Authentec

Downloadable Agent for iOS with native player

- Automatically generated API docs -

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 DACachedContent Class Reference

Represents a content cached -totally or partially- on the local storage for offline playback.

Public Member Functions

- (void) - startCaching

  Starts caching this content onto the local storage.
- (void) - pauseCaching

  Interrupts the caching of this content if it was in progress.
- (void) - purgeCache

  Removes all the data stored in the cache for this content.

Static Public Member Functions

- (DACachedContent ∗) + cachedContentWithURL:ofType:

  A factory method that MUST be used to obtain instances of DACachedContent for each content that is to be stored in the cache.
- (NSArray ∗) + allCachedContentItems

  Returns a list of all the content items currently stored in the cache (whether partially or fully).

Properties

- NSURL ∗ contentURL

  the original content URL
- DAContentType contentType

  the original content type
- float totalDuration

  the total duration of the content (in seconds) or -1 if not available yet

- float cachedDuration

  the duration of the content that is currently cached (in seconds)
- DACachingStatus status

  the current status of the cached content
- BOOL isReadyToPlay

  whether there is a minimum amount of content in cache to allow offline playback

3.1.1 Detailed Description

Represents a content cached -totally or partially- on the local storage for offline playback.

3.1.2 Member Function Documentation

3.1.2.1 + (DACachedContent ∗) cachedContentWithURL: dummy(NSURL ∗) *contentURL* ofType:(DAContentType) *contentType*

A factory method that MUST be used to obtain instances of DACachedContent for each content that is to be stored in the cache.

The instance returned allows you to start or resume caching the content onto the local storage as a background process. If the content was already partly or entirely stored in the cache then the object will reflect the current status.

The following DAContentType's are exclusively supported:

- PIFFType (remotely hosted PIFF files)

Passing unsupported content types to this method will result in an exception being thrown.

This method is thread safe and if invoked multiple times with the same content URL it will return a pointer to the first instance of DACachedContent created. The life cycle of this instance is managed internally by the Agent.

See also

DACachedContent for more information about the exposed details.

Parameters

contentURL	The URL of a content supported by the Agent.
contentType	The type of the content pointed by the contentURL parameter.

Returns

an instance of the DACachedContent class.

3.1.2.2 + (NSArray ∗) allCachedContentItems

Returns a list of all the content items currently stored in the cache (whether partially or fully).

Returns

an array of DACachedContent instances.

3.1.2.3 - (void) startCaching

Starts caching this content onto the local storage.

If the content was already partly cached then the caching will resume where it left.

Note: this method dispatches the work asynchronously to a concurrent queue and returns inmediatly.

3.1.2.4 - (void) pauseCaching

Interrupts the caching of this content if it was in progress.

The caching process can be resumed later on by invoking the startCaching method again.

Note: this method returns inmediatly but the cancellation might take a bit longer to be effective.

3.1.2.5 - (void) purgeCache

Removes all the data stored in the cache for this content.

Note: this method returns inmediatly but the purge might take a bit longer to be effective.

3.1.3 Property Documentation

3.1.3.1 - (NSURL *) contentURL [read, write, copy]

the original content URL

3.1.3.2 - (DAContentType) contentType [read, write, assign]

the original content type

3.1.3.3 - (float) totalDuration [read, write, assign]

the total duration of the content (in seconds) or -1 if not available yet

3.1.3.4 - (float) cachedDuration [read, write, assign]

the duration of the content that is currently cached (in seconds)

3.1.3.5 - (DACachingStatus) status [read, write, assign]

the current status of the cached content

See also

DACachingStatus for a list of possible values

3.1.3.6 - (BOOL) isReadyToPlay [read, write, assign]

whether there is a minimum amount of content in cache to allow offline playback

3.2 DAContentInfo Class Reference

The object that holds all the exposed information about a protected content.

Properties

- BOOL isProtected

  Indicates whether or not the content is protected with PlayReady DRM.
- DARightsStatus rightsStatus

  The status of the rights installed for the content (if any).
- NSDate * licenseStartDate

  The start date set in the license (nil when none has been specified).
- NSDate * licenseEndDate

  The expiration date set in the license (nil when none has been specified).
- BOOL rightsExpireAfterFirstUse

  Indicates whether or not the rights will only expire after the first use.
- NSArray * audioTrackNames

  List of audio track names (as NSString's) available in the content.

3.2.1 Detailed Description

The object that holds all the exposed information about a protected content.

3.2.2 Property Documentation

3.2.2.1 - (BOOL) isProtected [read, assign]

Indicates whether or not the content is protected with PlayReady DRM.

3.2.2.2 - (DARightsStatus) rightsStatus [read, assign]

The status of the rights installed for the content (if any).

3.2.2.3 - (NSDate∗) licenseStartDate [read, retain]

The start date set in the license (nil when none has been specified).

3.2.2.4 - (NSDate∗) licenseEndDate [read, retain]

The expiration date set in the license (nil when none has been specified).

3.2.2.5 - (BOOL) rightsExpireAfterFirstUse [read, assign]

Indicates whether or not the rights will only expire after the first use.

Note: In this case the expiration date is not known until after the rights consumption has began (i.e. the playback has started).

3.2.2.6 - (NSArray∗) audioTrackNames [read, retain]

List of audio track names (as NSString's) available in the content.

Note: currently only Smooth Streaming streams support this feature.

3.3 DownloadableAgent Class Reference

The Downloadable Agent class handles the playback of PlayReady protected content with the iOS native player.

Public Member Functions

- (id) - initWithContentURL:type:

  Initializes the Agent with the URL of the given content.
- (MPMoviePlayerViewController ∗) - moviePlayerViewController

  Sets up a playback session and returns an instance of the MPMoviePlayerViewController class configured to play the protected content.
- (MPMoviePlayerController ∗) - moviePlayerController

  Sets up a playback session and returns an instance of the MPMoviePlayerController class configured to play the protected content.
- (AVPlayerItem ∗) - avPlayerItem

  Sets up a playback session and returns an instance of the AVPlayerItem class configured to play the protected content.
- (BOOL) - contentInfo:error:

  Checks the status and the installed rights for the content.
- (BOOL) - acquireLicenseWithDelegate:error:

  Initiates the license acquisition process for the protected content.
- (BOOL) - installLicense:

  [DEPRECATED: This method is no longer required with the new LicenseAcquisitionDelegate protocol]
- (BOOL) - removeLicense

  Removes any installed rights for the current content.

Static Public Member Functions

- (BOOL) + importDeviceModelCertificate:forModelCertificate:

  Imports the device model certificate of the given type into the Agent.
- (BOOL) + importDeviceModelPrivateKey:forModelCertificate:

  Imports the private key for the device model certificate of the given type into the Agent.
- (NSData ∗) + generatePlayReadyChallengeForDRMHeader:

  Generates a PlayReady license acquisition challenge for the given DRM header.
- (BOOL) + resetDRMDatabase

  Resets the DRM database.
- (BOOL) + isSecureDevice

  Performs a set of security checks in order to determine whether the device can be considered secure or not.
- (NSData ∗) + getPlayReadyDeviceID

  Returns the 16 bytes PlayReady GUID that uniquely identifies the current device.

Properties

- NSURL ∗ contentURL

  The URL of the content the Agent has been initialized with.
- DAContentType contentType

  The type of content the Agent has been initialized with.
- NSError ∗ lastError

  The last error that ocurred when invoking any of the methods.
- NSData ∗ customLicenseChallengeData

  Should you need to add custom data to the challenge that is sent to the license server during a license acquisition request, set this property before invoking the license acquisition method.
- DAClientControlHdmiMode hdmiMode

  The type of HDMI client control.
- UIView ∗ hdmiAlternativeImage

  The view to display as alternative image on the external screen(s).
- NSString ∗ selectedAudioTrackName

  Sets the name of the audio track to be used for Smooth Streaming streams with multiple audio tracks.

3.3.1 Detailed Description

The Downloadable Agent class handles the playback of PlayReady protected content with the iOS native player.

3.3.2 Member Function Documentation

3.3.2.1 - (id) initWithContentURL: dummy(NSURL ∗) *contentURL* type:(DAContentType) *contentType*

Initializes the Agent with the URL of the given content.

This method returns an instance of the Agent ready to be used with (and only with) this particular content. The Agent shall be used to acquire and manage rights for the content and to obtain an instance of the native player which can be used to play it (assuming the necessary rights are installed).

The instance of the Agent must be retained for the entire duration of the playback and released only after the player has been dismissed. The Agent can be reused to play the same content repeteadly.

Consult the Downloadable Agent User Manual for more info about the supported content formats.

Parameters

contentURL	The absolute URL of a supported content (i.e. URL of an HLS playlist or an SS manifest).
contentType	The type of content the given URL points to.

Returns

an instance of the DownloadableAgent class initialized with the given content.

3.3.2.2 - (MPMoviePlayerViewController *) moviePlayerViewController

Sets up a playback session and returns an instance of the MPMoviePlayerViewController class configured to play the protected content.

This class is a simple view controller suitable for full screen playback only.

See also

http://developer.apple.com/library/ios/#documentation/mediaplayer/reference/mpmoviepl
_class/Reference/Reference.html

Returns

an instance of the MPMoviePlayerViewController class (autoreleased) or 'nil' if the session setup failed

3.3.2.3 - (MPMoviePlayerController *) moviePlayerController

Sets up a playback session and returns an instance of the MPMoviePlayerController class configured to play the protected content.

This class provides full control over the media player, allowing playback in custom views that do not need to be full screen.

See also

http://developer.apple.com/library/ios/#documentation/mediaplayer/reference/-
MPMoviePlayerController_Class/MPMoviePlayerController/MPMoviePlayerController.-
html

Returns

an instance of the MPMoviePlayerController class (autoreleased) or 'nil' if the session setup failed

3.3.2.4 - (AVPlayerItem *) avPlayerItem

Sets up a playback session and returns an instance of the AVPlayerItem class configured to play the protected content.

This method uses the AVFoundation framework, an advanced Cocoa framework that allows a higher degree of customization and extra capabilities for the player.

See also

http://developer.apple.com/library/ios/#documentation/AVFoundation/Reference/-AVPlayerItem_Class/Reference/Reference.html#//apple_ref/doc/c_ref/AVPlayer-Item

Returns

an instance of the MPMoviePlayerController class (autoreleased) or 'nil' if the session setup failed

3.3.2.5 - (BOOL) contentInfo: dummy(DAContentInfo **) *outContentInfo* error:(NSError **) *outError*

Checks the status and the installed rights for the content.

This method can be used to check:

- if the content URL passed during the initialization of the Agent is considered valid.
- if the content is protected with a supported DRM schema or not.
- if there are valid rights installed and if so what is their validity period, in order to determine when is neccessary to acquire or renew a license for this content.
- if the content is currently reachable. If this is not the case an error will be returned.

NOTE: This operation may require network conectivity and take some time to complete, therefore it is not recommended to invoke this method from the main thread if you want your UI to remain responsive.

See also

the DAContentInfo class for more information about the details exposed by this API.

Parameters

outContentInfo	[in/out] A pointer of type DAContentInfo which will be assigned to a new DAContentInfo instance
outError	[in/out] A pointer of type NSError which will be assigned to an instance only when an error ocurred

Returns

YES when the operation was completed successfully, NO otherwise

3.3.2.6 - (BOOL) acquireLicenseWithDelegate: dummy(id< LicenseAcquisitionDelegate >) *delegate* error:(NSError **) *outError*

Initiates the license acquisition process for the protected content.

The 'delegate' parameter is optional:

- If it is set to 'nil', the whole license acquisition protocol will be executed behind the scenes by the Agent. The return value will indicate whether the license was acquired and installed successfully or not.

- If a delegate object is passed the Agent will use it to delegate the sending of the request to the license server. Should the server require the device to join a domain in order to issue licenses, the delegate object should also implement this optional functionality (

  See also

  the LicenseAcquisitionDelegate protocol).

  The delegated license acquisition scenario allows for a lot more flexibility, such as for example when you want to route the request through a third party server rather than contacting the license server directly, in order to perform some sort of user authentication before issuing the licenses. You may also use it to add custom headers or cookies to the HTTP request.

The successful acquisition and installation of a license does not guarantee that the content can be played, as the license may contain rights that allow playback only in a future date. This condition should always be checked by verifying the installed rights after the acquisition and before starting the playback.

NOTE: This operation may involve network activity and take some time to complete, therefore it is not recommended to invoke this method from the main thread.

Parameters

delegate	Pointer to an instance of a class implementing the LicenseAcquisitionDelegate protocol or nil.
outError	[in/out] A pointer of type NSError which will be assigned to an instance only when an error ocurred

Returns

YES when the operation was completed successfully, NO otherwise

3.3.2.7 - (BOOL) installLicense:

[DEPRECATED: This method is no longer required with the new LicenseAcquisitionDelegate protocol]

Installs a new PlayReady license.

This method shall be used to install pass on the response received from the license server after a successful delegated license acquisition. This data is typically a UTF8 encoded XML license. The Agent will only verify that the license has been installed successfully but not whether the license grants playback rights at the current date & time.

Parameters

license	The response received from the license server

Returns

YES when the license was installed successfully, NO otherwise

3.3.2.8 - (BOOL) removeLicense

Removes any installed rights for the current content.

This method only affects the content the Agent was initialized with. It can be used to manually revoke the rights before the actual expiration date.

Returns

YES when the license was removed successfully or there was no installed rights, NO otherwise

3.3.2.9 + (BOOL) importDeviceModelCertificate: dummy(NSData ∗) *certificate* forModelCertificate:(ModelCertificateType) *type*

Imports the device model certificate of the given type into the Agent.

It's necessary to have successfully imported all the required model certificates and keys before attempting to create an instance of the Agent, otherwise the instantiation will fail.

Parameters

certificate	The device model certificate encrypted
type	The type of model certificate

Returns

YES if the import was successful, NO otherwise

3.3.2.10 + (BOOL) importDeviceModelPrivateKey: dummy(NSData ∗) *privateKey* forModelCertificate:(ModelCertificateType) *type*

Imports the private key for the device model certificate of the given type into the Agent.

It's necessary to have successfully imported all the required model certificates and keys before attempting to create an instance of the Agent, otherwise the instantiation will fail.

Parameters

privateKey	The private key encrypted
type	The type of model certificate this private key is for

Returns

YES if the import was successful, NO otherwise

3.3.2.11 + (NSData ∗) generatePlayReadyChallengeForDRMHeader: dummy(NSData ∗) *header*

Generates a PlayReady license acquisition challenge for the given DRM header.

This class method provides a way to generate a license challenge for a DRM header that is not tied to any particular content. For instance, when using the Microsoft Mediaroom service, the mobile device needs to authenticate with the server in order to be able to acquire licenses. The authentication is performed by sending a PlayReady challenge, which you can obtain by invoking this method, together with some specific HTTP headers (single sign-on token, etc).

Parameters

header	The DRM header you want to generate a challenge for.

Returns

The PlayReady license challenge generated or 'nil' if the invocation failed.

3.3.2.12 + (BOOL) resetDRMDatabase

Resets the DRM database.

This is a convenience method to perform a full reset of the DRM database, thus all the installed licenses will be lost. Note that it will be necessary to have previously imported the device model certificates for this operation to succeed.

The operation is not reversible.

Returns

YES when the operation was successful, NO otherwise

3.3.2.13 + (BOOL) isSecureDevice

Performs a set of security checks in order to determine whether the device can be considered secure or not.

This method provides a quick way to check whether the security of the operating system may have been compromised, i.e. the device has been 'jailbroken'. This means that the user has root access to the system and can potentially modify parts of it and/or install third party software components that could be used to break into the security of the system or the Agent. In this case you may opt to forbid or limit the playback protected content.

Note: this method should not be considered as 100% reliable as there might be ways to circunvent the detections.

Returns

NO when the checks have determined that the device is likely to be jailbroken, YES otherwise.

3.3.2.14 + (NSData ∗) getPlayReadyDeviceID

Returns the 16 bytes PlayReady GUID that uniquely identifies the current device.

3.3.3 Property Documentation

3.3.3.1 - (NSURL*) contentURL [read, retain]

The URL of the content the Agent has been initialized with.

3.3.3.2 - (DAContentType) contentType [read, assign]

The type of content the Agent has been initialized with.

3.3.3.3 - (NSError*) lastError [read, retain]

The last error that ocurred when invoking any of the methods.

3.3.3.4 - (NSData*) customLicenseChallengeData [read, write, retain]

Should you need to add custom data to the challenge that is sent to the license server during a license acquisition request, set this property before invoking the license acquisition method.

NOTE: The Agent does not interpret this data, it simply includes it in the license challenge message.

3.3.3.5 - (DAClientControlHdmiMode) hdmiMode [read, write, assign]

The type of HDMI client control.

Consult the documentation for more details.

3.3.3.6 - (UIView*) hdmiAlternativeImage [read, write, retain]

The view to display as alternative image on the external screen(s).

3.3.3.7 - (NSString*) selectedAudioTrackName [read, write, retain]

Sets the name of the audio track to be used for Smooth Streaming streams with multiple audio tracks.

The name string must match the value of the "Name" attribute of one of the StreamIndex elements with the attribute Type="audio".

Use the contentInfo method to retrieve the list of available audio track names. If no track name is selected the Agent will default to the the first audio track found in the manifest.

3.4 <LicenseAcquisitionDelegate> Protocol Reference

Protocol that defines the required method a delegate object must implement in order to perform a delegated license acquisition.

Public Member Functions

- (NSData *) - performDelegatedLicenseAcquisition:withChallenge:

  A callback method that shall execute a delegated license acquisition request.
- (NSData *) - performDelegatedJoinDomain:withChallenge:

  A callback method that shall execute a delegated join domain request.

3.4.1 Detailed Description

Protocol that defines the required method a delegate object must implement in order to perform a delegated license acquisition.

3.4.2 Member Function Documentation

3.4.2.1 - (NSData *) performDelegatedLicenseAcquisition: dummy(NSURL *) *licenseAcquisitionURL* withChallenge:(NSData *) *challenge* [required]

A callback method that shall execute a delegated license acquisition request.

The Agent will invoke this callback when it has been instructed to initiate the license acquisition protocol in delegated mode. A digitally signed challenge (in XML format) is generated by the Agent and passed to the delegate together with the license acquisition URL.

Once the delegate has submitted the challenge to the license server (typically using a SOAP request), the response must be returned to the Agent to be processed. If the request has failed or the response is not a valid SOAP message, you should directly return nil.

Parameters

licenseAcquisitionURL	The license acquisition URL
challenge	The challenge data encoded in UTF8

Returns

a NSData object containing the body of the HTTP response or nil.

3.4.2.2 - (NSData *) performDelegatedJoinDomain: dummy(NSURL *) *domainControllerURL* withChallenge:(NSData *) *challenge* [optional]

A callback method that shall execute a delegated join domain request.

The Agent will invoke this callback when it has detected that the server requires the device to join a domain in order to acquire a license. A digitally signed challenge (in XML format) is generated by the Agent and passed to the delegate together with the URL of the domain controller.

Once the delegate has submitted the challenge to the domain server (typically using a SOAP request), the response must be returned to the Agent to be processed. If the request has failed or the response is not a valid SOAP message, you should directly return nil.

Parameters

domain-ControllerURL	The URL of the domain controller
challenge	The challenge data encoded in UTF8

Returns

a NSData object containing the body of the HTTP response or nil.

Chapter 4

File Documentation

4.1 DACachedContent.h File Reference

Data Structures

- class DACachedContent

  Represents a content cached -totally or partially- on the local storage for offline playback.

4.2 DAContentInfo.h File Reference

Data Structures

- class DAContentInfo

  The object that holds all the exposed information about a protected content.

4.3 DADataTypes.h File Reference

Enumerations

- enum ModelCertificateType { PlayReady_OEM_Certificate, WMDRM_OEM_Certificate }

  List of device model certificate types required by the PlayReady SDK.
- enum DAContentType { HTTPLiveStreamingType, SmoothStreamingType, PIFFType }

  List of content types supported by the Downloadable Agent.
- enum DAErrorType { DAErrorInternalDRMAgentError, DAErrorContentDownloadFailed, DAErrorInvalidOr-UnexpectedContent, DAErrorSilentLicenseAcquisitionFailed, DAErrorDelegatedLicenseAcquisitionFailed, D-AErrorDelegatedJoinDomainFailed, DAErrorInvalidServerResponse, DAErrorJoinDomainNotSupported, DA-ErrorDelegatedJoinDomainNotSupported }

  List of all error codes reported by the Agent via NSError instances.
- enum DARightsStatus { DANoRights, DAValid, DARightsInFuture, DAExpired, DAUntrustedTime, DA-Unknown }

List of possible license status for a content.

- enum DAClientControlHdmiMode { DAHdmiDefault, DAHdmiDisable, DAHdmiAllow, DAHdmiAudioOnly, DAHdmiUseAlternative }

  List of possible client control HDMI modes.

- enum DACachingStatus { DACachingStatusNotCached, DACachingStatusInProgress, DACachingStatusPaused, DACachingStatusCompleted, DACachingStatusFailed }

  List of possible status for a content stored in cache.

4.3.1 Enumeration Type Documentation

4.3.1.1 enum ModelCertificateType

List of device model certificate types required by the PlayReady SDK.

Both types of model certificates, together with their corresponding private keys, must be imported into the Agent before attempting to use it. See the static methods available for importing model certificates.

Enumerator:

PlayReady_OEM_Certificate PlayReady OEM device model certificate or private key.

WMDRM_OEM_Certificate WindowsMedia DRM OEM device model certificate or private key.

4.3.1.2 enum DAContentType

List of content types supported by the Downloadable Agent.

Enumerator:

HTTPLiveStreamingType The content URL must point to a valid HTTP Live Streaming playlist, either the root or any variant playlist.

SmoothStreamingType The content URL must point to an Smooth Streaming manifest.

PIFFType The content URL must point to a PIFF file (.ismv), either stored locally on the file system (the URL must then be the absolute path, with or without the '`file://`' schema) or hosted on a remote HTTP server.

4.3.1.3 enum DAErrorType

List of all error codes reported by the Agent via NSError instances.

Enumerator:

DAErrorInternalDRMAgentError The DRM Agent failed to complete an operation. This is usually a severe error which requires further investigation. Please collect all possible logs from the device in order to provide enough proper context.

DAErrorContentDownloadFailed The content URL could not be retrieved successfully. Typically this indicates a network issue, a malformed URL or a missing file in the server. If the content URL points to a HLS playlist which contains other variant playlists, the failure to retrieve one of these would also trigger this error.

DAErrorInvalidOrUnexpectedContent The content retrieved is invalid or does not match the expected format. The file(s) retrieved from the given URL do not match the expected format for the type of content the Agent was initialized with.

DAErrorSilentLicenseAcquisitionFailed The Agent failed to perform the silent license acquisition. Indicates that the license acquisition was not successful, typically because the license server is unreachable or returned an error or an invalid license.

DAErrorDelegatedLicenseAcquisitionFailed The delegate object failed to perform the license acquisition.

DAErrorDelegatedJoinDomainFailed The delegate object failed to perform the join domain operation.

DAErrorInvalidServerResponse The delegate object returned an invalid or unexpected server response which did not contain valid rights objects.

DAErrorJoinDomainNotSupported The Agent does not support the join domain operation required by the license server.

DAErrorDelegatedJoinDomainNotSupported The delegate object does not support the join domain operation required by the license server. To prevent this error please implement the corresponding method of the LicenseAcquisitionDelegate protocol.

4.3.1.4 enum DARightsStatus

List of possible license status for a content.

Enumerator:

DANoRights There are no valid rights.

DAValid There are valid rights available now.

DARightsInFuture Rights will be available in the future.

DAExpired Right have expired.

DAUntrustedTime Rights may exist but cannot be used as no trusted time is available.

DAUnknown Default value for non DRM protected content.

4.3.1.5 enum DAClientControlHdmiMode

List of possible client control HDMI modes.

Enumerator:

DAHdmiDefault The agent controls the HDMI based on info from the license.

DAHdmiDisable The agent will fail the OPL check if HDMI video out is detected.

DAHdmiAllow The agent will allow HDMI video out even if the license doesnt allow.

DAHdmiAudioOnly The agent will allow only audio via HDMI.

DAHdmiUseAlternative The client will provide alternative image for displaying on HDMI output. Use the property hdmiAlternativeImage for setting the alternative UIView.

4.3.1.6 enum DACachingStatus

List of possible status for a content stored in cache.

Enumerator:

DACachingStatusNotCached the content is not cached

DACachingStatusInProgress the content caching process is in progress

DACachingStatusPaused the content caching process has been cancelled/paused

DACachingStatusCompleted the content is now fully stored on cache

DACachingStatusFailed the content caching process has failed due to errors

4.4 DownloadableAgent.h File Reference

Data Structures

- protocol <LicenseAcquisitionDelegate>

  Protocol that defines the required method a delegate object must implement in order to perform a delegated license acquisition.
- class DownloadableAgent

  The Downloadable Agent class handles the playback of PlayReady protected content with the iOS native player.

Variables

- NSString *const DownloadableAgentErrorDomain

  The error domain used in all errors reported by the Agent.
- NSString *const DownloadableAgentPlaybackRightsNotAvailableNotification

  Notification posted when the rights for a protected content become unavailable during the playback.
- NSString *const DownloadableAgentUntrustedSystemTimeNotification

  Notification posted when the local system time can not be trusted because a rollback to the system clock has been detected.
- NSString *const DownloadableAgentDidDownloadInvalidContentNotification

  Notification posted during playback when the Agent has determined that a file downloaded upon request of the player is invalid, meaning that the file format is not the expected (i.e.
- NSString *const DownloadableAgentDidReceiveServerErrorNotification

  Notification posted during playback when the Agent has received an HTTP error code as response when trying to download a file requested by the player.
- NSString *const DownloadableAgentPlaybackShouldBeAbortedNotification

  Notification posted during playback when the Agent has encountered a critical issue that prevents from processing a request properly and is not expecting to be able to recover successfully from it, therefore it is recommended to abort the playback.
- NSString *const DownloadableAgentPlaybackWillBeAbortedNotification

  Notification posted during playback when the Agent has detected a contidition that prevents the playback from continuing.
- NSString *const DownloadableAgentDidPostLogMessageNotification

Notification posted to make the all log messages produced by the Agent available at the application level (UI), in order to facilitate untethered or on the move testing and debugging.

- NSString ∗const DownloadableAgentDidReceiveServerErrorCodeUserInfoKey

  User info dictionary key to access the HTTP status code (as a NSNumber object)
- NSString ∗const DownloadableAgentDidReceiveServerErrorURLUserInfoKey

  User info dictionary key to access the failed URL (as a NSString object)

4.4.1 Variable Documentation

4.4.1.1 NSString∗ const DownloadableAgentErrorDomain

The error domain used in all errors reported by the Agent.

4.4.1.2 NSString∗ const DownloadableAgentPlaybackRightsNotAvailableNotification

Notification posted when the rights for a protected content become unavailable during the playback.

Typically this means that the time based license for the content has expired, but could also indicate a mismatch between the client and the server's time in such a way that the server is delivering licenses with the start date set to a time in the future (relative to the current time in the client).

The usual reaction to this notification is to try and acquire a new license, either silently or with consent of the user. The playback should resume automatically when a new valid license has been installed. If the player has enough buffered content to cover the time spent in acquiring the license, it may be able to continue playing uninterruptedly.

There is no userInfo dictionary.

4.4.1.3 NSString∗ const DownloadableAgentUntrustedSystemTimeNotification

Notification posted when the local system time can not be trusted because a rollback to the system clock has been detected.

Under this condition the Agent is unable to verify the validity of time based rights, therefore content decryption will not be possible until the date & time has been corrected. This does not affect to the life-time licenses (rights with no expiration date).

Note: after this notification is fired, network access will be required in order to validate the date & time again. A small degree of tolerance is always accepted.

There is no userInfo dictionary.

4.4.1.4 NSString∗ const DownloadableAgentDidDownloadInvalidContentNotification

Notification posted during playback when the Agent has determined that a file downloaded upon request of the player is invalid, meaning that the file format is not the expected (i.e.

not a valid HLS playlist or video segment). The most common cause for this error is that the device is roaming into a WiFi hotspot which requires some sort of user authentication or to accept a challenge in order to allow Internet access, and therefore the Agent is getting an HTML page rather than the content requested.

In this case it is recommened to alert the user to verify the wireless data connection before resuming the playback. Note that the player will internally keep trying to obtain this file again and again, so the playback should be resumed automatically once the device has a working data connection.

There is no userInfo dictionary.

4.4.1.5 NSString* const DownloadableAgentDidReceiveServerErrorNotification

Notification posted during playback when the Agent has received an HTTP error code as response when trying to download a file requested by the player.

HTTP status codes in the 4xx and 5xx range are reported as errors. This is mostly a convenience notification to log and report potential service issues within your content distribution network.

Note that redirections (3xx) are supported and honored by the Agent. Other failures such as network timeouts are not reported with this notification.

The userInfo dictionary of this notification contains two items, the URL of the file that caused the error and the HTTP status code returned, both of which can be retrieved using the following keys:

- DownloadableAgentDidReceiveServerErrorURLUserInfoKey (the object is a NSString)
- DownloadableAgentDidReceiveServerErrorUserInfoKey (the object is a NSNumber)

4.4.1.6 NSString* const DownloadableAgentPlaybackShouldBeAbortedNotification

Notification posted during playback when the Agent has encountered a critical issue that prevents from processing a request properly and is not expecting to be able to recover successfully from it, therefore it is recommended to abort the playback.

Otherwise the Agent is likely to enter into an error loop as the player silently tries to request the same or the next piece of content.

In this case the user should be told to report the error for further investigation.

There is no userInfo dictionary.

4.4.1.7 NSString* const DownloadableAgentPlaybackWillBeAbortedNotification

Notification posted during playback when the Agent has detected a contidition that prevents the playback from continuing.

In this case the Agent will immediatly shutdown certain internal components that will eventually force the player to quit even if you choose to ignore this notification.

As of the current version, this notification is only fired when a security breach is detected (i.e. an attacker is trying to bypass the DRM security), but in the future it could be used to report any other condition that would cause a playback failure.

There is no userInfo dictionary.

4.4.1.8 NSString* const DownloadableAgentDidPostLogMessageNotification

Notification posted to make the all log messages produced by the Agent available at the application level (UI), in order to facilitate untethered or on the move testing and debugging.

This notification is not available in production builds.

There is no userInfo dictionary.

4.4.1.9 NSString* const DownloadableAgentDidReceiveServerErrorCodeUserInfoKey

User info dictionary key to access the HTTP status code (as a NSNumber object)

See the DownloadableAgentDidReceiveServerErrorNotification notification.

4.4.1.10 NSString* const DownloadableAgentDidReceiveServerErrorURLUserInfoKey

User info dictionary key to access the failed URL (as a NSString object)

See the DownloadableAgentDidReceiveServerErrorNotification notification.