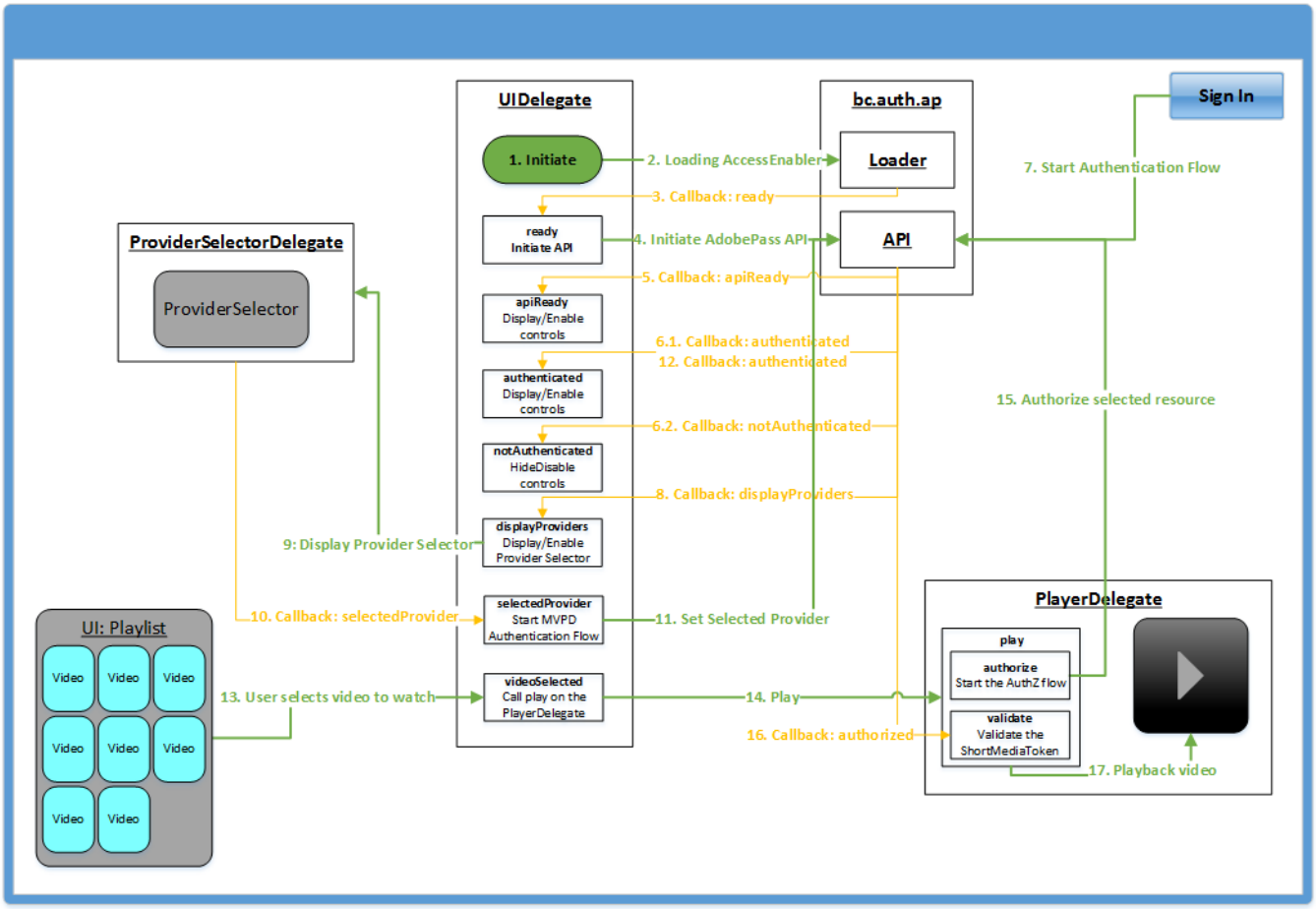

Usage

1. Instantiate the bc.auth.ap.Loader(profile, debugEnabled)
   `var loader = new bc.auth.ap.Loader(profile, debugEnabled);`
   `profile:String` – The Adobe Pass profile (`bc.auth.ap.Loader.STAGING` or `bc.auth.ap.Loader.PRODUCTION`)
   `debugEnabled:Boolean` – If true loads the Adobe Pass with debugging enabled

2. Instantiate the bc.auth.ap.API() – Assign to a accessible variable
   `_that.authAPI = new bc.auth.ap.API();`

3. Initiate the loading of the AccessEnabler: Invoke the load(callbackDelegate) on the Loader

```
loader.load({
    ready: function(accessEnabler) {
        //Library loaded and ready to use
        //accessEnabler:Object - The AccessEnabler instance

        //This is a good place to initiate the API (see below)
        //and to hide any loading progress UI
    },
```

```
    loaded: function() {
        //Library loaded, but not ready to use just yet
    },
    error: function(loadError) {
        //Error loading the library
        //loadError:bc.auth.model.AuthError – The Error object

        //The error callback could also be
        //an instance of bc.example.ErrorHandler, such as:
        //error: new bc.example.ErrorHandler();
    }
});
```

4. Initiate the API: Invoke the init(accessEnabler, callbackDelegate) on the API
   `accessEnabler` – This is the `accessEnabler` instance returned by the Loader `ready` callback above

```
_that.authAPI.init(accessEnabler, {
    apiReady: function(providers) {
        //The API is initiated and ready to use
        //providers:bc.auth.model.Provider[] – All available providers

        //This is a good place to show/enable some controls or wrappers
    },
    authenticated: function(provider) {
        //The user is authenticated with the returned provider
        //provider:bc.auth.model.Provider – Authenticated provider

        //This is a good place to hide/disable the sign-in button and
        //show/enable the sign-out button
        //It's also a good place to update the UI with the provider logo,
        //and display protected video items as unlocked
    },
    notAuthenticated: function(errorMsg) {
        //The user is not authenticated

        //This is a good place to hide/disable the sign-out button and
        //show/enable the sign-in button
        //It's also a good place to update the UI and remove any provider info
        //such as logo, and to display protected video items as locked
    },
    displayProviders: function(providers) {
        //The user needs to select what provider to authenticate with
        //providers:bc.auth.model.Provider[] – All available providers

        //Render your Provider Selector
        //Make sure to use a custom provider whitelist and only
        //display the providers from that whitelist
        //Capture and action the selected provider (more about this below)
    },
    renderIframe: function(provider, iframeId, width, height) {
        //The selected provider wants to load its login screen in an iframe
        //provider:bc.auth.model.Provider – The selected provider
        //iframeId:String – The id/name the Iframe must have
        //width:int - The required width of the Iframe
        //height:int – The required height of the Iframe

        //Render the Iframe "popup"
    },
    removeIframe: function(iframeId) {
        //The authentication is completed and the Iframe can now be destroyed
        //iframeId:String – The id of the Iframe to remove and destroy
```

```
    },
    authorized: function(requestorId, requestedResourceId, token) {
        //The user is authorized to view the selected video
        //requestorId:String – The requestorId
        //requestedResourceId:String – The authorized resourceId
        //token:String – The Adobe Pass Short Media Token

        //Informational, there is another authorized callback
        //as part of the authorize(callbackDelegate) call
    },
    notAuthorized: function(requestedResourceId, errorCode, errorDetails) {
        //The user is not authorized to view the selected video
        //requestedResourceId:String – The non-authorized resourceId
        //errorCode:String – The error code
        //errorDetails:String – The errorDetails (sometimes empty)

        //Informational, there is another notAuthorized callback
        //as part of the authorize(callbackDelegate) call
    },
    error: function(authError) {
        //Something went wrong
        //authError:bc.auth.model.AuthError – The Error object

        //The error callback could also be
        //an instance of bc.example.ErrorHandler, such as:
        //error: new bc.example.ErrorHandler();
    }
});
```

5. Setup click handlers for the Sign In/Out buttons
6. User clicks the Sign In button: Invoke the startAuthentication([callbackDelegate]) on the API, which will then trigger the displayProviders(providers) callbacks
   (Optional: Manipulate the UI, like disable/hide the Sign In button)

```
_that.authAPI({
    authenticated: function(provider) {
        //The user is authenticated with the returned provider
        //provider:bc.auth.model.Provider – Authenticated provider
    },
    notAuthenticated: function(errorMsg) {
        //The user is not authenticated
    },
    displayProviders: function(providers) {
        //The user needs to select what provider to authenticate with
        //providers:bc.auth.model.Provider[] – All available providers
    },
    renderIframe: function(provider, iframeId, width, height) {
        //The selected provider wants to load its login screen in an iframe
        //provider:bc.auth.model.Provider – The selected provider
        //iframeId:String – The id/name the Iframe must have
        //width:int - The required width of the Iframe
        //height:int – The required height of the Iframe
    },
    removeIframe: function(iframeId) {
        //The authentication is completed and the Iframe can now be destroyed
        //iframeId:String – The id of the Iframe to remove and destroy
    },
    error: function(authError) {
        //Something went wrong
        //authError:bc.auth.model.AuthError – The Error object
```

```
            //The error callback could also be
            //an instance of bc.example.ErrorHandler, such as:
            //error: new bc.example.ErrorHandler();
        }
    });
```

7. The Provider Selector is displayed and the User selects a Provider: Invoke the setSelectedProvider(provider) on the API, which will then trigger either a new window/tab to open or the renderIframe callback to be invoked in which the provider's login page will then be loaded.
8. The Provider Selector is displayed and the User cancels without selecting: Invoke the setSelectedProvider (null) on the API
9. User clicks on a protected video item: Invoke the authorize(resourceId, callbackDelegate) on the API

```
_that.authAPI(videoItem.resourceId, {
    authorized: function(requestorId, requestedResourceId, token) {
        //The user is authorized to view the selected video
        //requestorId:String - The requestorId
        //requestedResourceId:String - The authorized resourceId
        //token:String - The Adobe Pass Short Media Token

        //Validate the token, retrieve the video renditions and play the video
    },
    notAuthorized: function(requestedResourceId, errorCode, errorDetails) {
        //The user is not authorized to view the selected video
        //requestedResourceId:String - The non-authorized resourceId
        //errorCode:String - The error code
        //errorDetails:String - The errorDetails (sometimes empty)

        //Alert the user that they were not authorized to view the video
        //as part of the authorize(callbackDelegate) call
    },
    error: function(authError) {
        //Something went wrong
        //authError:bc.auth.model.AuthError - The Error object

        //The error callback could also be
        //an instance of bc.example.ErrorHandler, such as:
        //error: new bc.example.ErrorHandler();
    }
});
```

10. User clicks the Sign Out button: Invoke the logout() on the API, which will then trigger the notAuthenticated(errorMsg) callback

Components

Authentication: Adobe Pass

bc.auth.ap.Loader

This class is responsible for loading the AccessEnabler (a.k.a. Adobe Pass library).

The AccessEnabler comes in two different flavors, AccessEnabler.swf and AccessEnabler.js, where the former should always be attempted to be loaded first as it provides better security and flexibility, whereas the latter should only be used as a fallback option.

The Loader will mange all aspects of loading the AccessEnabler, including graceful fallback from AccessEnabler.swf to AccessEnabler.js as well as dynamically inject the SWFObject library if required.

Flow:

- ***load(callbackDelegate)***: Starts the loading process of the AccessEnabler.
  callbackDelegate: { **loaded**: Function(), **ready**: Function(accessEnabler), **error**: Object }

  1. loadNext() gets invoked, and attempts to load the first libType defined in the TYPES array (normally SWF), passing _internalDelegate.**loaded** as the successCallback and itself as the failCallback
     (Note: the failCallback also accumulates any error that's returned)

  2. _internalLoader.SWF(successCallback, failCallback) gets invoked

     2.1. Starts off by ensuring that the SWFObject is loaded and dynamically injects the SWFObject if it's not.

     2.2. Inject AccessEnabler.swf

     2.3. If successful **AND** Flash is blocked
     Applies a workaround for Flash blocking by blowing up the size on the loaded AccessEnabler.swf container to enable the blocking message and required action to become visible to the user

     2.4. If successful

     2.4.1. Invokes the successCallback function, which will then invoke the **loaded** callback method

     2.4.2. Waits for the AccessEnabler global callback ***swfLoaded*** (which if Flash was not blocked will almost happen instantly)

     2.4.3. ***swfLoaded*** callback gets invoked

     2.4.3.1. Hides AccessEnabler.swf container again (which was used for the Flash blocking workaround in 2.3)

     2.4.3.2. Disables the Adobe Pass default MVPD picker (only applies to SWF)

     2.4.3.3. Invokes the **ready** callback method and passes along the accessEnabler instance

     2.5. If not successful
     Invokes the failCallback which will re-invoke the loadNext method with an updated index, which will then attempt to load the next libType from the TYPES array (normally JS), passing _internalDelegate.**loaded** as the successCallback and itself as the failCallback

2.5.1. _internalLoader.JS(successCallback, failCallback) gets invoked

2.5.1.1. Inject AccessEnabler.js

2.5.1.2. If successful

2.5.1.2.1. Invokes the successCallback function, which will then invoke the **loaded** callback method

2.5.1.2.2. Waits for the AccessEnabler global callback ***entitlementLoaded***

2.5.1.2.3. ***entitlementLoaded*** callback gets invoked
Invokes the **ready** callback method and passes along the accessEnabler instance

2.5.1.3. If not successful
Invokes the failCallback which will re-invoke the loadNext method with an updated index, which will then be out of bounds for the TYPES array and trigger a call to the **error** callback method

- ***getAccessEnabler()***: Returns the AccessEnabler instance
- ***getAccessEnablerId()***: Returns the AccessEnabler DOM Id (only applies for SWF)

bc.auth.ap.API

This is an abstraction layer for the AccessEnabler. It will also handle the multi-step initiation internally, parse the config XML and providers, manages the AccessEnabler callbacks and re-fires callback methods on the appropriate callbackDelegate (see Usage for details about the callbacks).

- ***init(accessEnabler, callbackDelegate)***: Initiates the API
accessEnable: The AccessEnabler instance
callbackDelegate: See Usage

1. Defines the AccessEnabler callbacks
2. Assigns the accessEnabler to a local class variable
3. Binds the an error handler to the accessEnabler errorEvent
4. Invokes the accessEnabler.setRequestorId(REQUESTOR_ID, null, REQUESTOR_CONFIG)
5. AccessEnabler Callback setConfig(configXML) gets triggered

5.1. Parse the configXML – Returns true if the current domain is valid to use for the specified REQUESTOR_ID, false otherwise

5.1.1. Extracts and serialize the associated providers into bc.auth.model.Provider[]

5.1.2. Validates the current domain by comparing it to the whitelisted domains in the config

5.2. If domain is valid, then attempts to get the current/previous selected provider accessEnabler.getSelectedProvider

6. AccessEnabler Callback selectedProvider(provider) gets triggered

6.1. Looks up the matching bc.auth.model.Provider from the _that.provider variable and store it into the _that.selectedProvider variable.

6.2. _that.apiReady is false -> Invokes the accessEnabler.checkAuthentication()

7. AccessEnabler Callback setAuthenticationStatus(isAuthenticated, errorCode) gets triggered

7.1. Converts the argument isAuthenticated into a Boolean

7.2. _that.apiReady is false -> Triggers the API Callback apiReady(providers)

7.3. Triggers the API Callback authenticationStatus(isAuthenticated, errorCode)

7.4. If Authenticated: Triggers the API Callback authenticated(provider)

7.5. If Not Authenticated: Triggers the API Callback notAuthenticated(errorCode)

- ***getSelectedProvider()***:

1. Returns the currently selected provider, or invokes the accessEnabler.getSelectedProvider()

- ***checkAuthentication([authNCallbackDelegate])***:

1. Invokes the accessEnabler.checkAuthentication()

2. AccessEnabler Callback setAuthenticationStatus(isAuthenticated, errorCode) gets triggered

2.1. Converts the argument isAuthenticated into a Boolean

2.2. Triggers the API Callback authenticationStatus(isAuthenticated, errorCode)

2.3. If Authenticated: Triggers the API Callback authenticated(provider)

2.4. If Not Authenticated: Triggers the API Callback notAuthenticated(errorCode)

- ***startAuthentication([authNCallbackDelegate])***:
Starts the Authentication Flow

1. Invokes the accessEnabler.getAuthentication()

2. If Authenticated: AccessEnabler Callback setAuthenticationStatus(isAuthenticated, errorCode) gets triggered

2.1. Converts the argument isAuthenticated into a Boolean

2.2. Triggers the API Callback authenticationStatus(isAuthenticated, errorCode)

2.3. If Authenticated: Triggers the API Callback authenticated(provider)

3. If Not Authenticated: AccessEnabler Callback displayProviderDialog(providers) gets triggered
    3.1. Serialize the providers into bc.auth.model.Provider[]
    3.2. Triggers the API Callback displayProvider(serializedProviders)
    3.3. User selects their provider – See setSelectedProvider below

- ***logout([authNCallbackDelegate])***:
    1. Invokes the accessEnabler.logout()
    2. AccessEnabler Callback setAuthenticationStatus(isAuthenticated, errorCode) gets triggered
        2.1. Converts the argument isAuthenticated into a Boolean
        2.2. Triggers the API Callback authenticationStatus(isAuthenticated, errorCode)
        2.3. If Not Authenticated: Triggers the API Callback notAuthenticated(errorCode)

- ***setSelectedProvider(provider)***:
    1. Invokes the accessEnabler.setSelectedProvider(provider)
    2. Iframe Provider -> AccessEnabler Callback createIFrame(width, height) gets triggered
        2.1. Triggers the API Callback renderIframe(selectedProvider, iframeId, width, height)
        2.2. The Provider's login page gets rendered into the Iframe
        2.3. When authentication is complete -> AccessEnabler Callback destroyIFrame() gets triggered
            2.3.1. Triggers the API Callback removeIframe(iframeId)
    3. Redirect Provider -> New window/tab opens with the Provider's login page
    4. AccessEnabler Callback setAuthenticationStatus(isAuthenticated, errorCode) gets triggered
        4.1. Converts the argument isAuthenticated into a Boolean
        4.2. Triggers the API Callback authenticationStatus(isAuthenticated, errorCode)
        4.3. If Authenticated: Triggers the API Callback authenticated(provider)
        4.4. If Not Authenticated: Triggers the API Callback notAuthenticated(errorCode)

- ***authorize(resourceId, [authZCallbackDelegate])***:
    1. Invokes the accessEnabler.getAuthorization (resourceId)
    2. If Authorized -> AccessEnabler Callback setToken(requestedResourceId, token) gets triggered

2.1. If not Authenticated -> Triggers the API Callback authenticated(selectedProvider)

2.2. Triggers the API Callback authorized(requestorId, resourceId, token)

3. If Not Authorized -> AccessEnabler Callback tokenRequestFailed(requestedResourceId, errorCode, errorDetails) gets triggered

3.1. Triggers the API Callback notAuthorized(requestedResourceId, errorCode, errorDetails)

bc.auth.ap.ErrorMap

Defines the error codes that could be returned from Adobe Pass and adds additional info.

When an error is triggered from the AccessEnabler, the errorCode will be extracted, which will then together with this mapping be used to create a bc.auth.model.AuthError object by using the bc.auth.model.AuthError.Factory, which will then be used for the error callback

Authentication: Models

bc.auth.model.Provider

Defines the Provider with id, name and logo.

bc.auth.model.AuthError

Defines the AuthError with errorCode, errorMessage, errorDetails, isFatal and category.

bc.auth.model.AuthError.Factory

Static factory class for creating AuthError objects

Helpers

bc.helper.LoadHelper

Static helper class to dynamically load and inject SWF and JS files. It also has convenience methods to dynamically load and inject JQuery, and SWFObject.

Example

index.php

Markup for the landing page with a placeholder/wrapper for the Video Player. It also includes the playlistContainer.php

playlistContainer.php

Builds the playlist container populated with video items. The data is generated by the getPlaylist.php

getPlaylist.php

Generates MediaItem object for the playlistContainer.php.

new MediaItem(MEDIA_ID, IS_PROTECTED, VIDEO_FILE, IMAGE_FILE_PREFIX, VIDEO_TITLE)

php/MediaItem.php

Defines the MediaItem, with videoId, video (url), thumbnail (url), poster (url), title, isProtected and resourceId

bc.example.UIDelegate

Handles all the interaction and initiation – See Usage and Flow for details

bc.example.ErrorHandler

Handles the error callbacks, by assigning an instance of the ErrorHandler to the error callback

```
…
error: new bc.example.ErrorHandler()
```

bc.example.ProviderSelectorDelegate

Example implementation of a Provider Selector to manage the UI rendering and interaction.

```
displayProviders: function(providers) {
  var providerSelectorDelegate = new bc.example.ProviderSelectorDelegate();
  providerSelectorDelegate.injectProviderSelector(providers, {
    selectedProvider: function(provider) {
      //The User selected a provider

      //Call the setSelectedProvider(provider) on the API
    },
    close: function() {
      //The User cancelled the Provider Selector

      //Call the setSelectedProvider(null) on the API to reset the API
    }
  });
}
```

bc.example.PlayerDelegate

Example implementation of a Player delegate to manage the UI rendering, authorization, token validation, getting video renditions and initiate playback.

Instantiation is done in the init method of the UIDelegate

When the User selects/clicks a video, the UIDelegate is capturing the video metadata into a videoItem object. This videoItem is then passed into the PlayerDelegate.play method.

See Usage #9 for details.

See getMedia.php (below) for details about the validation and getting video renditions.

getMedia.php

1. Extracts the requestorId, resourceId, shortMediaToken and mediaId from the request

2. Validates the token by using exec, verify shell script and Adobe's MediaToken validator jar
3. Generates the Response item which will include the video renditions
4. Return the JSON response

Adobe Pass Media Token Validator

verifier-1.1.1511-production.jar

Adobe Pass' Media Token validator

verify

Shell script for invoking the Media Token validator and capturing the output

Complete Happy Auth Flow

1. [UIDelegate] Initiate
   1.1. [UIDelegate] Instantiate bc.auth.ap.Loader
   1.2. [UIDelegate] Instantiate bc.auth.ap.API
   1.3. [PlayerDelegate] bc.example.PlayerDelegate
      1.3.1. [PlayerDelegate] Inject NextGen player CSS
      1.3.2. [PlayerDelegate] Renders Player element
      1.3.3. [PlayerDelegate] Inject NextGen player JS
      1.3.4. [PlayerDelegate] Instantiate player with videojs
2. [UIDelegate] Invoke load on the bc.auth.ap.Loader instance passing along a callback delegate object
   2.1. [Loader] Assigns the callback delegate object to a local private variable
   2.2. [Loader] Attempts to load the first library type (SWF)
      2.2.1. [Loader] Ensures SWFObject is loaded, and if not goes ahead and loads it
      2.2.2. [Loader] Loads the SWF library
      2.2.3. [Loader] If successful calls the <CALLBACK_DELEGATE>.loaded and waits for the swfLoaded callback
         2.2.3.1. [Loader] swfLoaded callback gets invoked and <CALLBACK_DELEGATE>.ready is called, passing along the accessEnabler instance
         2.2.3.2. [Loader] Skip [2.3]
      2.2.4. [Loader] If unsuccessful, go to [2.3]
   2.3. [Loader] If [2.2] failed: Attempt to load the second library type (JS)
      2.3.1. [Loader] Loads the JS library
      2.3.2. [Loader] If successful calls the <CALLBACK_DELEGATE>.loaded and waits for the entitlementLoaded callback
         2.3.2.1. [Loader] The entitlementLoaded callback gets invoked and <CALLBACK_DELEGATE>.ready is called, passing along the accessEnabler instance
      2.3.3. [Loader] If unsuccessful, calls the <CALLBACK_DELEGATE>.error
3. [UIDelegate] Callback ready(accessEnabler)
   3.1. [UIDelegate] Hide loading spinner

4. [UIDelegate] Invoke init on the bc.auth.ap.API instance passing along the accessEnabler instance and a callback delegate object
    4.1. [API] Assigns accessEnabler and callback delegate object to local private variables
    4.2. [API] Calls the accessEnabler.setRequestor and waits for the setConfig callback
    4.3. [API] setConfig callback gets invoked
    4.4. [API] Parses the configXML and creates an array of bc.auth.model.Provider objects
    4.5. [API] Calls the accessEnabler.setProvider to get (if any) the previously selected provider
    4.6. [API] Checks that the apiReady flag has not been set and then invokes accessEnabler.checkAuthentication
    4.7. [API] setAuthenticationStatus callback gets invoked
5. [API] Checks that the apiReady flag has not been set and then <CALLBACK_DELEGATE>.apiReady is called
    5.1. [API] <CALLBACK_DELEGATE>.authenticationStatus is called
    5.2. [API] If Authenticated: <CALLBACK_DELEGATE>.authenticated is called passing along the current bc.auth.model.Provider object
    5.3. [API] If NOT Authenticated: <CALLBACK_DELEGATE>.notAuthenticated is called
6. [UIDelegate] AuthN Callbacks
    6.1. [UIDelegate] Callback authenticated (provider)
        6.1.1. [UIDelegate] Display current provider logo
        6.1.2. [UIDelegate] Hide/Disable <Sign In> button
        6.1.3. [UIDelegate] Show/Enable <Sign Out> button
        6.1.4. [UIDelegate] Enable PlayerDelegate
        6.1.5. [UIDelegate] Update the “Locked” icon to an “Unlocked” icon for each protected Video item
    6.2. [UIDelegate] Callback notAuthenticated ()
        6.2.1. [UIDelegate] Remove any provider logo
        6.2.2. [UIDelegate] Hide/Disable <Sign Out> button
        6.2.3. [UIDelegate] Show/Enable <Sign In> button
        6.2.4. [UIDelegate] Disable PlayerDelegate
        6.2.5. [UIDelegate] Update the “Unlocked” icon to an “Locked” icon for each protected Video item
7. [UIDelegate] <Sign In> button is clicked: Invoke startAuthentication on the bc.auth.ap.API instance passing along a AuthN callback delegate object
    7.1. [API] Assigns the AuthN callback delegate object to a local private variable
    7.2. [API] Calls the accessEnabler.setRequestor and waits for the callback
    7.3. [API] displayProviderDialog callback gets invoked
        7.3.1. [API] Creates an array of bc.auth.model.Provider objects from the untyped providers array argument
        7.3.2. [API] <CALLBACK_DELEGATE>.displayProviders is called (this is from the API.init call)
        7.3.3. [API] <AUTHN_CALLBACK_DELEGATE>.displayProviders is called
8. [UIDelegate] Callback displayProviders(bc.auth.model.Provider[])
    8.1. [UIDelegate] Instantiate bc.example.ProviderSelectorDelegate

9. [UIDelegate] Invoke injectProviderSelector on the bc.example.ProviderSelectorDelegate instance passing along a callback delegate object
    - 9.1.1. [ProviderSelectorDelegate] Assigns the callback delegate object to a local private variable
    - 9.1.2. [ProviderSelectorDelegate] Creates the Provider Selector UI with appropriate click event handlers
    - 9.1.3. [ProviderSelectorDelegate] Provider Selected: selectedProvider gets invoked
        - 9.1.3.1. [ProviderSelectorDelegate] Remove Provider Selector UI
        - 9.1.3.2. [ProviderSelectorDelegate] <CALLBACK_DELEGATE>.selectProvider is called passing along the selected bc.auth.model.Provider object
10. [UIDelegate] Callback selectedProvider(bc.auth.model.Provider)
11. [UIDelegate] Invokes setSelectedProvider on the bc.auth.ap.API instance passing along the bc.auth.model.Provider object
    - 11.1. [API] Assigns the selected provider to a private local variable
    - 11.2. [API] Calls the accessEnabler.setSelectedProvider and waits for the callback
    - 11.3. [API] setAuthenticationStatus callback gets invoked
        - 11.3.1. [API] <CALLBACK_DELEGATE>.authenticationStatus is called
        - 11.3.2. [API] If Authenticated: <CALLBACK_DELEGATE>.authenticated is called passing along the current bc.auth.model.Provider object
        - 11.3.3. [API] If NOT Authenticated: <CALLBACK_DELEGATE>.notAuthenticated is called
12. [UIDelegate] Callback authenticated (provider)
    - 12.1. [UIDelegate] Display current provider logo
    - 12.2. [UIDelegate] Hide/Disable <Sign In> button
    - 12.3. [UIDelegate] Show/Enable <Sign Out> button
    - 12.4. [UIDelegate] Enable PlayerDelegate
    - 12.5. [UIDelegate] Update the “Locked” icon to an “Unlocked” icon for each protected Video item
13. [UIDelegate] <Video Item> is clicked: videoSelected is invoked
    - 13.1. Pause currently playing video (if any)
    - 13.2. Capture video metadata from the selected Video Item
    - 13.3. Set the background to <VIDEO_ITEM>.poster
14. [UIDelegate] Invokes play on the b bc.example.PlayerDelegate instance passing along the captured <VIDEO_ITEM> object
    - 14.1. [PlayerDelegate] Assigns the Video Item object to a private local variable
    - 14.2. [PlayerDelegate] Checks is the video item is protected (i.e. Needs authorization)
    - 14.3. [PlayerDelegate] If Not Protected:
        - 14.3.1. [PlayerDelegate] Set the player src to the Video Item video url
        - 14.3.2. [PlayerDelegate] Starts playback
15. [PlayerDelegate] If Protected:
    - 15.1. [PlayerDelegate] Invokes authorize on the bc.auth.ap.API instance passing along the Video Item resourceId and a AuthZ callback delegate
        - 15.1.1. [API] Assigns the AuthZ callback delegate object to a local private variable
        - 15.1.2. [API] Calls the accessEnabler. getAuthorization and waits for the callback

15.1.3. [API] AuthZ Successful: setToken callback gets invoked and <CALLBACK_DELEGATE>:s.authorized is called passing long requestorId, requested resourceId and short media token

15.1.4. [API] AuthZ Unsuccessful: tokenRequestFailed callback gets invoked and <CALLBACK_DELEGATE>:s.notAuthorized is called passing along requested resourceId, errorCode and errorDetails

16. [PlayerDelegate] Callback authorized (requestorId, requestedResourceId, token)
    16.1. Validate token and return the actual video rendition(s)
17. Playback Video