DOWNLOAD PDF APPLICATION LIFECYCLE IN IOS

Chapter 1 : The iOS Application Lifecycle â€“ Hacker Noon

The App Life Cycle. Apps are a sophisticated interplay between your custom code and the system frameworks. The system frameworks provide the basic infrastructure that all apps need to run, and you provide the code required to customize that infrastructure and give the app the look and feel you want.

This article explains the Life cycle of a typical iOS application and also talks about the various states of the application. After going through this, one can get a fair idea about various transition states of App. While designing an application for iOS4, handling of new transitional states must be taken in consideration. Application Life Cycle on iOS The application life cycle constitutes the sequence of events that occurs between the launch and termination of application. In iOS, the user launches application by tapping its icon on the Home screen. There is a predefined sequence of events that occurs from the time of application start up to the time it quits. In iOS 4 and later, Third Party Applications now remain resident in memory by default after they are quit. This means that subsequent launches of the application may not always involve starting the application from scratch. Hence applications designed on iOS4 should be able to handle several different state transitions. Here is the depiction of events on iOS4: Till iOS 3, At the time of quit message application use to be terminated, here is the depiction of Applications life cycle till iOS 3: There were only two possible states of third party application in iOS 3: Active Not running But because of background applications support in iOS 4, following are the possible state of any third party application: The application has not been launched. The application is running in background but not receiving events. The application is running in foreground and is receiving events. Most applications enter this state briefly on their way to being suspended. However, an application that requests extra execution time may remain in this state for a period of time. In addition, an application being launched directly into the background enters this state instead of the inactive state. The application is in the background but is not executing code. The system moves application to this state automatically and at appropriate times. While suspended, an application is essentially freeze-dried in its current state and does not execute any code. During low-memory conditions, the system purges suspended applications without notice to make more space for the foreground application. After knowing these facts, there could be some obvious questions like When the user launches an application that currently resides in the background, the system moves it to the inactive state and then to the active state. This results in calls to the following methods of the application delegate: When an application opts out, it cycles between the not running, inactive, and active states and never enters the background or suspended states. Opting out of background execution is strongly discouraged but may be preferable under certain conditions. When the Active application is interrupted by an incoming phone call, SMS, or calendar notification, the application moves to the inactive state temporarily. It remains in this state until the user decides whether to accept or ignore the interruption. If the user ignores the interruption, the application is reactivated. If the user accepts the interruption, the application moves into the suspended state. History 29th October,

Chapter 2 : theinnatdunvilla.com Application Life Cycle

Every iOS Developer, should know application lifecycle. Application lifecycle helps to understand overall app behaviour. The main point of entry into iOS apps is UIApplicationDelegate.

Background and Suspended states. At any given moment, your app fells into one of the above states. The iOS operating system manages the application states, but the app is responsible for handling user-experience through these state transitions. Not Running â€” Either the application has not started yet or was running and has been terminated by the system. Inactive â€” An application is running in the Foreground but is not receiving any events. This could happen in case a Call or Message is received. An application could also stay in this state while in transition to a different state. Active â€" An application is running in the Foreground and receiving the events. This is the normal mode for the Foreground apps. The only way to go to or from the Active state is through the Inactive state. Background â€" An application is running in the background and executing the code. Freshly launching apps directly enter into In-Active state and then to Active state. In addition, an application being launched directly into the background enters this state instead of the inactive state. Suspended â€" An application is in the background but is not executing the code. The system moves the application to this state automatically and does not notify. In case of low memory, the system may purge suspended application without notice to make free space for the foreground application. Usually after 5 secs spent in the background, apps will transition to Suspend state, but we can extend the time if app needs. UIApplication object defines some methods which are called or will be responded to some of the above states which are most important, to let us work on those transition states regarding our app functionalities. Let us see in the following. As soon as application has successfully initiated launch process: You can execute your code if the launch was successful. You can finalise your interface and can provide the root ViewController to the window. You should use this method to restart any tasks that were paused or not yet started while the app was inactive. This can happen in case of any interruptions such as an incoming phone call or SMS message or Calendar alerts or when the user quits the app. You should use this method to pause any ongoing tasks or disable timers etc. You have approximately five seconds to perform any tasks and return back. In case you need additional time, you can request additional execution time from the system by calling beginBackgroundTask expirationHandler: If the method does not return before time runs out your app is terminated and purged from memory. You should use this to undo any change you made to your app upon entering the background. You should use this method to perform any final clean-up task. If the method does not return before time expires, the system may kill the process altogether. This method may be called in situations where the app is running in the background not suspended and the system needs to terminate it for some reason. For example the system will not call applicationWillTerminate when the device reboots.

Chapter 3 : â€ŽLife Cycle - Track Your Time on the App Store

Learn how to build native applications for the iPhone and iPad using the iOS 7 SDK. Simon Allardice shows you how to work with Xcode 5, the most popular framework for developing, debugging, and.

Forms is a platform to develop cross platform mobile applications by using XAML for front-end and C for backend of the application. Forms applications, all your code is shared. So, you can develop native Android, iOS and Windows apps. If you are going to make your first Xamarin app, this link can be helpful to you. Life cycle of Xamarin. Forms application When you create your Xamarin. Forms application, you will see 4 projects Portable Project This is the project where you are going to code 95 percent of your application code and this code is shared in all three platforms. Android This is the Android project where you are going to set Android application icon and splash screen. And all the other code comes from the potable project. Universal Windows Universal Windows Platform is the application platform to build both, Windows mobile and Windows desktop applications. You can see all four projects here. Start here with Xamarin. Xamarin Forms application life cycle consists of three virtual methods that are overridden to handle lifecycle methods. These methods are present in App. The three methods include- OnSleep OnResume These three methods are called when application is in start, sleep or resume state respectively. There is no method for application termination. Application is terminated from OnSleep method without any additional notification. You can see all of these methods in App. On Start Method OnStart method calls when your application is started at first. When the application starts, it reads all the code written in OnStart method.

Chapter 4 : Applicationâ€™s Life Cycle in iOS4 - CodeProject

The iOS Application Lifecycle. A pplication Life Cycle is very important to understand for all the iOS Developers, who want to make enriched, immersive and smooth User experience.

Not running Inactive e. If you are not using storyboards you will need to set up the window and root view controller of your application in application: Remember to also set the initial view controller on the storyboard ViewController lifecycle init coder: View controller are usually created from storyboards. When this is the case,init coder: This is not used often, so you can ignore the parameter. If you are curious, serialization transforms an object in a byte stream that you can save on disk or send over the network. During the initalization phase of a view controller, you usually allocate the resources that the view controller will need during its lifetime. These include model objects or other auxiliary controllers, like network controllers. Previous view controllers might also pass these objects to the current one, so you do not always need to instantiate them in every view controller. Things to take care of: Beware that the view of the view controller has still not been instantiated at this point. If you try to access it through the view property of UIViewController, the loadView method will be called. This might happen, for example, to work in a larger team where different members need to be change view controller interfaces without affecting the work of others. You also might have a project that was created when storyboards did not exist yet, so every view controller had its own nib file. Keep in mind that if your main storyboard starts getting too big, you can split it into more storyboards. You do not need to move every view controller in a separate nib file. If you create a view controller from a nib file, this initializer is called instead of init coder: You override this method only in case you want to build the whole interface for the view controller from code. If you are working with storyboards or nib files you do not have to anything with this method and you can ignore it. Its implementation in UIVIewController loads the interface from the interface file and connects all the outlets and actions for you. It is common to use this method to populate the user interface of the view controller with data before the user sees it. It is also a good place where to start some background activity where you need to have the user interface in place at the end. A common case are network calls that you need to do only once when the screen is loaded. Good place to init and setup objects used in the viewController. Important thing to remember: This method is called only once in the lifetime of a view controller, so you use it for things that need to happen only once. If you need to perform some task every time a view controller comes on screen, then you need the next method. Keep in mind that in this lifecycle step the view bounds are not final. You override this method for tasks that need you need to repeat every time a view controller comes on screen. This method can be called multiple times for the same instance of a view controller. Notifies the view controller that its view is about to be added to a view hierarchy Application: Usually you use this method to update the user interface with data that might have changed while the view controller was not on the screen. You can also prepare the interface for animations you want to trigger when the view controller appears. Code you need to execute only once should go into an initializer or viewDidLoad. In this step the view has bounds defined but the orientation is not applied viewWillLayoutSubViews: Called to notify the view controller that its view has just laid out its subviews. Make additional changes here after the view lays out its subviews. This method gets called after the view controller appears on screen. You can use it to start animations in the user interface, to start playing a video or a sound, or to start collecting data from the network. Before the transition to the next view controller happens and the origin view controller gets removed from screen, this method gets called. You rarely need to override this method since there are few common tasks that need to be performed at this point, but you might need it. After a view controller gets removed from the screen, this method gets called. You usually override this method to stop tasks that are should not run while a view controller is not on screen. For example, you can stop listening to notifications, observing other objects properties, monitoring the device sensors or a network call that is not needed anymore. You usually override deinit to clean resources that the view controller has allocated that are not freed by ARC. You can also stop tasks you did not stop in the previous method because you wanted to keep them in the background. A view controller going out of the screen does not mean that it will be deallocated afterwards. Many containers keep view controllers in memory. For example, as you go deeper in a navigation controller, all the previous view controllers stay in memory. A navigation controller releases view controllers only when navigating back up the hierarchy. For this reason, you have to keep in mind that a view controller that is not on the screen still works normally and receives notifications. When the memory starts to fill up, iOS does not use its limited hard disk space to move data out of the memory like a computer does. For this reason you are responsible to keep the memory footprint of your app low. If your app starts using too much memory, iOS will notify it. Since view controllers perform resource management, these notifications are delivered to them through this method. In this way you can take actions to free some memory. Keep in mind that if you ignore memory warnings and the memory used by your app goes over a certain threshold, iOS will end your app. This will look like a crash to the user and should be avoided.

Chapter 5 : android - What the equivalent of activity life cycle in iOS? - Stack Overflow

iOS Application lifecycle is the most important and the most basic flow which every iOS developer should understand in depth to make sure he uses the power of lifecycle methods coherently as apple directs.

Next Previous Strategies for Handling App State Transitions For each of the possible runtime states of an app, the system has different expectations while your app is in that state. When state transitions occur, the system notifies the app object, which in turn notifies its app delegate. You can use the state transition methods of the UIApplicationDelegate protocol to detect these state changes and respond appropriately. For example, when transitioning from the foreground to the background, you might write out any unsaved data and stop any ongoing tasks. The following sections offer tips and guidance for how to implement your state transition code. Check the contents of the launch options dictionary for information about why the app was launched, and respond appropriately. Show your app window from your application: UIKit delays making the window visible until after the application: For apps that support state restoration, the state restoration machinery restores your interface to its previous state between calls to the application: Apps are expected to launch, initialize themselves, and start handling events in less than 5 seconds. If an app does not finish its launch cycle in a timely manner, the system kills it for being unresponsive. Thus, any tasks that might slow down your launch such as accessing the network should be scheduled performed on a secondary thread. The Launch Cycle When your app is launched, it moves from the not running state to the active or background state, transitioning briefly through the inactive state. The default main function that comes with your Xcode project promptly hands control over to the UIKit framework, which does most of the work in initializing your app and preparing it to run. Figure shows the sequence of events that occurs when an app is launched into the foreground, including the app delegate methods that are called. The main difference is that instead of your app being made active, it enters the background state to handle the event and may be suspended at some point after that. When the app is launched into the foreground, this property contains the value UIApplicationStateInactive. When the app is launched into the background, the property contains the value UIApplicationStateBackground instead. You can use this difference to adjust the launch-time behavior of your delegate methods accordingly. When an app is launched so that it can open a URL, the sequence of startup events is slightly different from those shown in Figure and Figure Launching in Landscape Mode Apps that uses only landscape orientations for their interface must explicitly ask the system to launch the app in that orientation. Normally, apps launch in portrait mode and rotate their interface to match the device orientation as needed. For apps that support both portrait and landscape orientations, always configure your views for portrait mode and then let your view controllers handle any rotations. If, however, your app supports landscape but not portrait orientations, perform the following tasks to make it launch in landscape mode initially: Lay out your views in landscape mode and make sure that their layout or autosizing options are set correctly. Apps should always use view controllers to manage their window-based content. Using this key is equivalent to calling the setStatusBarOrientation: You can further subdivide this directory to organize your data files as needed. You must not modify any files inside your app bundle. Copying those files to the Application Support directory or another writable directory in your sandbox and modifying them there is the only way to use such files safely. For more information about where to put app-related data files, see File System Programming Guide. Your app continues to run in the foreground, but it does not receive touch events from the system. It does continue to receive notifications and other types of events, such as accelerometer events, though. In response to this change, your app should do the following in its applicationWillResignActive: Save data and any relevant state information. Stop timers and other periodic tasks. Stop any running metadata queries. Do not initiate any new tasks. Pause movie playback except when playing back over AirPlay. Enter into a pause state if your app is a game. Suspend any dispatch queues or operation queues executing non-critical code. You can continue processing network requests and other time-sensitive background tasks while inactive. When your app is moved back to the active state, its applicationDidBecomeActive: Thus, upon reactivation, your app should restart timers, resume dispatch queues, and throttle up OpenGL ES frame rates again. However, games should not resume automatically; they should remain paused until the user chooses to resume them. While the screen is locked, any attempts to access the corresponding files will fail. So if you have such files, you should close any references to them in your applicationWillResignActive: Always save user data at appropriate checkpoints in your app. Although you can use app state transitions to force objects to write unsaved changes to disk, never wait for an app state transition to save data. For example, a view controller that manages user data should save its data when it is dismissed. Responding to Temporary Interruptions When an alert-based interruption occurs, such as an incoming phone call, the app moves temporarily to the inactive state so that the system can prompt the user about how to proceed. The app remains in this state until the user dismisses the alert. At this point, the app either returns to the active state or moves to the background state. Figure shows the flow of events through your app when an alert-based interruption occurs. Instead, the banner is laid along the top edge of your app window and your app continues receive touch events as before. However, if the user pulls down the banner to reveal the notification center, your app moves to the inactive state just as if an alert-based interruption had occurred. Your app remains in the inactive state until the user dismisses the notification center or launches another app. At this point, your app moves to the appropriate active or background state. The user can use the Settings app to configure which notifications display a banner and which display an alert. A locked screen has additional consequences for apps that use data protection to encrypt files. The steps that occur when moving to the foreground are shown in Figure Objects in your app can use the default notification center to register for this notification. Be Prepared to Process Queued Notifications An app in the suspended state must be ready to handle any queued notifications when it returns to a foreground or background execution state. To make sure these changes are not lost, the system queues many relevant notifications and delivers them to the app as soon as it starts executing code again either in the foreground or background. To prevent your app from becoming overloaded with notifications when it resumes, the system coalesces events and delivers a single notification of each relevant type that reflects the net change since your app was suspended. Table lists the notifications that can be coalesced and delivered to your app. Most of these notifications are delivered directly to the registered observers. Some, like those related to device orientation changes, are typically intercepted by a system framework and delivered to your app in another way.

Chapter 6 : Application Lifecycle Demo for theinnatdunvilla.com - Xamarin | Microsoft Docs

As a matter of course, an app written in the iOS system goes through a set of states as it runs. These states are known as states of the app's lifecycle. As an app moves through the states of its lifecycle, the state of the app is defined by its level of activity such as Not Running, Active or.

Keep in mind that in this lifecycle step the view bounds are not final. Good place to init and setup objects used in the viewController. When this method gets called, the view of the view controller has been created and you are sure that all the outlets are in place. It is also a good place where to start some background activity where you need to have the user interface in place at the end. A common case are network calls that you need to do only once when the screen is loaded. This method is called only once in the lifetime of a view controller, so you use it for things that need to happen only once. BOOL animated You override this method for tasks that require you to repeat every time a view controller comes on screen. Keep in mind that this method can be called several times for the same instance of a view controller. This event is called every time the view appears and so, there is no need to add code here, which should be executed just one time. Usually you use this method to update the user interface with data that might have changed while the view controller was not on the screen. You can also prepare the interface for animations you want to trigger when the view controller appears. BOOL animated This method gets called after the view controller appears on screen. You can use it to start animations in the user interface, to start playing a video or a sound, or to start collecting data from the network. In some cases can be a good place to load data from core data and present it in the view or to start requesting data from a server. When the memory starts to fill up, iOS does not use its limited hard disk space to move data out of the memory like a computer does. If your app starts using too much memory, iOS will notify it. Since view controllers perform resource management, these notifications are delivered to them through this method. In this way you can take actions to free some memory. Keep in mind that if you ignore memory warnings and the memory used by your app goes over a certain threshold, iOS will end your app means this will look like a crash to the user and should be avoided. You rarely need to override this method since there are few common tasks that need to be performed at this point, but you might need it. You usually override this method to stop tasks that are should not run while a view controller is not on screen. For example, you can stop listening to notifications, observing other objects properties, monitoring the device sensors or a network call that is not needed anymore. He has been involved in defining the overall strategy of the company. He has been driving Zaptech Solutions since the last 8 years to deliver outstanding and exceptional solutions to the clients globally.

Chapter 7 : The App Life Cycle

iOS does send the applicationWillTerminale message, but only in the cases explained in the diagram, namely if you were compiling with SDK<4 (which can't be the case nowadays) or if you set UIApplicationExitsOnSuspend to YES in your app's theinnatdunvilla.com

Next Previous The App Life Cycle Apps are a sophisticated interplay between your custom code and the system frameworks. The system frameworks provide the basic infrastructure that all apps need to run, and you provide the code required to customize that infrastructure and give the app the look and feel you want. To do that effectively, it helps to understand a little bit about the iOS infrastructure and how it works. Understanding those design patterns is crucial to the successful creation of an app. It also helps to be familiar with the Objective-C language and its features. What is different is that for iOS apps you do not write the main function yourself. Instead, Xcode creates this function as part of your basic project. Listing shows an example of this function. With few exceptions, you should never change the implementation of the main function that Xcode provides. The only pieces that you have to provide are the storyboard files and the custom initialization code. At the heart of every iOS app is the UIApplication object, whose job is to facilitate the interactions between the system and other objects in the app. Figure shows the objects commonly found in most apps, while Table lists the roles each of those objects plays. The first thing to notice is that iOS apps use a model-view-controller architecture. This architecture is crucial to creating apps that can run on different devices with different screen sizes. It also reports key app transitions and some special events such as incoming push notifications to its delegate, which is a custom object you define. Use the UIApplication object as isâ€”that is, without subclassing. App delegate object The app delegate is the heart of your custom code. This object works in tandem with the UIApplication object to handle app initialization, state transitions, and many high-level app events. For example, a banking app might store a database containing financial transactions, whereas a painting app might store an image object or even the sequence of drawing commands that led to the creation of that image. In the latter case, an image object is still a data object because it is just a container for the image data. Apps can also use document objects custom subclasses of UIDocument to manage some or all of their data model objects. Document objects are not required but offer a convenient way to group data that belongs in a single file or file package. A view controller manages a single view and its collection of subviews. The UIViewController class is the base class for all view controller objects. It provides default functionality for loading views, presenting them, rotating them in response to device rotations, and several other standard system behaviors. UIKit and other frameworks define additional view controller classes to implement standard system interfaces such as the image picker, tab bar interface, and navigation interface. Most apps have only one window, which presents content on the main screen, but apps may have an additional window for content displayed on an external display. To change the content of your app, you use a view controller to change the views displayed in the corresponding window. You never replace the window itself. In addition to hosting views, windows work with the UIApplication object to deliver events to your views and view controllers. A view is an object that draws content in a designated rectangular area and responds to events within that area. Controls are a specialized type of view responsible for implementing familiar interface objects such as buttons, text fields, and toggle switches. The UIKit framework provides standard views for presenting many different types of content. You can also define your own custom views by subclassing UIView or its descendants directly. In addition to incorporating views and controls, apps can also incorporate Core Animation layers into their view and control hierarchies. Layer objects are actually data objects that represent visual content. Views use layer objects intensively behind the scenes to render their content. You can also add custom layer objects to your interface to implement complex animations and other types of sophisticated visual effects. What distinguishes one iOS app from another is the data it manages and the corresponding business logic and how it presents that data to the user. Most interactions with UIKit objects do not define your app but help you to refine its behavior. For example, the methods of your app delegate let you know when the app is changing states so that your custom code can respond appropriately. The UIApplication object sets up the main run loop at launch time and uses it to process events and handle updates to view-based interfaces. This behavior ensures that user-related events are processed serially in the order in which they were received. Figure shows the architecture of the main run loop and how user events result in actions taken by your app. As the user interacts with a device, events related to those interactions are generated by the system and delivered to the app via a special port set up by UIKit. Events are queued internally by the app and dispatched one-by-one to the main run loop for execution. The UIApplication object is the first object to receive the event and make the decision about what needs to be done. A touch event is usually dispatched to the main window object, which in turn dispatches it to the view in which the touch occurred. Other events might take slightly different paths through various app objects. The most common ones are listed in Table Many of these event types are delivered using the main run loop of your app, but some are not. Some events are sent to a delegate object or are passed to a block that you provide. For information about how to handle most types of eventsâ€”including touch, remote control, motion, accelerometer, and gyroscopic eventsâ€—see Event Handling Guide for iOS.

Chapter 8 : ios - What is the Life Cycle of an iPhone application? - Stack Overflow

Understanding the architecture of iOS applications is key to building better apps. There is an organized flow behind the scenes: the application life cycle, which drives the sequence of events and.

Thousands of apps are uploaded on both the Android and IOS app stores regularly but to create an app that renders successful results requires extensive research, planning and collaboration between the developer and the client. The advantage of having a mobile app is not only to extend business propositions but it also offers several benefits such as brand marketing, generating sales and improves user retention rate across the globe. With the advent of mobile app technology, most of our tangible commitments and obligations have become simpler. Connecting to other people, getting entertainment and even running businesses with the help of a single software is being revolutionized by mobile apps alone. Getting started With iOS App Development Having a mobile app is not elusive for most businesses or individuals but the proper execution of its development matters the most. The one thing similar in successful apps is thorough planning because with millions of mobile apps in the app stores not every app turns out to be successful, as some are never heard of again. There exist two major mobile operating systems and IOS app platform is one of them. The development process of both Android and IOS platform is more or less similar but there are certain rigid rules and guidelines that matter in the development cycle of IOS apps. Thorough Research The idea of creating an app without proper planning and research is a failure itself. The requirement identification phase comprises analyzing your target audience, the need to create the app and its core features. Analyzing the requirements is only done with a well-ordered teamwork or else many hindrances in managing the development of an app can occur. Once you have divided the tasks amongst the team, the next step is to perform a competitor analysis as it makes you evaluate the kind of marketing and development tactics they would be using. Technical Evaluation The need for a technical assessment is important in understanding the integration of back-end systems with the design interface of your app. Mobile apps built on IOS have different functionalities as compared to Android. Therefore, the need to understand these little intricacies before proceeding to create the app is important. A technical evaluation also comprises of finding the right architecture for your app system. The components of your chosen architecture communicate with various design elements that is why you need to be careful in selecting an architecture as it becomes difficult to revert any further shortcomings. Building Prototypes Pre-planning alone is not adequate to build an entire app. Most developers tend to skip this step but it becomes a hindrance for them in the rest of the development stages. You cannot formulate the entire look-and-feel of your app with a visualization alone. With tools such as Mockingbird, prototyping and wireframing is easier to perform and developers can easily collaborate with the client, in turn, saving more time. User Interface Designing Use interface is as important as the features and functions needed for the front-end and back-end development of an app. App developers fail to realize that no matter how good the functionalities of your app are, users will not stick if your app does not have an interesting and engaging user interface. To determine the complete visual shape of your mobile app it is necessary to analyze aspects that can make the user interface attractive and user-friendly. App Development Developing a prototype of your mobile app lends assistance in both the front-end and back-end development of the app. The most complex step is to turn an idea into a fully functional app and for that reason, the front-end development comprises of UI design elements and emphasizes on how the user will interact with the core interface. As for the back-end development, all the factors of server side functions come into play. From the structure to authentication protocols, the main objective is to focus on the server-side and give substance to the overall usability of the mobile app. Testing and Deployment Before deploying and launching your app on an app store, it is necessary to perform adequate testing in order to ensure that your app maintains good quality standards. The testing phase mainly comprises of UAT, beta, security testing, and reviewing codes. If your app clears these phases then your app is ready for its launch. Once the testing is complete, the final step is to launch your app. The developer will scan it for the last time and publish the app on the app store. After approval from the app store administrators, your app will be available for the users to view. The lifecycle of your mobile app does not end after its launch. From that point onwards, many factors of iPhone app development related to app store optimization, monetization and app updates come into play. For a mobile app to yield you great results in terms of sales generation, recognition and even monetary benefits, constant management and acquiring app insights are essential for future development purposes.

Chapter 9 : News, Tips, and Advice for Technology Professionals - TechRepublic

Application lifecycle demo for theinnatdunvilla.com 07/17/; 2 minutes to read Contributors. In this article. This article and sample code demonstrates the four application states in iOS, and the role of the AppDelegate methods in notifying the application of when states get changed.

In this article iOS regulates background processing very tightly, and offers three approaches to implement it: Register a Background Task - If an application needs to complete an important task, it can ask iOS not to interrupt the task when the application moves into the background. For example, an application might need to finish logging in a user, or finish downloading a large file. Register as a Background-Necessary Application - An app can register as a specific type of application that has known, specific backgrounding requirements, such as Audio , VoIP , External Accessory , Newsstand , and Location. These applications are allowed continuous background processing privileges as long as they are performing tasks that are within the parameters of the registered application type. As of iOS 7, applications can also register to update content in the background using Background Fetch or Remote Notifications. Application States and Application Delegate Methods Before we dive into the code for background processing in iOS, we need to understand how backgrounding affects the lifecycle of an iOS application. The iOS application lifecycle is a collection of application states and methods for moving between them. An application transitions between states based on the behavior of the user and the backgrounding requirements of the application. The movement is illustrated by the following diagram: Not Running - The application has not yet been launched on the device. Inactive - The application is interrupted by an incoming phone call, text, or other interruption. Backgrounded - The application moves into the background and continues executing background code. Suspended - If the application does not have any code to run in the background, or if all code has completed, the app will be Suspended by the OS. This happens in low-memory situations, or if the user manually terminates the application. Since the introduction of multitasking support, iOS rarely terminates idle applications, and instead keeps their processes Suspended in memory. It also means applications can move freely from the Suspended state back into the Backgrounded state without drawing on system resources. OnActivated - This is called the first time the application is launched, and every time the app comes back into the foreground. This is the place to put code that needs to run every time the app is opened. OnResignActivation - If the user receives an interruption such as a text or phone call, this method gets called and the app is temporarily inactivated. Should the user accept the phone call, the app will be sent to the background. DidEnterBackground - Called when the app enters the backgrounded state, this method gives an application about five seconds to prepare for possible termination. Use this time to save user data and tasks, and remove sensitive information from the screen. WillEnterForeground - When a user returns to a backgrounded or suspended application, and launches it into the foreground, WillEnterForeground gets called. This is the time to prepare the app to take the foreground by rehydrating any state saved during DidEnterBackground. OnActivated will be called immediately after this method completes. WillTerminate - The application is shut down, and its process is destroyed. This method only gets called if multitasking is not available on the device or the OS version, if memory is low, or if the user manually terminates a backgrounded application. Note that suspended applications that get terminated will not call WillTerminate. The following diagram illustrates how the application states and lifecycle methods fit together: It is launched by double-tapping the Home button, and shows the applications whose processes are alive: Using the App Switcher, users can scroll through snapshots of all backgrounded and suspended applications. Tapping an application launches it into the foreground. Swiping up removes the application from the background, terminating its process. Important The App Switcher does not show a difference between backgrounded and suspended applications. Background App Refresh Settings iOS 7 increases user control over the Application Lifecycle by allowing users to opt out of backgrounding for applications registered for background processing. This does not prevent applications from running background tasks. If Background App Refresh is set to off, the application will be suspended immediately upon entering the background, and prevented from doing any background processing: For an example, refer to the Check Background Refresh Setting recipe.