Conceptual Overview

Block objects provide a way for you to create an ad hoc function body as an expression in C, and C-derived languages such as Objective-C and C++. In other languages and environments, a block object is sometimes also called a “closure”. Here, they are typically referred to colloquially as “blocks”, unless there is scope for confusion with the standard C term for a block of code.

Block Functionality

A block is an anonymous inline collection of code that:

- Has a typed argument list just like a function
- Has an inferred or declared return type
- Can capture state from the lexical scope within which it is defined
- Can optionally modify the state of the lexical scope
- Can share the potential for modification with other blocks defined within the same lexical scope
- Can continue to share and modify state defined within the lexical scope (the stack frame) after the lexical scope (the stack frame) has been destroyed

You can copy a block and even pass it to other threads for deferred execution (or, within its own thread, to a runloop). The compiler and runtime arrange that all variables referenced from the block are preserved for the life of all copies of the block. Although blocks are available to pure C and C++, a block is also always an Objective-C object.

Usage

Blocks represent typically small, self-contained pieces of code. As such, they’re particularly useful as a means of encapsulating units of work that may be executed concurrently, or over items in a collection, or as a callback when another operation has finished.

Blocks are a useful alternative to traditional callback functions for two main reasons:

1. They allow you to write code at the point of invocation that is executed later in the context of the method implementation.

   Blocks are thus often parameters of framework methods.

2. They allow access to local variables.

   Rather than using callbacks requiring a data structure that embodies all the contextual information you need to perform an operation, you simply access local variables directly.