Background Execution

* The Background Execution is a state of an application and the App goes to this state when, the user is not actively using the app, and then system moves it to the background state.
* The background state for many apps, is just a brief stop on the way to the app is being suspended.
* When we suspend the app it will improve the battery life and it also allows the system to devote important system resources to the new foreground app that has drawn the user’s attention.
* In iOS most of the apps can move to the suspended state easily enough but there are also legitimate reasons for apps to continue running in the background.
* For example a hiking app might want to track the user’s position over time so that it can display that motion overlaid on top of a hiking map. An audio app might need to continue playing music over the lock screen. Other apps might want to download content in the background so that it can minimize the delay in presenting that content to the user.
* When we find it necessary to keep the app running in the background, iOS helps us to do so efficiently and without draining system resources or the user’s battery. The techniques offered by iOS fall into three categories:
* Apps that start a short task in the foreground can ask for time to finish that task when the app moves to the background.
* Apps that initiate downloads in the foreground can hand off management of those downloads to the system, thereby allowing the app to be suspended or terminated while the download continues.
* Apps that need to run in the background to support specific types of tasks can declare their support for one or more background execution modes.
* It is advisable that trying to avoiding doing any background work unless doing so improves the overall user experience.
* In iOS an app might move to the background because of many reasons like user launched a different app or because the user locked the device and is not using it right now.
* In above mentioned situations, the user is signaling to the app that it does not need to be doing any meaningful work right now.
* So continuing to run the App in such conditions will only drain the device’s battery and might lead the user to force quit your app altogether.
* So we must be care full about the work that we do in the background and avoid it when we can avoid it.

## Executing Finite-Length Task

* Apps moving to the background are expected to put themselves into an inactive state as quickly as possible so that they can be suspended by the system.
* If the app is in the middle of a task and needs a little extra time to complete that task, it can call the following methods :
* [**beginBackgroundTaskWithName:expirationHandler:**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/occ/instm/UIApplication/beginBackgroundTaskWithName:expirationHandler:) or
* [**beginBackgroundTaskWithExpirationHandler**:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/occ/instm/UIApplication/beginBackgroundTaskWithExpirationHandler:) method of the UIApplication object to request some additional execution time
* By calling either of these methods mentioned above delays the suspension of app temporarily, giving it a little extra time to finish its work.
* Upon completion of that work, the app must call the **endBackgroundTask**: method to let the system to know that it is finished and can be suspended.
* Each call to the beginBackgroundTaskWithName: expirationHandler: or beginBackgroundTaskWithExpirationHandler: method generates a unique token to associate with the corresponding task.
* When the app completes its task, it must call the **endBackgroundTask:** method with the corresponding token to let the system know that the task is complete.
* When failure to call the **endBackgroundTask**: method for a background task will result in the termination of the app.
* If we provided an expiration handler when starting the task, the system calls that handler and gives the one last chance to end the task and avoid termination.
* We do not need to wait until the app moves to the background to designate background tasks.
* The more useful design is to call the **beginBackgroundTaskWithName:** **expirationHandler**: or **beginBackgroundTaskWithExpirationHandler:** method before starting a task and call the **endBackgroundTask:** method as soon as we finish. We can even follow this pattern while your app is executing in the foreground.
* The Example below shows how to start a long-running task when the app transitions to the background.
* In this example, the request to start a background task includes an expiration handler just in case the task takes too long.
* The task itself is then submitted to a dispatch queue for asynchronous execution so that the **applicationDidEnterBackground**: method can return normally.
* The use of blocks simplifies the code needed to maintain references to any important variables, such as the background task identifier.
* The **bgTask** variable is a member variable of the class that stores a pointer to the current background task identifier and is initialized prior to its use in this method.

**Example:**

- (void)applicationDidEnterBackground:(UIApplication \*)application

{

bgTask = [application beginBackgroundTaskWithName:@"MyTask" expirationHandler:^{

// Clean up any unfinished task business by marking where you

// stopped or ending the task outright.

[application endBackgroundTask:bgTask];

bgTask = UIBackgroundTaskInvalid;

}];

// Start the long-running task and return immediately.

dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{

// Do the work associated with the task, preferably in chunks.

[application endBackgroundTask:bgTask];

bgTask = UIBackgroundTaskInvalid;

});

}

**Note:** It is recommended to provide an expiration handler when starting a task, but if we want to know how much time the app has left to run, get the value of the [**backgroundTimeRemaining**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/occ/instp/UIApplication/backgroundTimeRemaining) property of **UIApplication**

* In our own expiration handlers, we can include additional code needed to close out the task.
* However, any code we include must not take too long to execute because, by the time our expiration handler is called, our app is already very close to its time limit. For this reason, perform only minimal cleanup of our state information and end the task.

## Downloading Content in the Background

* Downloading is most common thing in every App. When user starts downloading files, apps should use an **NSURLSession** object to starts downloads so that the system can take control of the download process in case when the app is suspended or terminated.
* When we configure an **NSURLSession** object for background transfers, then the system manages those transfers in a separate process and reports status back to the app in the usual way.
* If the app is terminated while transfers are ongoing, the system continues the transfers in the background and launches your app (as appropriate) when the transfers finish or when one or more tasks need the app’s attention.
* To support in the background transfers, we must configure the **NSURLSession** object appropriately.
* To configure the session, it is mandatory to create **NSURLSessionConfiguration** object and set several properties to appropriate values. After that we have to pass that configuration object to the appropriate initialization method of **NSURLSession** when creating the session.

**The process for creating a configuration object that supports background downloads is as follows:**

1. Create the configuration object using the [**backgroundSessionConfigurationWithIdentifier**:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLSessionConfiguration_class/index.html#//apple_ref/occ/clm/NSURLSessionConfiguration/backgroundSessionConfigurationWithIdentifier:) method of **NSURLSessionConfiguration**.
2. Set the value of the configuration object’s [**sessionSendsLaunchEvents**](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLSessionConfiguration_class/index.html#//apple_ref/occ/instp/NSURLSessionConfiguration/sessionSendsLaunchEvents) property to [**YES**](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/index.html#//apple_ref/doc/c_ref/YES).
3. If the app starts transfers while it is in the foreground, it is recommend that we also set the [**discretionary**](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLSessionConfiguration_class/index.html#//apple_ref/occ/instp/NSURLSessionConfiguration/discretionary) property of the configuration object to [**YES**](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/index.html#//apple_ref/doc/c_ref/YES).
4. Configure any other properties of the configuration object as appropriate.
5. Use the configuration object to create your **NSURLSession** object.

* Once we configured, the **NSURLSession** object seamlessly hands off upload and download tasks to the system at appropriate times.
* When the tasks finish while the app is still running (either in the foreground or the background), then the session object notifies its delegate in the usual way.
* If tasks have not yet finished and the system terminates the app, the system automatically continues managing the tasks in the background.
* If the user terminates the app, then the system cancels any pending tasks.
* When all of the tasks associated with a background session are complete, then the system re launches the terminated app (assuming that the sessionSendsLaunchEvents property was set to [**YES**](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/index.html#//apple_ref/doc/c_ref/YES) and that the user did not force quit the app) and calls the app delegate’s[**application:handleEventsForBackgroundURLSession:completionHandler:**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/application:handleEventsForBackgroundURLSession:completionHandler:) method.
* The system may also re launch the app to handle authentication challenges or other task-related events that require the app’s attention.
* In the implementation of the delegate method, use the provided identifier to create a new **NSURLSessionConfiguration** and **NSURLSession** object with the same configuration as before.
* The system reconnects the new session object to the previous tasks and reports their status to the session object’s delegate.

References:

* https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html#//apple\_ref/doc/uid/TP40007072-CH4-SW1