Penthera 201: Developing with the iOS SDK

iOS SDK 4.1.1

February 11, 2021

Introduction to development of an application with the Penthera iOS SDK 4.1.1. Covers architecture, accessing the SDK, fundamentals, setup, and basic feature implementation.

Table of Contents

Welcome to the Penthera SDK

Welcome to the Penthera SDK developer guide. If you have a basic understanding of the Penthera platform, and familiarity with mobile media apps in general, this guide should enable you to integrate our Penthera SDK into a mobile app and make use of its core features. When you are done with this material we have advanced guides to help you enable expanded features which may be useful to your company and your users.

We will start with a brief discussion of typical media ecosystem components, the architecture of the Penthera SDK and how it fits into that media ecosystem, and then proceed with how to integrate our SDK into your app. If you are entirely new to Penthera or to mobile media app ecosystems, you may benefit from reading our Penthera 101: Introduction to Penthera Development document before proceeding here.

Fundamentals of the Penthera SDK

The SDK dev guide will demonstrate some basics of using the SDK, including how to run the sample SDK app we provide, how to set up the SDK in your own development project, some fundamentals of how the SDK is configured, how your code can observe the state of the engine and its downloads, as well as some examples of the most common functions your app will use.

Demo code included in this guide and in the SDK demo app demonstrates common ways to use the SDK. This is just a portion of the overall SDK functionality. After you're done looking through the demo code, read our advanced topics and best practices guides, and have a look at the API itself to see what else is available.

Typical Integrations

The Penthera Virtuoso SDK and Penthera Cloud (sometimes referred to as our "backplane") participate within your ecosystem to provide Penthera's Download2Go features and more. Your team includes the Virtuoso SDK within your mobile app, where you integrate the SDK with your application code via local API calls. The Virtuoso SDK handles necessary communications with the Penthera Cloud.

Client applications usually integrate the SDK with other services which your infrastructure provides. These frequently include:

Media Player

The Virtuoso SDK is designed to support integration with your media player of choice. When your app is ready to play an asset which is in our SDK's managed asset download queue, or to play a Penthera FastPlay-enabled streaming asset, we provide easy hooks to retrieve the appropriate reference which your app provides to your media player.

We frequently encounter teams using standard players such as AVPlayer on iOS, ExoPlayer on Android, as well as Bitmovin player and others. The SDK provides wrappers which make it easy to get started with some of the standard players, as well as instructions on how to achieve deep integration with most any player you may choose.

Media Catalog

Your application is responsible for interacting with your media catalog, where it retrieves asset information for display in your user interface. When appropriate, such as to request an asset be downloaded for offline playback, your app uses the asset information to instruct our SDK to manage the download.

Content Distribution Network (CDN)

Virtuoso SDK is designed to transparently support various CDNs. Typically your app code retrieves asset information from your media catalog, then provides asset URLs and other details (e.g., desired bitrates and languages) to our SDK. This information is provided to our SDK when your code requests our download engine to manage the download, and our SDK retrieves the asset from the CDN. For asset types which use a manifest, our SDK retrieves the manifest from the URL you provide, parses it, and downloads the relevant components from the CDN.

User Authorization

Your application is responsible for interacting with your backend to perform any necessary user authentication & authorization. Your application code will register devices to the Penthera Cloud through our SDK, and will startup our SDK with an appropriate user identifier.

In support of various privacy regulations, we have no requirement for you to provide the Penthera SDK or Cloud with any user-identifiable information. Your code may provide our SDK with user and device identifiers which are only associated with real users within your own app code or authentication system. Our only requirement is that user and device identifiers are unique to those users and devices. The user ID you use to startup our SDK may even represent a family unit instead of an individual person.

Analytics

The Penthera Cloud collects information about the usage of our SDK in your apps. This information is as anonymous as the user, device and asset IDs you provide to us. We provide some analytic information to you within our backend web interface, and can also provide the raw data to you for further analysis. We periodically look for patterns of anomalies in the backend data which can help us identify potential customer issues.

The Virtuoso SDK also allows you to send custom events from your app to our Cloud in order to augment the standard data we collect for you. Your app code can also retrieve many of our SDK events on the device, if you choose to push those direct from the device to your own analytic engine.

Digital Rights Management (DRM)

The Virtuoso SDK provides out-of-box support for various popular DRM systems, including Apple FairPlay and Google Widevine. The Virtuoso SDK automates the management of persistent/offline DRM keys so that your users' assets are available for offline playback according to rules embedded in the DRM keys and according to business rules you configure in our Cloud. Our SDK will attempt to renew DRM licenses at opportune times during connectivity to ensure that they are as up-to-date as possible

The SDK also provides several layers of APIs for coding to any custom DRM server or key management requirements. Your team has easy access in the SDK to make the most typical modifications required by third-party DRM servers, such as additional request headers, GET/POST parameters, base-64 encoding/decoding, and JSON request/response bodies. For even greater customization the SDK also supports powerful custom low-level access to the DRM interactions.

Supported Mobile OS Versions

The Penthera SDK supports the vast majority of currently shipping mobile devices. At this time the officially supported mobile OS versions are:

- iOS and iPadOS 10 and greater

Earlier versions than these are not officially supported. If you wish to support earlier OS versions you may want to try earlier versions of our SDK, and current versions of our SDK may sometimes still work with recent unsupported OS versions.

We may attempt to answer questions about unsupported versions, but we cannot prioritize resources to resolve new issues discovered on unsupported versions which are not also present on supported versions.

NOTE

Our iOS SDK has worked on iOS 8+ with DRM-free content, Widevine or DRM system other than FairPlay. FairPlay offline playback requires iOS 10+ due to iOS restrictions. There is little reason why function with these older OS versions should be lost any time soon, but we are no longer actively testing new releases on iOS 8 - 10.

Users and Devices

To Penthera a “User” is a person, household, or other entity that owns a device. When you call the start-up method on the Virtuoso SDK, your code must supply a UserID.

The SDK uses its own internal logic to assign a unique DeviceID to the device. The SDK uploads this pair (UserID, DeviceID) to the Penthera Cloud, which associates the Device with the User. The Penthera Cloud uses the user and device IDs to enforce business rules such as the “max download-enabled devices per user” setting.

The External Device ID is an optional field we maintain for your convenience. When set by your code, the external device ID will be reported to the Penthera Cloud with device registration and in device logs. You may wish to use this for an ID which matches that device in your own backend systems.

The Penthera Cloud provides a convenient mechanism to use your External Device ID in lieu of Penthera's internal DeviceID. You can look up a device's activity on the Penthera Cloud user interface using the External Device ID. You can even perform a "remote-delete" (single asset) or "remote-wipe" (all assets) from the Penthera Cloud using an External Device ID.

NOTE

Downloading becomes available on a device when the SDK is started. If you allow some users to download and others not, based upon their login credentials, it is a good idea to delay calling the SDK startup method until your enclosing app code has verified that the user is a paying customer and/or download-enabled user.

When your app detects that a customer has transitioned from being a download-entitled user to a non-download-entitled user, there are recommended two options. If this transition is permanent, call the unregister method in the SDK to remove the device from the account. This unregister will delete all previous downloads, remove the device from the user account on the Penthera Cloud, and remove the device from your Penthera billing calculations in future months. If the transition is temporary, call the shutdown method on the SDK. Once the SDK is shut down, the user will not be able to play downloaded assets, but the assets will not be deleted from the device so they can be immediately available again once the SDK is started again with their same user. The device will not impact billing if it remains in shutdown status through an entire billing cycle.

NOTE

The iOS SDK stores the DeviceID in the device keychain. This allows the DeviceID to persist across installs, and prevents it from changing under most conditions. During development, it is possible for the DeviceID to change, if you do anything on the device or within your build that would reset the application area in the device keychain. This could include a factory reset on the device, changing the app bundle ID, or other infrequent operations. Since Apple removed the iOS SDK "device ID" value and forbid using the "advertising ID," there's no way to 100% guarantee a consistent, perpetual device ID. And, Apple's "identifier for vendor" does not persist across installs. The DeviceID value that Penthera uses will persist across app upgrades / installs, but some limited user actions may reset it. This will be a very infrequent thing, and the only reason the device ID value should change in production code would be if the user actively took steps to reset their device keychain. Apple's stance is that app developers should treat that as a "new device" anyway.

Asset Identifiers

Upon creation, each asset is assigned an internal identifier for use with the SDK, local to the device, and also a UUID which is used in analytics reporting to the backplane. Each asset creation method also contains a parameter allowing your code to provide an external string identifier for the asset, which is used to associate the asset with its ID in your catalog. The external identifier is required, and it must be unique to each asset.

How Downloading Works

The Penthera SDK proceeds through the download queue from top to bottom, attempting to complete each queued asset in order. For multi-segment assets, such as HLS and DASH, the engine will download multiple segments from within the asset simultaneously, but will focus on segments from one top-level asset at a time.

If a recoverable error occurs while downloading a file, the engine will immediately retry that file up to two more times. Immediate retries of the same file in the current downloading asset are known as the "inner retries". If the file continues to fail through each inner retry, the retry count for the asset is incremented by one, and the engine moves on to the next available asset. If a fatal, non-recoverable error occurs, the retry count is immediately set to its max.

After completing a pass through the entire queue, the engine will begin another pass if any items exist which might still be downloaded. These could include any assets whose retry counts were incremented in the previous pass, any assets which had non-permanent permission denials, or any new items added to the download queue. On each pass through the queue the engine will attempt again to finish any assets whose retry count has not reached the maximum number of attempts. These attempts are known as the "outer retries". The SDK may also retry failed assets at other opportune times, such as when the user returns to the app.

NOTICE

The default number of inner and outer retries is three, which we refer to as our "Rule of Threes."

If files of an asset had experienced errors, but on subsequent attempts files on that asset are successfully downloaded, the retry count for the asset is reset to zero. This helps to ensure that transient errors are less likely to cause a permanent asset failure. With many brief transient errors, such as in poor network conditions, the retry count on an asset may rise and fall repeatedly.

TIP

A method exists in the SDK to manually reset the retry count on an asset, which will cause the engine to attempt download of that item again without needing to delete it from the download queue and re-add it. In iOS this is the `clearDownloadRetryCountOnComplete:` method on `VirtuosoAsset`, and in Android this is the `clearRetryCount(assetId)` method on the `IQueue` interface.

DOWNLOAD MAY BEGIN BEFORE PARSING COMPLETES

As of the Penthera Android SDK 3.15.14 and iOS SDK 4.0, the engine can begin to download segmented asset files before the manifest parsing is complete. This allows downloads to begin sooner, but may result in parsing and downloading notifications arriving in a different order than expected in the past.

Coding with the SDK

This section will provide details on various aspects of coding with the SDK, including adding the SDK to your project, starting up the SDK, observing state, logging, and other topics.

Accessing Penthera SDKs on Github / Archiva

You have access to the Penthera repository at GitHub (https://github.com/penthera) where you can download the latest SDK to access documentation, change notes, documented headers, example projects, and more.

Download2Go iOS SDK

The iOS package contains:

- **iOS Developer Guide PDFs**
- **Tutorials**: Example projects containing code for various important SDK functions
- **VirtuosoClientEngine**: The libraries and supporting files that you'll include in your own project
  - **Debug & Release versions of the SDK**, in bitcode and non-bitcode formats
  - **HTML docs of the SDK** which are an important source of information about the SDK, especially some lesser-used features not covered elsewhere in documentation

TIP

The HTML docs and the header (.h) files are an important source of knowledge which is frequently overlooked.

- **Change List PDF** Release notes for SDK versions
- **README.md** High-level info about Penthera and the SDK

Running the SDK examples: iOS

Within the release distribution zip and available from GitHub, we provide a variety of example projects. These illustrate how to perform various SDK functions within a working code project. The examples use public-domain videos (HLS and mp4), hosted by Penthera on Amazon AWS, but you could replace these with references to your own videos while exploring the functions.

To build and run the examples:

1. Open the project. Make sure the provisioning profiles are set correctly. The SDK currently supports iOS and iPadOS 10, so ensure that your build settings are configured for an appropriate deployment target.
2. Add the public/private key that Penthera gave you. (If we didn't, then just ask!) Look for the placeholder where you need to include these keys. If you forget this step, the app will not compile.
3. Compile and run the app from XCode.

Set up the SDK: iOS

The SDK is packaged as an XCFramework. This allows XCode to automatically access header and resource files that it needs.

There are three ways to include the SDK in your project: directly importing and configuring the framework, referencing the framework via CocoaPods, and referencing the framework via Swift Package Manager. When Carthage adds support for XCFrameworks, our current SDK will already be compatible with that as Continue here to directly import, skip to Add the Framework with CocoaPods [10] or skip to Add the Framework with Swift Package Manager [10].

IMPORTANT

Note that the `VirtuosoClientDownloadEngine.xcframework` supports bitcode, but lacks Widevine support. This is the recommended version for most customers.

The `VirtuosoClientDownloadEngineWidevine.xcframework` version supports Widevine but not bitcode, and *should only be used if you require Widevine*. At this time Google does not produce DASH/Widevine support libraries which will run in the iOS Simulator, so if you opt for Widevine support you must test on a real device and ship without bitcode.

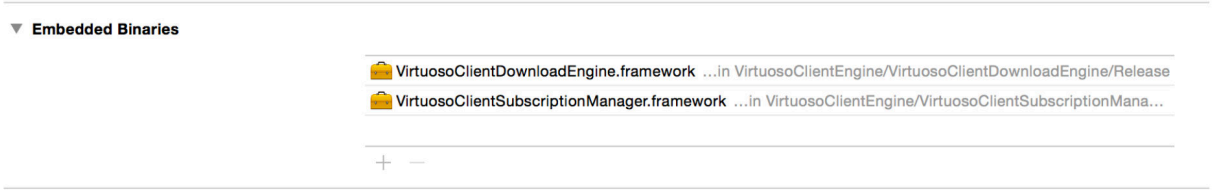
Each version of the framework comes in both release and debug forms. You should use the release version for your final build, but also for most other purposes.

Add the framework directly to your project

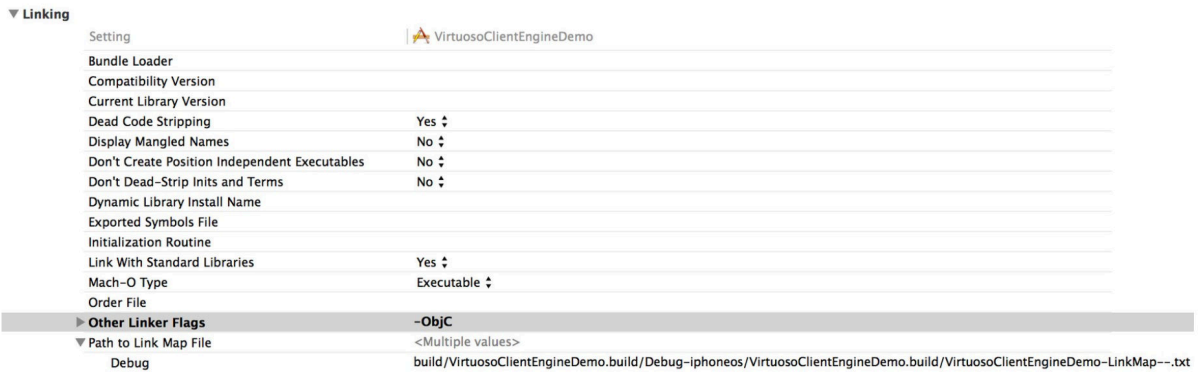
To include the SDK in your project, navigate to the SDK VirtuosoClientEngine directory and drag the VirtuosoClientDownloadEngine XCFramework into your XCode project.

Change your build settings

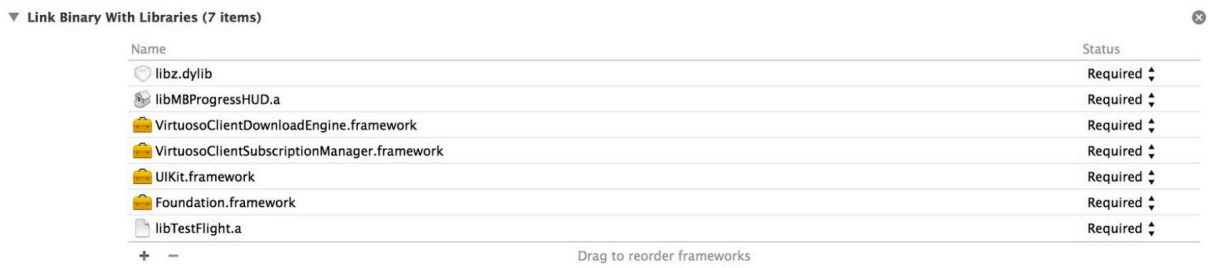
Under the General section in “Embedded Binaries”, add the VirtuosoClientDownloadEngine.framework:



Under the Linking section in “Other Linker Flags,” , add the –ObjC flag:



In the “Link Binary With Libraries” build phase, ensure that the Virtuoso framework is included, and add the stock library, libz.dylib:



Add the Framework with CocoaPods

Instead of adding the Virtuoso framework to your project directly, you may choose to setup your project using the popular dependency management system, CocoaPods. To use CocoaPods follow these steps:

1. Add Penthera's pod spec repository to your CocoaPods installation by executing the command:
   `pod repo add Download2Go-specs https://github.com/penthera/Download2Go-specs.git`
2. Reference Penthera's repository as a source in your Podfile:
   ```
   source 'https://github.com/penthera/Download2Go-specs.git'
   source 'https://github.com/CocoaPods/Specs.git
   ```
3. Include the Penthera frameworks in your Podfile.
   `pod 'VirtuosoClientDownloadEngine', '~> 4.1.1'`
4. Execute 'pod install' and you are ready to start coding.

NOTICE

After Penthera's iOS SDK version 3.14.x we removed a second pod which contained a subscriptions feature. If updating from 3.14.x, remove any podfile reference you may have to `VirtuosoClientSubscriptionManager`.

Add the Framework with Swift Package Manager

Instead of manually adding the SDK framework, or adding via Cocoapods, you may add the Penthera SDK to your project with Swift Package Manager. In your project, navigate to the Swift Packages tab and add our GitHub repository URL:

```
https://github.com/penthera/Download2Go-ios
```

Then complete the Xcode wizard for adding the Penthera SDK as a Swift Package.

Modify your app info.plist

In your app's info.plist:

1. Add "Application Uses Wifi" and set it to YES.

NOTE

This tells iOS to keep WiFi connections open in the background, rather than automatically timing out and transferring the connection to cellular. This improves download performance and minimizes cellular data usage.

2. In “Required Background Modes”, add the option “App downloads content in response to push notifications”. Alternatively, do this by going to the “Capabilities” application tab, in the “Background Modes” section, and enabling the “Remote Notifications” checkbox.

NOTE
The Penthera Cloud Server sends APN (push messages) to the SDK to trigger certain actions, e.g. download and delete. To send a push message to a device, the SDK registers a push notification token from your app to the Penthera Cloud. The SDK performs this function for you, so you do not need to add code to register the push token. See Push Notifications: iOS for a discussion of push notification support.

3. In “Required Background Modes”, add the option “App downloads content from the network”. Alternatively, do this by going to the “Capabilities” application tab, in the “Background Modes” section, and enabling the “Background fetch” checkbox.

NOTE
This is strongly recommended and should be performed unless your app prevents background downloads.

APPLICATION TRANSPORT SECURITY
All communications between the Penthera SDK and Penthera Cloud communications are SSL enabled. The local proxy used by the SDK for local-only serving of downloaded video to your player does not use SSL. In iOS 10+, Apple allows connections to localhost without requiring SSL, and the Penthera SDK will fully function without any security exceptions.

If you are supporting iOS 9, an additional step is required. In iOS 9 Apple first added Application Transport Security (ATS) exceptions, to enforce the use of SSL. This initial version of ATS caused *all* non-SSL connections to be rejected in apps built with the iOS 9 SDK or greater. The initial version of ATS would reject even on non-SSL connections to localhost. After iOS 9, Apple relaxed the ATS constraint on local-only connections. In iOS 9 (only) the ATS constraints cause problems with the Penthera SDK use of non-SSL connections to localhost. Therefore, if you are supporting iOS 9, for video playback to function, you need to add an ATS exception for “localhost” into your info.plist NSAppTransportSecurity key, as follows:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>localhost</key>
        <dict>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <true/>
        </dict>
    </dict>
</dict>
```

If you are only supporting iOS 10+, you do not need this configuration key.

Include Header When Necessary

You should now be able to compile your project successfully with no build errors.

Anywhere you want to access SDK classes or methods, you must import the main header:

```
#import <VirtuosoClientDownloadEngine/VirtuosoClientDownloadEngine.h>
```

We'll assume this line is included when we present you with code examples. If you have enabled pre-compiled headers for your project, we suggest you include this import in your app's .pch prefix file. This will automatically import the .h file in all your code by default.

Engine Startup: iOS

Initializing the Singleton

`VirtuosoDownloadEngine` is the main SDK class. It is a singleton. Calling the `instance` method will initialize the object if needed. Call the `instance` method each time you need to refer to the engine, rather than holding a reference to it. For example, to set the enabled status of the engine:

```
[VirtuosoDownloadEngine instance].enabled = TRUE;
```

`VirtuosoDownloadEngine` automatically cleans itself up upon receiving critical system events, like app termination or low memory conditions.

CAUTION

Avoid holding any long-lived reference to an instance of `VirtuosoDownloadEngine`. Because the engine may release its resources to free app memory or in response to a remote-kill from the Penthera Cloud, any long-lived reference could easily be an undefined state. To ensure consistent behavior, grab a handle with the `instance` method whenever needed.

Startup

Once you have a handle to the Engine singleton, you will call startup on it. Engine startup executes asynchronously, invoking a closure when startup completes, to avoid blocking code running on your UI MainThread. Startup will make network calls, update the local database, and if the user changes, will delete previously downloaded assets.

Engine parameters are passed at startup using an instance of the `VirtuosoEngineConfig` class. This config class defines the configuration properties that can be passed. Create an instance of this object, applying your settings, and pass it to `VirtuosoDownloadEngine` startup.

IMPORTANT

The `VirtuosoDownloadEngine.startup` method should NOT be repeatedly called, such as when your views are displayed. Invoking this method is time, network, and CPU intensive. Invoke it once you have all of the requisite configuration data, and then not again unless the app is restarting.

NOTE
Penthera will provide you the URL of your Penthera Cloud instance, along with an app-specific public/private key pair that allows your SDK instances to authenticate to your Cloud instance.

Unless you restrict download support to a subset of your users, apps normally perform the engine start-up early in application startup, such as inside the `application:didFinishLaunchingWithOptions` method of your `AppDelegate`. It is common to start the Penthera engine after your user has authenticated with your identity system.

```
- (BOOL)application:(UIApplication *)application
      didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  // Configure logger delegate messages
  [VirtuosoLogger addDelegate:self];

  // Configure logging
  [VirtuosoLogger setLogLevel:kVL_LogWarning];
  [VirtuosoLogger enableLogsToFile:NO];

  // Initialize engine and fetch a handle to the singleton instance
  VirtuosoDownloadEngine* engine = [VirtuosoDownloadEngine instance];

  // Global “on switch” for downloading
  [engine setEnabled:YES];

  VirtuosoEngineConfig *engineConfig = [[VirtuosoEngineConfig alloc]
    initWithUser:@”an_id_you_assign_for_this_user”
    backplaneUrl:@”https://penthera.provided.url”
    publicKey::@"penthera_provided_public_key"
    privateKey:@"penthera_provided_private_key"
    externalDeviceID:@”an_id_you_assign_for_this_device” ];

  [engine startup:engineConfig startupCallback:^(kVDE_EngineStartupCode
status) {
    switch(status) {
      case kVDE_EngineStartupSuccess: {
        //startup succeeded
      }
      break;

      default:
        //startup failed
        //see enum in VirtuosoConstants.h for set of failure states
    }
  }];

  ...
}
```

TIP

In addition to the callback, the SDK Engine delegate method `downloadEngineStartupComplete` on the `VirtuosoDownloadEngineNotificationsDelegate` is called when engine startup completes.

It is your responsibility to supply a unique user ID to the SDK when you call the startup method. The SDK uses this user ID in reporting, and to enforce business rules (such as “max number of download-enabled devices per user”). If you don’t know the user ID by the time `didFinishLaunchingWithOptions` executes, you’ll need to delay calls to startup until you know the user ID, such as after an opening user login dialog.

For more details on UserID, DeviceID, and related topics, see Penthera 203: Best Practices and Penthera 101: Introduction to Penthera Development.

DEVICE STARTUP & SYNCING MAY IMPACT BILLING

Startup of the Virtuoso engine from a client device, and the periodic sync the engine performs with the Penthera Cloud servers, are what identify a device as being "active" in the Penthera ecosystem. The number of active devices in a given month impacts Penthera resource utilization, and is the typical mechanism from which Penthera derives its client billing.

The typical best practice for a client application is to leave users in the active state who rely on download and other SDK features. For such users you would startup the Virtuoso engine soon after they launch or log in to your app, and you would leave the engine running so it can manage background downloads and other features automatically.

If you limit download and other SDK features to a subset of your users, such as a premium user account, you will normally only startup the Virtuoso engine for those premium users (and not for users who are unable to use Virtuoso SDK features).

State Management: iOS

To integrate the Penthera SDK with your various application user interface components, you will want to understand state reporting in the SDK, including the general SDK engine state, the state of the download queue and various queued assets, as well as states and updates related to the Penthera Cloud (the "backplane").

While the traditional method of receiving state change messages was to register `NSNotification` listeners with `NSNotificationCenter`, we have simplified the process by offering a delegate notification pattern. You need only implement a desired delegate, register that with the SDK, and the Engine will invoke your delegate methods with type-safe parameters. This eliminates the need to manually add listeners to `NSNotificationCenter` for the various engine notifications. Our notification delegates can be implemented by any view or model object with a minimum number of required delegate methods. Many of the delegate methods are optional, so check header documentation in `VirtuosoDownloadEngineNotificationsManager.h` for more information on when each method is triggered, which are required, and which are optional.

TIP

When you register one of our notification delegates, by default it will receive callbacks on the main thread. If you wish to receive callbacks off the main thread, you should provide the desired queue when you register the delegate with the SDK.

The following example shows a `ViewController` that implements our download engine delegate, `VirtuosoDownloadEngineNotificationsDelegate`, to handle various notifications. Such a delegate instance is registered to receive SDK callbacks by providing it to one of the init methods when creating an instance of `VirtuosoDownloadEngineNotificationManager`.

```
class ViewController: UIViewController,
VirtuosoDownloadEngineNotificationsDelegate {
    var downloadEngineNotifications:
VirtuosoDownloadEngineNotificationManager!

    override func viewDidLoad(){
        super.viewDidLoad()
        downloadEngineNotifications
            = VirtuosoDownloadEngineNotificationManager.init(delegate: self)
    }

    func downloadEngineDidStartDownloadingAsset(_ asset: VirtuosoAsset) {
    }

    func downloadEngineProgressUpdated(for asset: VirtuosoAsset) {
    }

    func downloadEngineProgressUpdatedProcessing(for asset: VirtuosoAsset) {
    }

    func downloadEngineDidFinishDownloadingAsset(_ asset: VirtuosoAsset) {
    }

    func downloadEngineDidEncounterError(for asset: VirtuosoAsset,
        error: Error?,
        task: URLSessionTask?,
        data: Data?,
        statusCode: NSNumber?) {
    }

    func downloadEngineInternalQueueUpdate() {
    }

    func downloadEngineStartupComplete(_ succeeded: Bool) {
    }
}
```

The SDK also still supports the *legacy* `NSNotificationCenter` approach of sending various `NSNotification` messages to let your app know about changes in status. If you choose the legacy approach, your code would register to receive whichever notifications are relevant to your app. Notifications may regard general state changes, process results, warnings, and errors. Look for the latest set of these notifications in `VirtuosoNotifications.h`. Receiving any of these notifications will give you

the appropriate status update, and additional information is often contained in the `userInfo` dictionary contained in the notification.

Engine Status: iOS

One of the most common needs for feedback from the Penthera SDK is to know the general status of the Penthera download engine. To receive state change notices your app will want to register an instance of VirtuosoDownloadEngineNotificationsDelegate which implements the following method:

```
-(void)downloadEngineStatusChange:(kVDE_DownloadEngineStatus)status
statusInfo:(VirtuosoEngineStatusInfo* _Nonnull)statusInfo;
```

Register your delegate by initializing an instance of VirtuosoDownloadEngineNotificationManager. When your delegate method is called by the SDK, your code will have access to the new status and related status info.

IMPORTANT

Note that the "blocked" status is a recoverable state, which the SDK will sort out on its own. Only the "errors," disabled and auth-failed statuses are expected to require intervention. by the user or your code.

TIP

Two other engine status events relate to the beginning and ending of a data store upgrade, which only occurs during first launch after some SDK updates. These are discussed in Upgrading the SDK: iOS.

The delegate method is preferred, but alternatively, your code may use the legacy `NSNotificationCenter` approach to observe the engine status notifications with the following name:

```
extern NSString* kDownloadEngineStatusDidChangeNotification;
```

When the legacy NSNotifications are received, retrieve the value in `userInfo` for key `kDownloadEngineStatusDidChangeNotificationStatusKey`. This will be one of the `kVDE_DownloadEngineStatus` values from `VirtuosoConstants.h`. From this you will know if the entire engine status is idle, downloading, disabled, blocked, error, auth-failed, or otherwise.

Asset Queue & Download Status: iOS

To receive asset queue and download status updates, your app will want to register an instance of `VirtuosoDownloadEngineNotificationsDelegate` which implements any of the following methods. Register your delegate to receive callbacks from the SDK by initializing an instance of `VirtuosoDownloadEngineNotificationsManager`. See `VirtuosoDownloadEngineNotificationsManager.h` for more details and for additional useful methods.

```
-(void)downloadEngineDidStartDownloadingAsset:(VirtuosoAsset*
_Nonnull)asset;

-(void)downloadEngineProgressUpdatedForAsset:(VirtuosoAsset* _Nonnull)asset;

-(void)downloadEngineProgressUpdatedProcessingForAsset:(VirtuosoAsset*
_Nonnull)asset;

-(void)downloadEngineDidFinishDownloadingAsset:(VirtuosoAsset*
_Nonnull)asset;

-(void)downloadEngineDidFinishDownloadingAncillary:(VirtuosoAncillaryFile*
_Nonnull)ancillary forAsset:(VirtuosoAsset* _Nonnull)asset;

-(void)downloadEngineInternalQueueUpdate;

-(void)downloadEngineDidEncounterWarningForAsset:(VirtuosoAsset*
_Nonnull)asset error:(NSError* _Nullable)error;

-(void)downloadEngineDidEncounterErrorForAsset:(VirtuosoAsset*
_Nonnull)asset
                                         error:(NSError* _Nullable)error
                                          task:(NSURLSessionTask*
_Nullable)task
                                          data:(NSData* _Nullable)data
                                    statusCode:(NSNumber*
_Nullable)statusCode;

-(void)downloadEngineDeletedAssetId:(NSString* _Nonnull)assetID;

-(void)downloadEngineIsEnteringBackground:(NSArray*
_Nullable)continuingAssets pausingAssets:(NSArray* _Nullable)pausingAssets;

-(void)downloadEngineStatusChange:(kVDE_DownloadEngineStatus)status
statusInfo:(VirtuosoEngineStatusInfo* _Nonnull)statusInfo;
```

Error and Warning Descriptions

The following chart summarizes error conditions and Virtuoso's behavior. For the most up-to-date list of errors, see the `kVDE_DownloadErrorCode` enumeration in `VirtuosoConstants.h`.

Condition	Description	Retry?
Invalid mime type	The MIME type advertised by the HTTP server for the file isn't included in the MIME types whitelist you provided earlier.	No (see note below)
Final file segment size disagrees with server-provided segment size	After a segment download completes, the on-disk file size didn't match the expected size, as reported by the server.	Yes
Network error	Some network issue (HTTP 404,416, etc.) caused the download to fail.	Yes
File System Error	The OS couldn't write the file to disk. In most cases, the root cause is a full disk.	Yes

NOTE
The SDK never gives up completely on an incomplete asset which remains in queue. Error and retry counts are reset when the engine is started, so downloads will be attempted again for *any* incomplete assets. This means that even assets with invalid MIME types will be retried eventually, such as after a subsequent startup of the engine. Some errors, such as the MIME error, are considered fatal within the lifetime of the current engine instance, while other errors are retried within the current engine instance lifecycle.

Legacy NSNotifications for Asset & Queue Status

The `VirtuosoDownloadEngineNotificationsDelegate` is preferred for your ease of use, but asset and download queue status updates are also still available as `NSNotifications` in two forms: notification of normal updates, and notification of errors & warnings. Each notification key is listed in `VirtuosoConstants.h` for your further investigation.

Normal Status Updates

To receive a variety of status updates for assets and the download queue during normal operations, register to observe any of the following named `NSNotifications`:

```
extern NSString* kDownloadEngineStatusDidChangeNotification;
extern NSString* kDownloadEngineDidStartDownloadingAssetNotification;
extern NSString* kDownloadEngineProgressUpdatedForAssetNotification;
extern NSString*
kDownloadEngineProgressUpdatedForAssetProcessingNotification;
extern NSString* kDownloadEngineInternalQueueUpdateNotification;
extern NSString* kDownloadEngineDidFinishDownloadingAssetNotification;
extern NSString* kDownloadEngineIsEnteringBackgroundNotification;
```

The relevant `VirtuosoAsset`, if any, is contained in the `userInfo` dictionary of the notification, under the key `kDownloadEngineNotificationAssetKey`. In some circumstances other info in contained in the `userInfo` dictionary, and statuses of the asset itself may also be useful when processing these notifications.

Errors & Warnings

The Penthera SDK communicates download issues for a given asset to the enclosing app at two levels of severity: errors and warnings. Both are sent as named `NSNotifications`:

```
extern NSString* kDownloadEngineDidEncounterErrorNotification;
extern NSString* kDownloadEngineDidEncounterWarningNotification;
```

1. When the SDK encounters an issue that will eventually cause the file to be marked as blocked, it issues a `kDownloadEngineDidEncounterErrorNotification`. It will send this notice even if the file download will be retried, but with repeated errors with the same download the SDK will likely stop retries on this asset and mark it as being in an error state. The error state is unrecoverable without intervention from your code or a user action (e.g., removing the download from queue, or resetting the download queue if you provide the user with that option).
2. When the SDK determines a potential issue exists (such as when the server reported size and expected size do not match), it issues a `kDownloadEngineDidEncounterWarningNotification`. This notice indicates that something unexpected happened, but the file download will still finish and be marked as successfully completed. Warnings are usually either recoverable or reasonably ignored.

In the case of both notices, the userInfo dictionary sent with the notice will contain an `NSError` object in the `kDownloadEngineNotificationErrorKey` that contains detailed information about the error

that was encountered. The `VirtuosoAsset` which encountered the error will be contained in the `kDownloadEngineNotificationAssetKey`.

Penthera Cloud Status: iOS

As with engine, asset and queue status notifications, some status updates relative to the Penthera Cloud can be received using a delegate or as named `NSNotifications`. Most of these require no action from your application, and are typically handled by the SDK. Some of the events may warrant taking some action in your app, such as disabling features, logging the user out, or providing visual feedback to the user.

IMPORTANT

Not all messages are available via the delegate, so review the methods of `VirtuosoBackplaneNotificationsDelegate` and compare them to the kBackplane keys in VirtuosoConstants to determine what may be of interest for your app.

To receive delegate callbacks for some of the Penthera Cloud (backplane) events, implement methods of the `VirtuosoBackplaneNotificationsDelegate` and use it to instantiate an instance of `VirtuosoBackplaneNotificationsManager`. The delegate methods include:

```
-(void)backplaneSyncCompleteWithStatus:(Boolean)status error:(NSError*
_Nullable)error;

-(void)backplaneRemoteKill;

-(void)backplaneStartingRemoteKill;

-(void)backplaneDidUnregisterDeviceWithStatus:(Boolean)success error:
(NSError* _Nullable)error;
```

Even more cloud/backplane events are available as NSNotifications. The complete set of cloud status NSNotifications are listed in `VirtuosoConstants.h`, with a selection as follows:

```
extern NSString* kBackplaneDidUnregisterDeviceNotification;
extern NSString* kBackplaneDeviceLimitReachedNotification;
extern NSString* kBackplaneInvalidCredentialsNotification;
extern NSString* kBackplaneDeviceAlreadyRegisteredNotification;
extern NSString* kBackplaneCommunicationsFailureNotification;
extern NSString* kBackplaneSyncResultNotification;
extern NSString* kBackplaneDeviceSaveResultNotification;
extern NSString* kBackplaneLogsSentNotification;
extern NSString* kBackplaneRemoteKillNotification;
```

Error Domains: iOS

Error domains for NSErrors returned by the SDK help customers categorize Error.code values. Penthera now creates errors using error domains that are grouped and matched directly to the source of the error as defined by an Enum type in the SDK. The following table shows the Error Domain, and the Error Codes that will be associated with errors in that domain.

Error Domain	Related Error Codes
Virtuoso.DownloadErrorCode	kVDE_DownloadErrorCode (enum)
Virtuoso.HttpResponseCode	HttpResponse status codes, i.e. 200-500

Error Domain	Related Error Codes
Virtuoso.BackplaneStatusCode	kVBP_StatusCode (enum)
Virtuoso.DownloadErrorType	kVDE_DownloadErrorType (enum)
Virtuoso.ErrorCode	kVDE_ErrorCode (enum)
Virtuoso.PlayerError	kVDE_PlayerErrorCode (enum)
Virtuoso.AssetResourceLoader	kVAV_ErrorCode (enum)

Background vs Foreground Downloading: iOS

When your app is in the foreground and the Virtuoso engine is active, the SDK will manage and optimize downloads within the foreground process. The SDK makes use of the iOS Background Transfer Service (BTS) to download while the enclosing app is not actively running. The SDK automatically handles the handoff of download management between the foreground and background processes to optimize the overall experience.

Apple strictly constrains what an app can do when it is not in the foreground. The SDK provides as much functionality as possible given these constraints.

WARNING

Apple's Background Download Limits

Apple's BTS policies governing background download are intentionally opaque. Empirically, Penthera has observed that BTS will download up to around 4GB per 24-hour period per app, then will suspend BTS downloading for that application until the next day. This is unpublished behavior by Apple and may change with future iOS versions.

Background Download Session Notices

The Penthera SDK injects code into your AppDelegate chain to ensure the SDK receives notices from background download sessions. Prior to version 4.0 of our SDK you were required to add some code to your `UIApplicationDelegate`, but this is no longer necessary.

Delivery of background events to `UIApplicationDelegate` for your own features, if you need them, should not be impacted by the SDK and should work as documented by Apple.

Configure Logging: iOS

There are two separate logging paths in the SDK:

Event Logging

The SDK can capture many different events of potential business value, e.g. when a download starts and stops, offline playout, etc.

You can configure which of these events the SDK uploads to the Backplane. By default, most events are enabled; you should choose which events you want to enable or disable. Available events are listed in the kVL_LogEvent enumeration.

Use the setLoggingEnabled:forEvent: or setLoggingEnabledForAllEvents: methods to enable event logging. We suggest you do this prior to the logger startup call.

NOTICE

You cannot disable the “download queued”, “download start”, “download complete”, “download error”, “max errors reset”, or “reset” events. Attempts to do so will have no effect.

IMPORTANT

Due to required differences in implementation, the Penthera iOS and Android SDKs differ with regard to download-start events. In the iOS SDK the download-start event will only occur the first time the download starts for a given queued asset. In the Android SDK, the download-start events will recur if the asset download resumes after it is stopped. See the Android documentation for more details, if needed.

Debug Logging

By default the SDK logging system is very quiet, but when configured otherwise the SDK is capable of generating lots of developer-friendly debug information. You can send log output to various locations (console, log file) or you can implement a logger delegate and handle it manually.

To modify the logging configuration, use any of the following after the logger startup call:

- `[VirtuosoLogger addDelegate:id<VirtuosoLoggerDelegate>];`

  Adds an *optional* delegate to receive SDK log events. Any such delegate must follow the `VirtuosoLoggerDelegate` protocol. Methods in the protocol can be used to receive SDK events if you wish to handle them in your own custom logging mechanisms.
- `[VirtuosoLogger setLogLevel:kVL_LogError];`

  Configures the verbosity of logging output. In this example we are setting the log level to "Errors only." The default setting is "warnings and errors." A setting of either errors-only or warnings-and-errors-only is *recommended for production*. During development you may wish to use a more verbose level. See the SDK headers to learn all the available log levels.

  TIP

  See the example code included in the SDK distribution, which demonstrates best practice code to automatically enable verbose logging during development while automatically restricting to warnings-and-errors level for production.
- `VirtuosoLogger.backplaneLoggingEnabled = true;`

  Enables logging of events arriving from the Penthera Cloud. These are disabled by default, and *you may want to leave them disabled for production*.
- `VirtuosoLogger.proxyLoggingEnabled = true;`
- Enables logging of events generated by the SDK's local playback proxy. These are disabled by default, and *you may want to leave them disabled for production*.
- `[VirtuosoLogger enableLogsToFile:NO];`

  Controls whether all log output is also written to a file in the app’s documents directory. A setting of "NO" is recommended for production.

Implementing Basic Features

Once you have learned the fundamentals of the Penthera SDK, and installed & configured the SDK in an app, this section will step you through the code necessary to implement the most common features in your app.

Enable/Disable for Downloads: iOS

The SDK provides two switches that control download. They have different purposes. The SDK only downloads if both toggles are 'on'.

SDK "Master Switch"

A Boolean property on the VirtuosoSDK instance in the SDK that you can set to toggle downloading. Use this to disable downloads, for example, when your app is streaming video and you don't want to share bandwidth between streaming and download. Be careful: this value persists across app restarts.

```
VirtuosoDownloadEngine.instance.enabled = true
```

Backplane-Enforced Toggle

The Backplane enforces a limit on the number of Devices that each User may have enabled for download. To do so, the Backplane maintains, for each known Device, a flagindicating whether the Device is permitted to download. If new devices are created in a user account, and the limit hasn't yet been reached, then the new devices are automatically enabled on the Backplane.If the limit has been reached, then newly-added Devices are disabled from downloading via the read-only device property:

```
VirtuosoDevice *myDevice = [VirtuosoDevice currentDevice];
Boolean deviceEnabledForDownload = [myDevice enabled];
```

You can request a change to this flag by calling the following method. The device needs to be online and able to connect to the Backplane for this to succeed.

```
[myDevice
    updateDownloadEnabled:YES
    onComplete:^(Boolean success, NSError *error)
    {
        //update your UI to reflect state
    }
];
```

NOTICE
The Backplane also offers a web API to enable/disable devices. See the server documentation.

Queue an Asset for Download: iOS

In the SDK, every downloadable video instance is a `VirtuosoAsset`. A `VirtuosoAssetConfig` instance is used to set asset parameters, and is then passed to the initializer to create a `VirtuosoAsset` instance. In addition to the init params of the `VirtuosoAssetConfig`, the `VirtuosoAssetConfig` contains various other parameters with reasonable defaults. You may modify additional config pa-

rameters before instantiating your `VirtuosoAsset` instance. Unless you change the defaults, your `VirtuosoAsset` will automatically be added to the download queue.

CAUTION

The Penthera SDK APIs to create and download assets invoke code paths that make network calls and write to CoreData to store asset configuration details. Best practice for using the Penthera SDK includes invoking many of the methods using background dispatch queues. This is necessary to ensure UI refreshes are smooth and uninterrupted by long running operations like network calls and disk writes.

NOTE

In general, assets will be downloaded in the order that they were added to the queue. This is not guaranteed, as various events and rules may result in changes to the download order.

Queue an HLS Video

When an HLS video asset is created, the SDK will download and parse the related `m3u8` manifest. Based on parsing result the SDK will download all the required HLS fragments and various other elements listed in the manifest. To queue an HLS asset, create an instance of `VirtuosoAssetConfig` with appropriate parameters, and use that config to instantiate a `VirtuosoAsset` instance.

```
VirtuosoAssetConfig *config = [[VirtuosoAssetConfig alloc]
    initWithURL:@"http://path/to/main/manifest.m3u8"
    assetID:@"your_unique_asset_id"
    description:@"Test HLS Video"  //not used by the SDK
    type:kVDE_AssetTypeHLS];

//bitrate defaults to maximum unless set; set to 0 to select lowest bitrate
//[config setMaximumBitrate:...];

//set a DRM type, if needed, and see our documentation for enabling DRM
//[config setProtectionType:kVDE_AssetProtectionTypeFairPlay];

[VirtuosoAsset assetWithConfig:config];
```

NOTICE

`maximumBitrate` specifies which HLS profile the SDK should select for download, from among the HLS profiles available in the manifest. The SDK will download the the highest bitrate not exceeding `maximumBitrate`. If no profile exists lower than the provided maximum, the engine will select the lowest bitrate profile. Set `maximumBitrate=0` to force the SDK to use the lowest profile, or leave it at INT_MAX to use the highest profile.

NOTICE

You should leave `protectionType` at its default of `kVDE_AssetProtectionType-Passthrough` unless you are using DRM. Also use the default if you are integrating a DRM type that is not built into the Penthera SDK. If you are using DRM protection, you should most likely set the config parameter `includeEncryptionKeys` to NO. Take a look at the definition of `kVDE_AssetProtectionType` in `VirtuosoConstants.h` for a list of built-in DRM systems, and see Digital Rights Management (DRM): iOS for further details on implementing DRM with the Penthera SDK.

Queue a Single (Flat) File (e.g., mp4)

To queue a single file asset, create an instance of VirtuosoAsset with an appropriately configured VirtuosoAssetConfig instance:

```
VirtuosoAssetConfig *config = [[VirtuosoAssetConfig alloc]
    initWithURL:@"http://path/to/file.mp4"
    assetID:@"your_unique_asset_id"
    description:@"Test File"  //not used by the SDK
    type:kVDE_AssetTypeNonSegmented];

[VirtuosoAsset
    assetWithConfig:config];
```

Refer to the SDK header files for a full description of the behavior and syntax of every parameter.

Queue an HSS Video

HSS works similarly to HLS, but unlike HLS, the HSS format uses a separate audio and video data stream. So instead of specifying `maximumBitrate`, you need to specify both `maximumBitrate` and `maximumAudioBitrate`. As above, the SDK will select and download the profile with the highest bit rate not exceeding the values you specify.

```
VirtuosoAssetConfig *config = [[VirtuosoAssetConfig alloc]
    initWithURL:@"http://path/to/main/video.ism/Manifest"
    assetID:@"your_unique_asset_id"
    description:@"Test HSS Video"  //not used by the SDK
    type:kVDE_AssetTypeHSS];

//bitrates default to maximum unless set; set to 0 to select lowest bitrate
//[config setMaximumBitrate:...];
//[config setMaximumAudioBitrate:...];

[VirtuosoAsset assetWithConfig:config];
```

NOTE

Enabling Smooth Stream with a custom media player: The SDK does not automatically "repackage" downloaded/downloading assets. Penthera's local proxy service provides the player for the enclosing app the exact same representation of the asset as was downloaded. Thus, if you point your custom player at the SDK's playback URL, everything should work as-is.

By default, most Smooth Stream encoders create a very large number of extremely small video segments. This can lead to excessive network overhead during download and cause potential performance issues. If your Smooth Stream videos have been encoded with larger timescales, thus resulting in fewer segments with larger segment sizes, that can reduce the instance of these potential issues. Generally speaking, better download performance can be achieved by using HLS or DASH.

Pause/Resume Download: iOS

Asset downloads can be paused at the engine level (all downloads) or at the asset level.

Pause All Downloads

Use the ‘enabled’ flag on the download engine to start and pause all downloads.

Pause:

```
[[VirtuosoDownloadEngine instance]setEnabled:NO];
```

Resume:

```
[[VirtuosoDownloadEngine instance]setEnabled:YES];
```

Pause An Asset Download

Use the paused property on the individual asset to pause/resume the single download.

```
myAsset.isPaused = true
```

Resume:

```
myAsset.isPaused = false
```

Note that when an individual asset pause is removed, the asset may still wait for other downloads to complete before it is able to resume download activity.

Cancel Downloads: iOS

To cancel one asset download:

```
[[VirtuosoDownloadEngine instance] removeFromQueue:asset];
```

To cancel all downloads:

```
[[VirtuosoDownloadEngine instance] flushQueue];
```

List Assets: iOS

To access the list the assets in queue:

```
NSArray *assets = [[VirtuosoDownloadEngine instance] assetsInQueue];
```

Delete an Asset: iOS

To delete a single asset from the user device, call the following on the asset itself:

```
[myAsset deleteAssetOnComplete:^{
    //completion block for the async deletion
}];
```

If the same method is called with a nil completion block, it will execute synchronously and return when finished. If a completion block is provided, the deletion occurs asynchronously and the method call returns immediately.

Delete All Downloads: iOS

If for any reason your app needs to delete all downloaded videos:

```
[VirtuosoAsset deleteAll];
```

This method executes asynchronously, but due to the nature of the request *the SDK engine stops other functions until the deletions are completed*. If you want your code to receive a notification upon completion, use the delegate callback `downloadEngineAllAssetsDeleted` with `VirtuosoDownloadEngineNotificationManager`, or register for the NSNotification `kDownloadEngineAllAssetsDeletedNotification`.

If you shutdown the engine with one user ID, and later start it back up with a different user ID, the SDK automatically deletes all the assets of the original user at that time. If you want a user's assets to be deleted sooner, such as when you de-authenticate them, or in response to a user interface action, you can use this `deleteAll` method.

In addition, the administrator may use the Penthera Cloud web UI to schedule a remote wipe of any device. Contact support or see Penthera Cloud documentation for additional details.

Play Downloaded Content: iOS

The SDK provides a set of player-related classes to help make playback easier. Use of these classes is optional, but does provide some convenience. Using the built-in classes automatically handles built-in DRM licensing and appropriate logging of the “play start” and “play stop” log events.

If you need to use your own closed-source player, or need more control over aspects of playback than the built-in classes allow, you may also choose to use the SDK’s local HTTP proxy, VirtuosoClientHTTPServer, that sits between a media player and downloaded HLS and HSS items.

There are three built-in classes you can use to make playback easier:

- **VirtuosoAVPlayer**: A drop-in replacement for Apple’s AVPlayer class. You can useVirtuosoAVPlayer anywhere you already have an AVPlayer to enable automatic FairPlay and Widevine licensing and event logging.
- **VirtuosoPlayerView**:A UIView subclass built on top of VirtuosoAVPlayer. It provides a basic user interface that you can build from, or disable entirely and add your own user interface elements.
- **VirtuosoPlayerViewController**:A full view controller built on top of VirtuosoPlayerView and VirtuosoAVPlayer. For rapid prototypes and proof of concept applications, this classprovides the fastest mechanism to play back MP4, HLS, and DASH videos encoded with FairPlay or Widevine.

NOTE
Most Penthera customers use the Penthera SDK in conjunction with a commercial DRM system. Common DRM works with the provided VirtuosoPlayerView. See Digital Rights Management (DRM): iOS for further details of DRM support including how to use a custom player if required for your DRM system.

There are three ways to play a VirtuosoAsset:

- **filePath**: If you have downloaded a standalone video file, such as an MP4 or ISMV file, thenyou can access the downloaded file directly for playback
- **playUsingPlaybackType:fromViewController:onSuccess:onFail:** This method automatically handles windowing, creation and maintenance of the VirtuosoClientHTTPServer instance used for playback, and playback itself. It uses a VirtuosoPlayerViewController for playback. Call this method from the view controller you wish to present the video player from. This method is unsupported for HSS video assets.
- **playUsingPlaybackType:andPlayer:onSuccess:onFail:** If you need to use a custom player, you can use this method instead. This method automatically handles windowing and creation and maintenance of the VirtuosoClientHTTPServer instance used for playback. It is your responsibility to further configure the player, to start playback if necessary, and to present the player in the UI hierarchy. If you are using a more complex player, additional integration steps may be required, such as using a VirtuosoClientHTTPServer instance directly.

Unregister Device: iOS

To unregister the local device:

```
[[VirtuosoDevice currentDevice]
    unregisterOnComplete:^(Boolean success, NSError *error)
    {
        //if success, update UI
        //else, handle error as appropriate
    }
];
```

DEVICES: DISABLED NOT DELETED
If devices you have remote wiped disappear from the device list in API calls, then all is well. Our backplane does not ever completely delete devices. It just disables them and disassociates them from whatever user ID they were last registered with when you call unregister/remote wipe. We filter out such devices from the basic web and device API call but those devices will still appear in the Penthera Cloud web admin GUI at present. So if you remote wipe a device and it vanishes from the API call device list but not the Penthera Cloud GUI, then all is working as expected.

Troubleshooting

The various SDK functions will log to your application log. If you are unable to determine the cause of issues on your own, contact Penthera support for assistance via email to support@penthera.com.

To help resolve issues, we will usually need a description of the scenario, any code snippets you may be able to provide, and ideally a complete *debug-level* log file from the application session in which you experience the issue.

In the Penthera iOS SDK, there are two subsystems whose logging is disabled by default: the local proxy and the interactions with the Penthera Cloud. To get the most detail for debugging and support, you will want to enable logging for those subsystems in addition to configuring the general SDK logging level to be more verbose. See Configure Logging: iOS [20] for details.

For issues related to communications off the device, such as errors during download, or failures of methods which rely on communications with the Penthera Cloud, our support will also ask for a log of the network communications captured with the Charles Proxy or other web debugging application. The Charles app can be found at https://www.charlesproxy.com. A Charles log of the full session starting at app launch is usually more helpful than a log of a partial session. Charles capture configuration can be confusing, but Penthera Support can help achieve the ideal setup.

What Next?

Browse through the example projects in the Tutorials directory of the SDK distributions. The ReadMe files explain what capabilities are covered by each example. The examples are available in various programming languages, and can easily be built & run in your IDE. When running the examples, use the demo/development keys and URL we have provided you for the Penthera Cloud.

In the SDK distribution, look for additional API details in the code-level documentation (javadoc for Android, header docs for iOS).

The following publications may also be of interest. These can be read online in the Penthera ZenDesk instance, and are also available as PDFs.

201: Developing with the iOS SDK (PDF, Penthera online support)

202: Developing with the Android SDK (PDF, Penthera online support)

301: iOS SDK Beyond the Basics (PDF, Penthera online support)

302: Android SDK Beyond the Basics (PDF, Penthera online support)

203: Best Practices (Penthera online support)

312: Known Issues (Penthera online support)