by instances of the `NSException` class

If you need to write code that might cause an exception to be thrown, you can enclose that code inside a try-catch block:

```
    @try {
        // do something that might throw an exception
    }
    @catch (NSException *exception) {
        // deal with the exception
    }
    @finally {
        // optional block of clean-up code
        // executed whether or not an exception occurred
```

```
    }
```

If an exception is thrown by the code inside the `@try` block, it will be caught by the `@catch` block so that you can deal with it. If you're working with a low-level C++ library that uses exceptions for error handling, for example, you might catch its exceptions and generate suitable `NSError` objects to display to the user.

If an exception is thrown and not caught, the default uncaught exception handler logs a message to the console and terminates the application.

You should not use a try-catch block in place of standard programming checks for Objective-C methods. In the case of an `NSArray`, for example, you should always check the array's `count` to determine the number of items before trying to access an object at a given index. The `objectAtIndex:` method throws an exception if you make an out-of-bounds request so that you can find the bug in your code early in the development cycle—you should avoid throwing exceptions in an app that you ship to users.

For more information on exceptions in Objective-C applications, see *Exception Programming Topics*.
