Integrating the Roku Advertising Framework

- Getting Started
- Use Cases
    - Client Side Ad Insertion
        - Single Preroll Ad Pod
        - Sequential Rendering
        - Custom Scheduling
        - Example
    - Enabling Nielsen Digital Ad Ratings
    - Frequency Capping and Targeting using RIDA
        - RIDA Specific Parameters
    - Custom Buffering Screens
        - Default Ad Buffering Screen
        - Custom Buffering Screen Using Content Meta-data (Fixed Positioning)
        - Custom Buffering Screen Using Content Meta-data (Custom Positioning)
    - Custom Ad Parsing and Rendering
        - Custom Ad Parsing
        - Custom Ad Rendering
- Requirements for Server Side Ad Insertion
    - Implementation Details
- Testing your RAF Implementation
- API Reference
    - Construction
        - Roku_Ads()
    - Control
        - General
            - fireTrackingEvents(adStructure as Object, ctx as Object) as Boolean
        - Client Ad Insertion
            - getAds(msg as string) as Object
            - showAds(ads as Object, ctx as Object, view as Object) as Boolean
        - Server-Stitched Ads
            - stitchedAdHandledEvent(msg as Object, player as Object) as roAssociativeArray
    - Configuration
        - General
            - setAdUrl(url as String)
            - getAdUrl() as String
            - setContentGenre(genres as String, kidsContent as Boolean)
            - setContentId(id as String)
            - setContentLength(length as Integer)
            - setAdPrefs(useRokuAdsAsFallback as Boolean, maxRequests as Integer)
            - setAdConstraints(maxHeight as Integer, maxWidth as Integer, maxBitrate as Integer, supportedMimeTypes as Object)
            - setAdBreaks(contentLength as Integer, adBreakTimes as Integer)
            - importAds(adPodArray as Object)
            - enableJITPods(enabled as Boolean)
            - setTrackingCallback(callback as Function, obj as Object)
            - setDebugOutput(enabled as Boolean)
            - getLibVersion() as String
        - Nielsen DAR
            - setNielsenGenre(genre as String)
            - setNielsenAppId(id as String)
            - setNielsenProgramId(id as String)
        - Nielsen DCR
            - getNielsenContentData() as String
        - General Audience Measurement
            - enableAdMeasurements(enabled)
        - Server-Stitched Ads

        - stitchedAdsInit(adPodArray as roArray)
    - Buffer Screen Customization
        - setAdBufferScreenContent(contentMetaData as Object)
        - enableAdBufferMessaging(enableMsg as Boolean, enableProgressBar as Boolean)
        - setAdBufferScreenLayer(zOrder as Integer, contentMetaData as Object)
        - clearAdBufferScreenLayers()
        - setAdBufferRenderCallback(callback as Function, obj as Object, timeout as Integer)
- URL Parameter Macros
    - Example
- Ad Structure
- Tracking
- Roku Genre Tags
- Nielsen DAR Genre Tags

Getting Started

The RAF library is intended to allow developers to focus their effort on the core design of their applications, and provide them with the ability to quickly and easily integrate video advertising features with minimal impact to the rest of their application. This section provides an overview of the fundamental steps required to integrate with such applications. Other developers may wish to have more control over certain features within their applications, such as rendering the ads with a custom UI, or integrating the event tracking with a 3rd-party analytics API. The API Reference provid es more detailed information on the video ad API, and the Use Cases section offers examples designed to cover a variety of different scenarios. The video ad features are provided as a common library deployed and managed as a hidden channel.

The following line must be placed in the manifest file for any applications using the Roku Ad Framework library:

Manifest Entry

```
bs_libs_required=roku_ads_lib
```

Client applications do not include any additional BrightScript modules as part of their own channel package. Instead, the “Library” keyword is used. The following line should be the first entry in your `main.brs` file:

```
Library "Roku_Ads.brs"
```

The library interface is obtained by calling the constructor with no arguments:

```
adIface = Roku_Ads()
```

If the client application is using Roku’s default ad server to fill ads, no other configuration is required. Otherwise, configure the ad URL before making the ad request call:

> You may wish to check URL Parameter Macros to see if any parameter values in the ad URL should be replaced with the provided macros.

```
adIface.setAdUrl(myAdUrl)
```

Aside from the Configuration interface, there are two main methods used to control ad parsing and rendering. The first, getAds(), makes the initial request to the ad server, parses the server response, and returns the structure of ads to be rendered prior to, or during playback, of the selected content:

```
adPods = adIface.getAds()
```

Any preroll ads present in the returned set of ad pods can be immediately rendered by calling:

```
shouldPlayContent = adIface.showAds(adPods)
```

Checking and acting on the return value here allows the application to determine if the user exited out of the ad (for example, by pressing the “Back” button on the remote) and return to the content selection screen before playing the main content.

If the application is only showing preroll ads, the above five lines are sufficient. If the ad server URL was configured for additional midroll and/or postroll ads, the client application should periodically call getAds() with the message from the content video playback loop to determine when to halt the content playback and render the ads:

Calling getAds() in a while Loop

```
while shouldPlayContent
  videoMsg = wait(0, contentVideoScreen.GetMessagePort())
  adPods = adIface.getAds(videoMsg)
  if adPods <> invalid and adPods.Count() > 0
    contentVideoScreen.Close() ' stop playback of content
    shouldPlayContent = adIface.showAds(adPods) ' render current ad pod
    if shouldPlayContent
      ' *** Insert client app’s resume-playback code here
    end if
  end if
  ' *** Insert client app’s video event handler code here
end while
```

Please note that the system overlay behavior has been modified in Roku OS 8. Every time RAF is rendered, the Video node will not be in focus. For the Roku system overlay to slide out when the * button is clicked, the Video node should be set to be in focus. Otherwise, the channel retains control over the * button and will need to handle button presses on their own. To set the Video node in focus again, use the following code snippet:

```
sub init()
m.top.setFocus(true)
setVideo()
end sub
```

Use Cases

The video ad library is intended to support a variety of use cases, depending on the requirements of the application. The sample code presented here is provided for illustrative purposes of each of these cases and is not intended to represent required or optimal usage in client applications. For clarity and concision, error and object validity checking are omitted in these examples.

In all cases, the library must first be included and its interface constructed as described in Getting Started. Additionally, unless the client application is using Roku's default ad URL (which currently provides only a single ad), the ad URL must be configured before requesting an ad pod:

> You may wish to check URL Parameter Macros to see if any parameter values in the ad URL should be replaced with the provided macros.

```
Library "Roku_Ads.brs"

adIface = Roku_Ads()
adIface.setAdUrl(myAdUrl)
adPods = adIface.getAds()
```

At this point, the ad server response has been fully parsed and is available in the adPods Ad Structure.

Client Side Ad Insertion

If the client application has no need for custom UI or user interaction during ad rendering, it is recommended to use the default rendering method showAds(). This method handles rendering and control of interactive and video ads, as well as displaying basic messaging UI (e.g., "Your program will continue after these messages") and feedback UI ("Ad 1 of 3"). Calling showAds() with an array of ad pods (such as the structure returned from the initial call to getAds()) will render the first pod scheduled as a preroll. Calling it with a single ad pod will render that pod, regardless of its `renderSequence` attribute.

Single Preroll Ad Pod

Just call showAds() with the adPods value that the application obtained above:

```
shouldPlayContent = adIface.showAds(adPods)
```

Note that the return value should still be checked to see if the user exited the ad, and therefore should also exit out of content playback back to a selection screen.

Sequential Rendering

Typically, if the ad service URL is configured to return a slate of ad pods to be presented throughout the presentation of the content, it is sufficient to use getAds() as an event listener in the content video event loop, as described in Getting Started, to determine when the scheduled ad breaks should occur:

Sequential Ad Pod Rendering Example

```
shouldPlayContent = adIface.showAds(adPods)
while shouldPlayContent
  videoMsg = wait(0, contentVideoScreen.GetMessagePort())
  adPods = adIface.getAds(videoMsg)
  if adPods <> invalid and adPods.Count() > 0
    contentVideoScreen.Close() ' stop playback of content
    shouldPlayContent = adIface.showAds(adPods) ' render current ad pod
    if shouldPlayContent
      ' *** Insert client app's resume-playback code here
    end if
  end if
  ' *** Insert client app's video event handler code here
end while
```

This usage of getAds() also automatically implements the default policy that determines whether to re-render ads that have already been viewed. This policy permits the user to rewind content up to 5 minutes before a scheduled ad break before displaying that ad pod again.

Custom Scheduling

Alternatively, there may be instances where the application must have greater control over when ad breaks occur. As an example, if the ad service is configured to return a VAST 2.0 response without temporal ad breaks, the application could re-interpret this unstructured response and schedule rendering of those ads as necessary:

Custom Ad Scheduling Example

```
adBreakSchedule = [adBreakTime1, adBreakTime2, adBreakTime3]
scheduledPods = []
adBreakIndex = 0
for each ad in adPods[0].ad
  ' schedule one ad per ad break
  scheduledPods.Push({viewed : false,
                      renderSequence : "midroll",
                      duration : ad.duration,
                      renderTime : adBreakSchedule[adBreakIndex],
                      ads : [ad]
                      })
  adBreakIndex = adBreakIndex + 1
end for
```

Default sequential rendering could then be used by first importing this new `scheduledPods` ad structure, as described in Custom Ad Parsing and Rendering.

Or, if the library's ad rendering features are desired without the default ad scheduling mechanism, the application may completely control which ads are scheduled for rendering:

Complete Ad Rendering Control Example

```
shouldPlayContent = true
adBreakIndex = 0
while shouldPlayContent
  videoMsg = wait(0, contentVideoScreen.GetMessagePort())
  if videoMsg.isPlaybackPosition()
    curPos = videoMsg.GetIndex()
    nextPod = scheduledPods[adBreakIndex]
    if curPos > nextPod.renderTime and not nextPod.viewed
      contentVideoScreen.Close() ' stop playback of content
      shouldPlayContent = adIface.showAds(nextPod) ' render next ad pod
      adBreakIndex = adBreakIndex + 1
      if shouldPlayContent
        ' *** Insert client app's resume-playback code here
      end if
    end if
  end if
  ' *** Insert client app's video event handler code here
end while
```

This type of custom ad scheduling may also be necessary if the client application relies on multiple ad services to fill its ad slots. For this case, separate calls are made to setAdUrl(), followed by getAds(), to get the ads from each service. Then scheduling and rendering can be done using

one of the methods described above.

Example

For an example, see: FullRAFSceneGraphSample.zip

Enabling Nielsen Digital Ad Ratings

As third parties such as Nielsen measure all video advertising in the digital ecosystem, Roku mandates that all channels enable Nielsen DAR in their Roku apps. Even if the publisher does not have a direct relationship with Nielsen, it is important to enable that in the app so as to avoid an app update when/if there is a decision made to enable Nielsen.

Nielsen DAR must first be enabled using:

```
enableNielsenDAR(true)
```

This need only be done once and can be called at any point before any ads are rendered.

Next, a Nielsen application ID must be set for accurate campaign measurement. If you are not working with Nielsen directly, you can use Roku's default App Id: `P2871BBFF-1A28-44AA-AF68-C7DE4B148C32`

```
setNielsenAppId("P2871BBFF-1A28-44AA-AF68-C7DE4B148C32")
```

Each content item will then need to be tagged appropriately. The required attributes are:

- title (NielsenProgramId) - this value should the title of a movie or series and should not uniquely identify episodic content
- length of the content (in seconds)
- Nielsen genre (see Nielsen DAR Genre Tags)

```
setNielsenProgramId("TED Talks")
setContentLength(1200)
setNielsenGenre("GV")
```

The content-specific metadata should be set when new content is selected before any ads are rendered, so that the values can be added to the custom Nielsen impression tags for the ads. If any of these metadata are unknown for a particular piece of content (for example, live streams may not have a specific content length), applications should call the appropriate function with an empty parameter to clear the value.

Frequency Capping and Targeting using RIDA

The Roku ID for Advertising is a device identifier available to Roku publishers for development and marketing purposes. It is designed to generally follow the guidelines established for the IDFA (Identifier for Advertising) used on other platforms. The RIDA limits disclosure of users' identifying information and allows the ability to opt-out of remarketing or reset the ID at any time. It is the ID that Roku offers to its publishers to enable frequency capping and targeted advertising on the Roku platform.

The app can use the GetRIDA() API to get the RIDA for the device. RIDA should only be passed for audience targeting if "Limit Ad Tracking" is not set in the Roku Settings UI. If the RIDA is not set, it should still be used for frequency capping and when the ad server may need to know the user state. The app can check that by calling the IsRIDADisabled() API.

Retrieving RIDA Example

```
Function getRIDA() as String
    RIDA = ""
    dev_info = createObject("roDeviceInfo")
    if not dev_info.IsRIDADisabled()
       RIDA = dev_info.GetRida()
   end if

    return RIDA
End Function
```

RIDA Specific Parameters

Many leading ad servers such as FreeWheel and DFP have Roku specific parameters in their ad request that the app can pass the RIDA in.

For Freewheel, the parameters are:

```
_fw_did=rida:<roku-device-id>
_fw_vcid2=<roku-device-id>
```

Example

```
url =
http://my.ad.server.net/?my_first_param=MyFirstValue&other_param=SomeOtherValue&_fw_did=
rida:<roku-device-id>
```

For DFP, the parameter is called `rdid`. Additional details available here: https://support.google.com/dfp_premium/answer/6238701?hl=en

Example

```
url =
http://pubads.g.doubleclick.net/gampad/request-type?my_first_param=MyFirstValue&other_pa
ram=SomeOtherValue&rdid=<roku-device-id>
```

Replace <roku-device-id> with the RIDA for audience targeting.

Custom Buffering Screens

RAF also supports multiple ways of customizing the buffering screen which appear before ad playback.

Default Ad Buffering Screen

The default ad buffering screen displays a message and a progress bar. Both attributes can either be enabled or disabled using enableAdBufferMessaging().

Custom Buffering Screen Using Content Meta-data (Fixed Positioning)

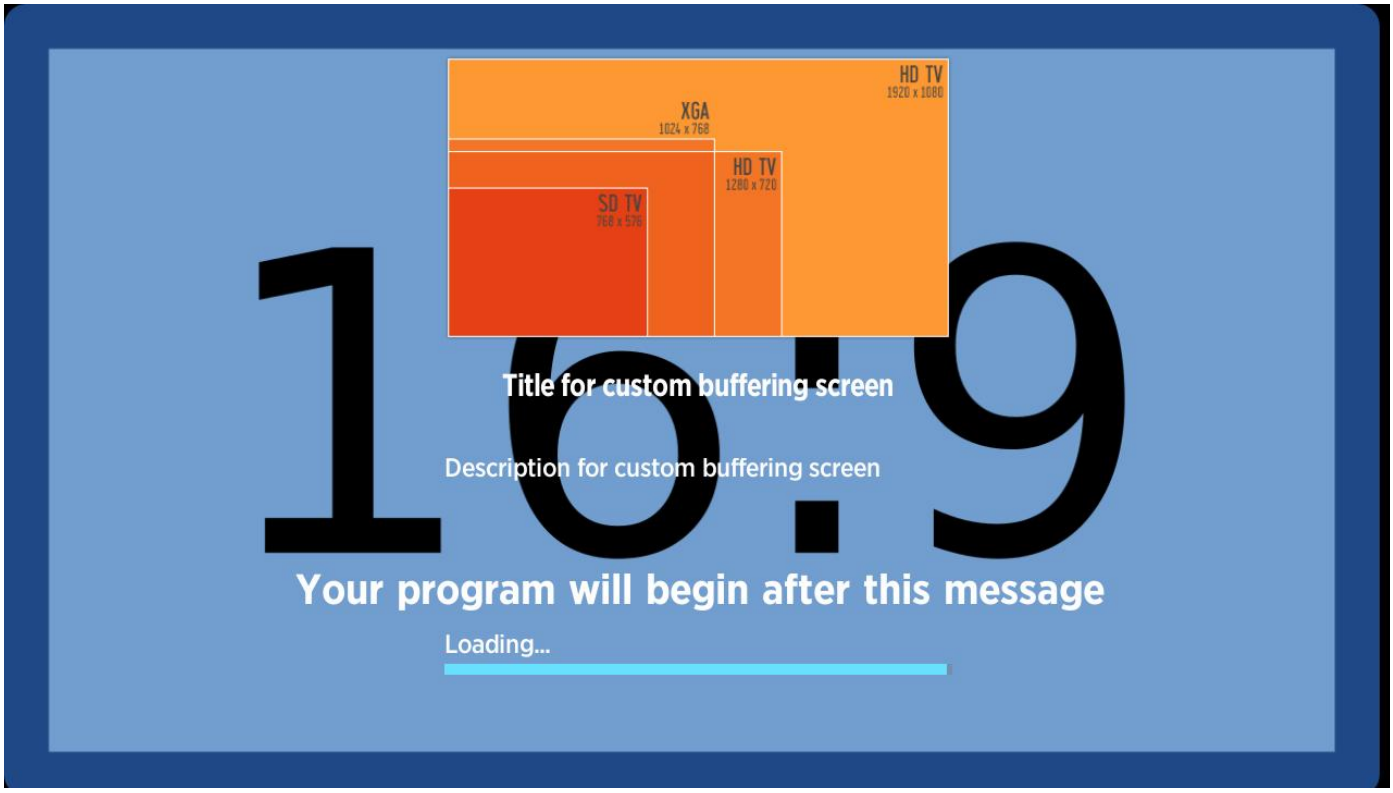
The buffering screen can also be customized by passing a Content meta-data object to setAdBufferScreenContent(). This function does not support custom positioning. Instead, use SetAdBufferScreenLayer() as described in the next section.

The supported content meta-data attributes are:

Attribute	Positioning	Example (below image)
HDBackgroundImageUrl	Aligned to top-left corner	https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/Aspect-ratio-16x9.svg/1280px-Aspect-ratio
SDBackgroundImageUrl	Aligned to top-left corner	n/a
HDPosterUrl	Aligned to top-center	http://static.commentcamarche.net/ccm.net/faq/images/0-BX4VeV6H-resolution-comparison-s-.png
SDPosterUrl	Aligned to top-center	n/a
Title	Center-aligned relative to and displayed below PosterUrl	"Title for custom buffering screen"
Description	Left-aligned relative to PosterUrl	"Description for custom buffering screen"

```
bufferScreenContent = {}
bufferScreenContent.HDBackgroundImageUrl =
"https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/Aspect-ratio-16x9.svg/1280px-
Aspect-ratio-16x9.svg.png"
bufferScreenContent.HDPosterUrl =
"http://static.commentcamarche.net/ccm.net/faq/images/0-BX4VeV6H-resolution-comparison-s
-.png"
bufferScreenContent.Title = "Title for custom buffering screen"
bufferScreenContent.Description = "Description for custom buffering screen"

adIface.SetAdBufferScreenContent(bufferScreenContent)
```

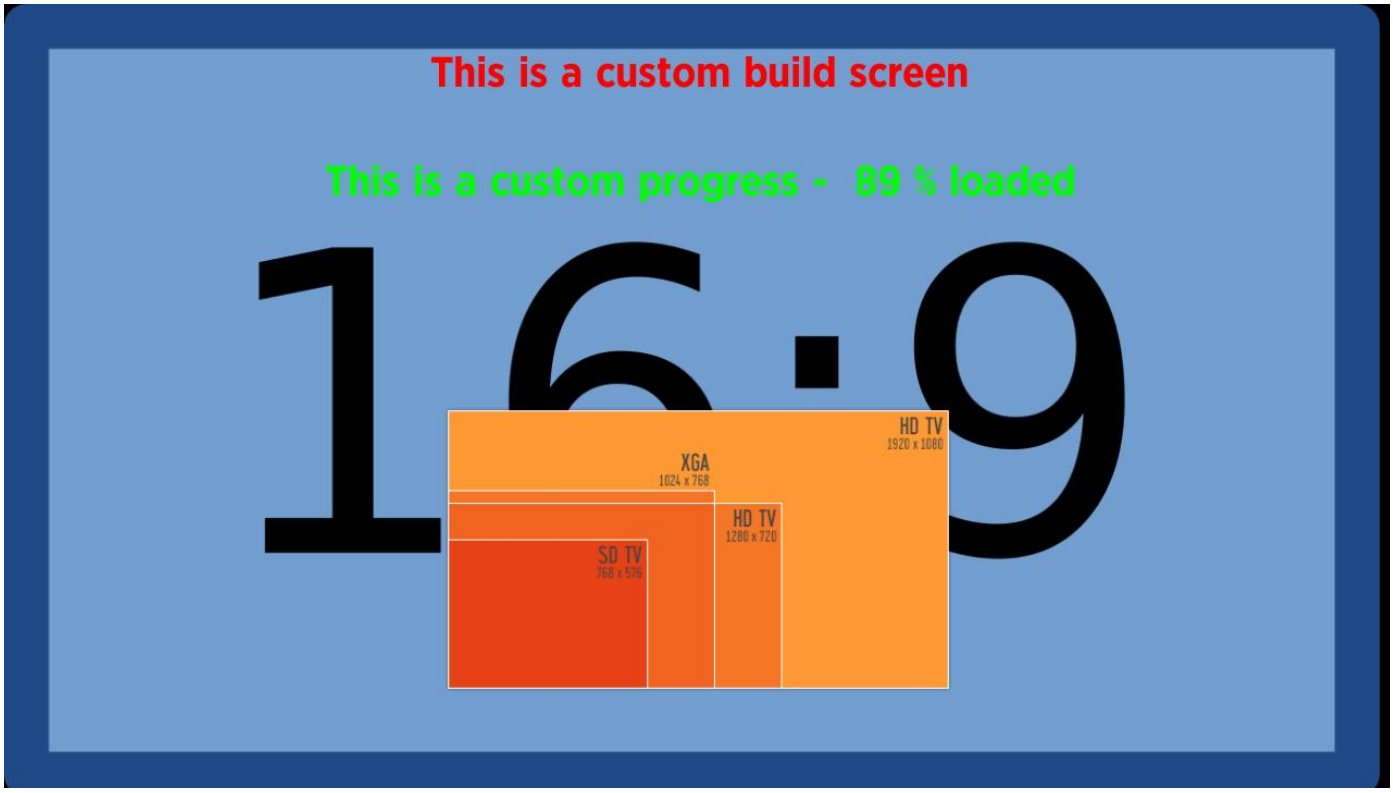


Custom Buffering Screen Using Content Meta-data (Custom Positioning)

For a complete custom buffering screen, setAdBufferScreenLayer() allows the same content meta-data attributes as setAdBufferScreenContent(), but enables you to customize the positioning and other roImageCanvas attributes.

Custom Buffering Screen Using Layers

```
layers = [
 {Url: BackgroundImageUrl}
    {Url: PosterUrl, TargetRect : {x : 405, y : 370, w : 467, h : 262}}
    {
     Text : "This is a custom build screen"
        TextAttrs : { Color : "#FF0000", HAlign : "Center", Font : "Large"}
        TargetRect : {y : 50, h : 30}
    }
]
adIface.setAdBufferScreenLayer(2, layers)
```



For an example on different custom buffering screen implementations, see CustomBufferScreenSceneGraphSample.zip.

Custom Ad Parsing and Rendering

Custom ad parsing and rendering requires explicit approval from Roku to ensure proper ad delivery and quality. Please reach out to advertising@roku.com for verifying your implementation prior to submitting your channel for publication.

Custom Ad Parsing

Some applications may use an ad service that returns an unsupported response format, but can still take advantage of the library's ad rendering features. In this case, the application is reponsible for requesting and parsing the ad response and structuring the ads into an array of pods according to the required Ad Structure. Scheduling and rendering can then proceed as described in **Client Side Ad Insertion** by first calling the importAds() method with the ad structure constructed externally by the client:

```
adIface.importAds(myAdPodArray)
```

Custom Ad Rendering

Client applications may elect to control the ad rendering within the application, either to provide custom UI while loading ads, or because the ads are rendered by a method unsupported by the video ad library (e.g., by server-side video stitching). It is sufficient to make a single call to getAds() to get the entire Ad Structure. The client application is then responsible for using the `streams` data to render the ads, and also must trigger the Tracking events when the requisite conditions are met.

The fireTrackingEvents() method is used to trigger event tracking, including client macro replacement and processing of Nielsen DAR beacons. The `adStructure` parameter passed in to this method can be either an `adPod` or a single ad from Ad Structure, depending on whether the event is relevant for the entire pod (such as `PodStart` or `PodComplete`) or for a single ad (all other event types). The `ctx` parameter should contain either a `type` string from the Tracking events or a `time` numerical value for firing time-dependent events such as quartile beacons. If both are specified, the `type` value takes precedence. Custom tracking event types can be added and fired as well, but client code should attempt to use conventional event types such as "Impression" where appropriate, as certain operations like Nielsen DAR parameter substitution rely on the "Impression" event type.

> The `type` values are case-sensitive, so will only fire events with names that match exactly the type specified.

Client code should fire all supported tracking events specified by Tracking during ad rendering when the appropriate conditions are met. Some events need not be fired such as `Error`, which is specific to VAST parsing only. Events corresponding to operations unsupported during ad rendering also need not be fired, such as `Rewind`, `Mute`, or `AcceptInvitation` (which is specific to ads with interactive elements).

As an example, if `ad` contains the Ad Structure for a video ad that the client application has just begun rendering, the `Impression` beacons for that ad could be fired with a single call:

```
adIface.fireTrackingEvents(ad, {type: "Impression"})
```

While the ad playback progresses, assuming the variable `adProgressTime` holds a value representing the number of seconds since the ad began rendering, the quartile beacons can be sent via:

```
adIface.fireTrackingEvents(ad, {time: adProgressTime})
```

If the ad were paused by the user, then the client app would fire the `Pause` beacons:

```
adIface.fireTrackingEvents(ad, {type: "Pause"})
```

Requirements for Server Side Ad Insertion

1. Frequency Capping and Targeting Requirements

For apps that serve ads via SSAI, the outbound ad call to the ad server is not made from the client. The app will have to assume the onus of passing the RIDA from the client to the server side component of the SSAI vendor or the ad server.

> Please work with your SSAI vendor on how to pass the RIDA to the SSAI server side component. In some cases, the app may need to pass the RIDA as part of an ad call, in other cases there may be a web service that the app needs to call.

2. Nielsen Digital Ad Ratings Requirements

For server side ad inserted applications, call fireTrackingEvents() in RAF and ensure that the Nielsen beacons are passed to RAF via that API. It is valid to pass all impression beacons to RAF via this API. For non-Nielsen beacons, RAF will be just a pass-through.

> All these data points are sent directly to Nielsen via RAF. RAF does not keep or save any of these data elements on the device or any cloud storage.

3. Uniform Ad Experience Requirements

To enhance user engagement and consistency of ads served on the Roku platform, RAF supports rendering of both video and interactive ads (from certain vendors such as Brightline and Innovid) in server-stitched streams. This allows Roku to display standard UI components and behavior (ad counter, ad timer, trickplay support, etc.) for server-stitched streams.

> Any interactive ad types that are not enabled in RAF when server-stitched would need to be rendered by the application. See Custom Ad Rendering above.

Implementation Details

There are two API methods required for these use cases. First, the application is responsible for requesting and parsing the ad response, and structuring the ads into an array of pods, according to the required Ad Structure. This ad metadata may, in some cases, come from a third party SDK provided by your stitching platform.

- **For server stitched ads, the 'time' member of the 'tracking' data (in Ad Structure) for each ad is required**. The value of this data member should correspond to the absolute time in the entire stream, not just relative to the current ad.
    - Example: For a 30-second ad that starts at 15:00 in the stitched stream, the 'Impression' beacons for that ad should be set to 900 seconds and the 'Midpoint' beacons for that ad should be 915 seconds (i.e., @15:15). The 'time' member should still be omitted for beacons that do not depend on time (such as 'Pause' or 'AcceptInvitation').
- The meaning of postroll stitched ads is slightly different than for client-inserted ads, since the ads are part of the stream. **Client code can still set the 'renderSequence' for the pod to 'postroll', but all time values should still refer to the absolute position within the stitched stream.**

Scheduling and rendering is then initialized by first calling the `stitchedAdsInit()` method with the ad structure constructed by the client:

```
adIface.stitchedAdsInit(myAdPodArray)
```

Playback of the stitched stream is then started via an roVideoPlayer object (or optionally, a wrapped interface that matches the specification described in `stitchedAdHandledEvent()`). In the event loop for the video player, the app should then first check if an ad renderer handled the event, as well as checking for exit condition. If no ad renderer handled the event, control falls through to the application's regular event-handling logic:

Server-Side Ad Insertion Example

```
playContent = true
while playContent
  msg = Wait(0, videoPlayer.GetMessagePort())
  currentAd = adIface.stitchedAdHandledEvent(msg, videoPlayer)

  if currentAd <> Invalid and currentAd.evtHandled
    ' ad handled event, take no further action
    if currentAd.adExited
      ' user exited, return to content selection
      playContent = false
    end if
  else
    ' if no current ad or ad did not handle event, fall through to default event
handling here
    ' ... Your application's usual event-handling code here ...
  end if
end while
```

- If currentAd = invalid, then no current ad is being rendered, and the app can handle the event normally.
- If currentAd <> invalid and currentAd.evtHandled = true, then the ad renderer handled the event, and no further action should be taken on that event.
- If currentAd.adExited = true, then the user exited the ad renderer and the app should exit playback and return to content selection.
- Only if no current ad is being rendered or currentAd.evtHandled = false should the app handle the event in any way. Keep in mind that ad rendering can create new roImageCanvas objects with their own navigation, or roVideoPlayer objects with their own internal state and position. These will in general have nothing at all to do with any such object created by the content playback app, yet they will share the same message port so that the application event loop can forward all events to the ad renderer first.

Alternatively, an roAssociativeArray can wrap and mimic the interface of the roVideoPlayer parameter of stitchedAdHandledEvent(). See method description for the minimum required key-value pairs. This wrapped interface is useful if there are other actions to be taken on player control methods (such as analytics fired when the stream is paused, etc.)

Testing your RAF Implementation

To test your RAF implementation, you do not need to pass any URL argument to `setAdUrl()`. Use `setAdUrl()` as you would for the revenue split agreement and either omit the URL argument or the `setAdUrl()` call entirely. This allows you to check that ads are served correctly to users of the channel, but no revenue will actually be generated.

API Reference

Construction

Roku_Ads()

Description:

Returns the Roku ad parser/renderer object.

This is the main entry point for instantiating the ad interface. The object returned has global scope, since it is meant to represent interaction with external resources (the ad server and any tracking services) that have persistence and state independent of the ad rendering within a client application. Manages ad server requests, parses ad structure, schedules and renders ads, and triggers tracking beacons.

Control

General

fireTrackingEvents(adStructure as Object, ctx as Object) as Boolean

Description:

Triggers event tracking, including parameter substitution for Nielsen DAR, when library client code handles the ad rendering. Can be used in scenarios where the RAF ad renderer is not used (e.g., custom ad rendering or server-stitched ads).

Arguments:

- adStructure – Can refer to a pod (array) of ads or a single ad. Must at least contain a `Tracking` array member (see Ad Structure example), and may optionally contain an 'adServer' member string.
- ctx – structure to capture context-specific trigger conditions. 'type' key-value pair used to trigger events of a specific type. 'time' key-value pair used to trigger time-dependent events at or prior to this time.

Returns:

true if all beacons of the requested type are successfully fired, otherwise false.

Available since:

version 1.2

Client Ad Insertion

getAds(msg as string) as Object

Description:

Gets set of ads to be rendered now. When called with no parameters, this function returns the full list of all ad pods parsed from the ad server response. When called with the msg parameter, this function can be used as an event listener in the client application's main video playback loop to check whether midroll or postroll ads should be shown or not.

Arguments:

- msg – Optional, depending on use case. Typically, this would be a message returned from a WaitMessage() call on the message port of the roVideoScreen or roVideoPlayer object during content playback. This allows determination of which ads are scheduled for rendering based on playback position, user action, or other conditions.

Returns:

available ad pod(s) scheduled for rendering or invalid, if none are available

showAds(ads as Object, ctx as Object, view as Object) as Boolean

Description:

Render any ads scheduled for display. When called with an array of ad pods (e.g., using the value returned from the initial call to getAds()), this is interpreted to mean that any preroll ad pod present should be rendered. Client applications should always check the return value and, if false, should exit content playback and return to the content selection screen. Typically, this occurs when the user presses the "Back" button during ad playback.

Arguments:

Argument	Type	Required	Description
ads	array of ad pods	required	Ads to be rendered. Can represent either a single pod of ads or an array of ad pods.
ctx	associative array	optional	An associative array that allows client code to provide new offset and total to ad counter to support use cases involving interleaving RAF rendering with custom rendering within a single pod of ads. When used, it should be in the form of: `{ start: Integer, total: Integer }` Ex. `{ start: 1, total: 4 }` would display as: **Ad 1 of 4** in the top left corner during ad playback.
view	renderable node	optional [1]	Parameter representing a renderable node that the ad UI can be parented to.

The `view` parameter, when present, allows SceneGraph rendering of ads into an app that uses SceneGraph for content rendering.

> Please note the dimensions of the `view` object will be used to position RAF's UI elements, so it must be properly sized. Having dimensions larger than the current video playback resolution can place RAF UI elements such as the progress bar off screen.

- For server-stitched use case, this should be the `Video` node of the content player.
- For non-stitched use cases, it can be any renderable node in the scene whose lifetime is guaranteed during the duration of ad rendering. Render any ads scheduled for display.

[1] Added in version 2.0. This argument is required for all SceneGraph applications.

Example:

See RAF4RSG sample for a sample demonstrating the use of RAF in a Video node.

Returns:

`true` if ad pod was rendered to completion, `false` if user exited before render completion.

Server-Stitched Ads

stitchedAdHandledEvent(msg as Object, player as Object) as roAssociativeArray

Description:

Determines if a stitched ad is being rendered, lets the ad renderer attempt to handle the event, and returns metadata about the ad and the event handled state. This method is only intended for use in rendering server-stitched ads. The advertising framework must first be initialized using *stitchedAdsInit()* before calling this method.

Arguments:

- msg – returned object from a Wait() call on the message port used by the stitched video player. May be consumed by the ad renderer to measure playback state or provide user interactivity with stitched ad.
- player – player interface to allow ad renderer to control stitched video stream. If invalid or not specified, only beacons will be fired, and no interaction will be allowed or additional UI rendered during ad display. This parameter can be simply the roVideoPlayer instance used to play the stitched stream. It can also be an roAssociativeArray that contains methods congruent to the ifVideoPlayer interface, in case there is additional client code that should be executed when an ad renderer controls the stream (analytics, etc.)

The player parameter, if passed as an *roAssociativeArray* in an app where video is played with *roVideoPlayer* (non-RSG), must contain the following methods:

```
{ ' Returns message port for player
  GetMessagePort : Function() as Object,
  ' Pauses a stitched video stream
  Pause : Function() as Boolean,
  ' Resumes a paused stitched stream
  Resume : Function() as Boolean,
  ' Seeks to absolute position (in ms) within stream
  Seek : Function(offsetMs as Integer) as Boolean,
  ' Plays stitched video stream
  Play : Function() as Boolean,
  ' Stops stitched video stream
  Stop : Function() as Boolean
}
```

For RSG apps using a *Video* node for stitched ad playback, the *player* parameter should be an *roAssociativeArray* of the following form::

```
{
  sgNode : video, ' the video node which will render the stitched stream
  port : port ' the message port on which (at least) the "position" and "state" fields
of the above video node are observed
}
```

Returns:

- *Invalid* if no stitched ad is currently being rendered
- an *roAssociativeArray* that represents the current ad context and state

The return value, when an ad is being rendered, is of the form:

```
{
  adIndex : Integer, 'Index of current ad within pod
  adPodIndex : Integer, 'Index of current pod
  evtHandled : Boolean, 'True if event was handled by ad renderer
  adExited : Boolean, 'True if user exited ad rendering
  adCompleted : Boolean, 'True if ad has completed rendering
}
```

If the return value indicates that there is a stitched ad being rendered and that the event was handled by the renderer, the client application must take no action on that event. If the ad was exited, the client app should stop playback and return to the content selection screen.

Available since:

version 1.6

Configuration

General

setAdUrl(url as String)

Description:

Sets ad URL to use for a new getAds() request.

> You can only receive payment for ads shown in your application when the Roku Ad Framework is properly configured with a valid URL assigned by your ad service or by Roku. Please contact advertising@roku.com to discuss monetization options and obtain an ad URL if you wish to use Roku to fill ad inventory in your application. Using the default URL is useful only for development and testing purposes and you will not receive payment for ad impressions from the default URL.

Arguments:

- url – URL to set as the current ad service request (or omit this parameter to use the default Roku ad service)

getAdUrl() as String

Description:

Convenience function to get the currently-configured ad URL, or the default Roku ad server URL if none has been configured.

Returns:

current ad URL

setContentGenre(genres as String, kidsContent as Boolean)

Description:

Allows potential ad targeting by specifying a set of genre tags to associate with the content or the ad request. Can clear genre tags by passing an empty string or omitting the parameter. The semantics and implementation of targeting based on genre values are dependent on the configured ad server, but for a list of currently-supported tags supported by the Roku ad server, refer to Roku Genre Tags.

Arguments:

- genres – Comma-delimited string or array of genre tag strings.
- kidsContent - Optional boolean value to indicate if content is targeted towards children (`true`) or not (`false`).

setContentId(id as String)

Description:

Allows potential ad targeting by specifying an identifier for the content video. Passing an empty string or omitting the parameter will clear the content ID.

Arguments:

- id – String value representing content to allow potential ad targeting

setContentLength(length as Integer)

Description:

Configures content length to extend ad targeting properties for Nielsen DAR. Also used for determining VMAP relative ad break times. If called with no parameter, will clear any prior content length set.

Arguments:

- length – Integer value representing total length of content (in seconds)

Available since:

version 1.1

setAdPrefs(useRokuAdsAsFallback as Boolean, maxRequests as Integer)

Description:

Configures general ad request preferences. The default is for Roku to backfill ads if this method is not called or `useRokuAdsAsFallback` is not set to `false`.

Arguments:

- useRokuAdsAsFallback – Indicates whether the default Roku backfill ad service URL should be used in case the client-configured URL fails to return any renderable ads.
- maxRequests – Number of retries allowed if the ad service fails to return any renderable ads. The maximum retries allowed is 2.

setAdConstraints(maxHeight as Integer, maxWidth as Integer, maxBitrate as Integer, supportedMimeTypes as Object)

Description:

Configure media constraints to filter renderable video ads. By default, the MIME types are configured for “video/mp4”, “video/mp4-h264”, “video/x-mp4”, “application/x-mpegurl”, and “application/json”. Any additional known types can be mapped to their stream format by setting this parameter before calling getAds().

Arguments:

- maxHeight – Maximum vertical dimension of renderable ad (in pixels)
- maxWidth – Maximum horizontal dimension of renderable ad (in pixels)
- maxBitrate – Maximum allowable bitrate for renderable ad streams (in Kbps)
- supportedMimeTypes – Associative array with entries of the form {“mimeType” : “stream- Format”}

setAdBreaks(contentLength as Integer, adBreakTimes as Integer)

Description:

Configures content playback parameters, which can be used for scheduling relative-positioned ad breaks in VMAP ad service responses. If you know that your application uses VMAP ad URLs and they are configured to use “nn%” timeOffset values, then you must specify the contentLength prior to calling getAds(). If VMAP is configured to use “#mm” timeOffset values, you must first specify a set of ad break times. Calling with empty parameters will reset these to invalid values. The content length can also be set independently via setContentLength() if ad break times are not required.

Arguments:

- contentLength – Total length of video content (in seconds)
- adBreakTimes – Array of suggested offsets into content playback to insert ad breaks (in seconds)

setAdExit(enabled as Boolean)

Description:

The default behavior is to enable exiting ad render (e.g., via “Back” button) to return to content selection screen in the application. Some use cases may require disabling this behavior if the user should not be allowed to skip ads when there is no applicable content selection mechanism.

Arguments:

- enabled – Configures ad exit behavior during rendering.

Available since:

version 1.1

importAds(adPodArray as Object)

Description:

Resets the internal ad pod cache to allow client code to import a set of ads from unsupported ad service response formats or when aggregating ads from multiple ad services. The application is responsible for ensuring that the ad pods in the array contain all the required data members.

Arguments:

- adPodArray – Array of ad pods structured in accordance with the required Ad Structure.

Available since:

version 1.1

enableJITPods(enabled as Boolean)

Description:

For applications that use a VMAP or SmartXML ad response to structure multiple ad pods, including midrolls, the JIT (or “Just In Time”) feature can be used to avoid pre-fetching all ad metadata before the content playback begins. When enabled, ad call redirects for midrolls are deferred until a certain time before the ad pod is rendered. This mechanism relies on the host app’s continuous use of the BrightScript getAds API method with the content video position event to determine when to resolve the deferred ads.

Note: JIT is used as a global setting; if the app has mixed content streams, where some content should not use JIT (such as server-stitched ads), then the host app is responsible for disabling this functionality before any ad calls are made for such streams.

Arguments:

- Enabled – Boolean value to configure “Just In Time” fetching of midroll ads. By default, JIT is disabled and must be explicitly enabled via this API.

Available since:

version 2.4

setTrackingCallback(callback as Function, obj as Object)

Description:

Allows library client to set a callback function to be called when ad tracking events are fired or checked. Callback functions must have the signature:

```
Sub CallbackFunc(obj = Invalid as Dynamic, eventType = Invalid as Dynamic, ctx =
Invalid as Dynamic)
```

The `obj` parameter is an opaque object always passed through to the callback. The `eventType`, if set, is a string specifying a tracking event that is fired. Event names correspond to Tracking. The `ctx` is an optional associative array that encapsulates metadata associated with VAST-specified macros or ad render progress. Each member of the `ctx` array should separately be considered optional (i.e., client code should check for valid values before operating on these data members). Generally, if `ctx.eventType` is not set, then `ctx.time` should be set and indicates ad render progress:

```
{errType: String,  errCode: String,  errMsg : String,  time : Int | Float (playback
position, in s),  url : String (rendered asset URI),  ad : Associative Array
representing ad structure for current ad,  adIndex: Int (logical index of current ad
within ad pod) }
```

Arguments:

- callback – function matching the required function signature
- obj – an opaque object supplied by client to be passed to callback

Available since:

- version 1.1
- version 1.4: new ad, adIndex context members

setDebugOutput(enabled as Boolean)

Description:

Allows library client to configure extended debug output. Disabled by default.

Arguments:

- enabled – Boolean value to enable/disable extended debug logging

Available since:

version 1.2

getLibVersion() as String

Description:

Allows client applications to get the Roku Advertising Framework library version.

Returns:

version string of the form "`<major>.<minor>`"

Nielsen DAR

enableNielsenDAR(enabled as Boolean)

Description:

Applications using Nielsen DAR must explicitly enable the framework to operate on the custom impression tag parameters.

Arguments:

- enabled – Boolean value to enable/disable custom Nielsen DAR tags.

Available since:

version 1.1

> Please contact advertising@roku.com for more information on how to use audience measurement features.

setNielsenGenre(genre as String)

Description:

Allows for ad campaign measurement using Nielsen DAR tags by specifying a primary genre for the content being played, according to the Nielsen genres defined in Nielsen DAR Genre Tags. Examples: “CS” for an episode of “Seinfeld”, “N” for a “60 Minutes” episode.

Arguments:

- genre – String identifying primary content genre for Nielsen DAR tags.

Available since:

version 1.1

setNielsenAppId(id as String)

Description:

Allows for ad campaign measurement using Nielsen DAR tags. The value of this application ID is uniquely assigned to your application by Nielsen and *must* be configured before rendering any ads containing Nielsen beacons.

Arguments:

- id – String identifying Nielsen-assigned application ID

Available since:

version 1.2

setNielsenProgramId(id as String)

Description:

Allows for ad campaign measurement using Nielsen DAR tags. The value of this program ID should consistently reflect the program (movie title or series name) and should not uniquely identify episodic content. Examples: “The Price is Right”, “Beverly Hills 90210”, “Police Academy 7: Mission to Moscow”.

Arguments:

- id – String identifying content program for Nielsen DAR tags.

Available since:

version 1.2

Nielsen DCR

getNielsenContentData() as String

Description:

Returns an encrypted Nielsen RIDA parameter string for apps wishing to use the Nielsen SDK for DCR measurements.

Returns:

An encrypted Nielsen RIDA parameter string

Available since:

version 1.9

General Audience Measurement

enableAdMeasurements(enabled)

Description:

Applications using audience measurement features must explicitly enable the framework to operate on the custom impression tag parameters, in conjunction with the setContentGenre(), setContentId(), and setContentLength() APIs to provide more valuable measurement data.

Arguments:

- enabled – Boolean value to enable/disable audience identifiers in measurement tags.

Available since:

version 2.1

> Please contact advertising@roku.com for more information on how to use audience measurement features.

Server-Stitched Ads

stitchedAdsInit(adPodArray as roArray)

Description:

Imports ad metadata to be used for server-stitched ad rendering and resets the internal state before handling events. The application is responsible for ensuring that the ad pods in the array contain all the required data members. In particular, for server-stitched ads, all time-dependent tracking beacons (Impression and quartile beacons) must have a valid time data member set, with a value relative to the entire stitched stream. For example, if a 30-second ad starts at 10:00 within the stitched stream, its Impression beacons should have track.time = 600.0 and its Midpoint beacons should have track.time = 615.0, and so on. This method is used in conjunction with `stitchedAdHandledEvent()` to implement ad rendering within server-stitched video streams.

Arguments:

- adPodArray – Array of ad pods structured in accordance with the required Ad Structure.

Available since:

version 1.6

Buffer Screen Customization

setAdBufferScreenContent(contentMetaData as Object)

Description:

Allows the client application to set metadata for content used to populate the default ad buffer screen. contentMetaData conforms to the format defined in Content Meta-Data and can contain any or all of the following:

```
{ HDBackgroundImageUrl : String (URL for HD background image),
  SDBackgroundImageUrl : String (URL for SD background image),
  HDPosterUrl : String (URL for HD video poster),
  SDPosterUrl : String (URL for SD video poster),
  Title : String (Content title),
  Description : String (Content description)
}
```

Arguments:

- contentMetaData – roAssociativeArray which contains metadata representing information to be displayed in the default ad buffer screen.

Available since:

version 1.6

enableAdBufferMessaging(enableMsg as Boolean, enableProgressBar as Boolean)

Description:

Allows the client application to enable/disable specific features on the default ad buffer screen.

Arguments:

- enableMsg – Boolean value to enable/disable ad messaging text on default ad buffer screen. The default value is `true`.
- enableProgressBar – Boolean value to enable/disable ad buffering progress bar on default ad buffer screen. The default value is `true`.

Available since:

version 1.6

setAdBufferScreenLayer(zOrder as Integer, contentMetaData as Object)

Description:

Allows the client application to set individual layer metadata for custom ad buffer UI. contentMetaData conforms to the format defined in Content Meta-Data. Allowed values of both the zOrder and contentMetaData parameters are those allowed by roImageCanvas.

Arguments:

- zOrder – Integer layer index for display of this set of metadata
- contentMetaData – roAssociativeArray representing metadata for this UI layer

Available since:

version 1.6

clearAdBufferScreenLayers()

Description:

Allows the client application to clear all metadata in all layers previously set for the custom buffer screen.

Available since:

version 1.6

setAdBufferRenderCallback(callback as Function, obj as Object, timeout as Integer)

Description:

Allows the client application set a callback function and timeout value for ad buffering events, to provide opportunity for analytics methods or animation of elements on custom buffer screen. The `callback` parameter is a function that must have the following signature:

```
Function(obj as Dynamic, eventType as String, ctx as Dynamic) as Void
```

The first parameter to this callback is the obj parameter, defined above. The eventType parameter can take the following values:

- `BufferingStart`
- `BufferingEnd`
- `ReBufferingStart`
- `ReBufferingEnd`
- `Progress`
- `Timeout`

The ctx parameter is an roAssociativeArray that can contain the following:

```
{ ' array of content metadata set via setAdBufferScreenLayer, or Invalid
  canvasLayers : roArray of roAssociativeArrays,
  ' progress percentage [0-100]. Optional, only for "Progress" event type
  progress : Integer
  ' ad metadata for currently buffering ad
  ad : roAssociativeArray,
  ' index of current ad within pod
  adIndex : Integer
}
```

Arguments:

- callback – callback function to receive ad buffer events. The default value is Invalid.
- obj – opaque object passed to callback function. The default value is Invalid.
- timeout – Integer number of milliseconds to wait on buffer events before timing out. The default value is 0 (no timeout).

Available since:

version 1.6

URL Parameter Macros

The video ad library allows parameter values to be substituted in ad request and tracking URLs. This allows for dynamic configuration of values that are either not directly exposed to the client application or are unnecessary for it to initialize and maintain. These values are typically used for ad targeting, interaction tracking, and development purposes, or to optimize the ad experience for the user's device.

URL Parameter	Description
ROKU_ADS_TRACKING_ID	RIDA (Roku ID for Advertising) value used for device identification
ROKU_ADS_LIMIT_TRACKING	Set to true or false, depending on whether user has limited ad tracking

ROKU_ADS_APP_ID	Identifies the client application making the ad request
ROKU_ADS_APP_VERSION	Used to obtain the application version string
ROKU_ADS_LIB_VERSION	Used to obtain the RAF library version string
ROKU_ADS_CONTENT_ID	Identifies the content to allow for ad targeting
ROKU_ADS_CONTENT_GENRE	Identifies the content categorization to allow for ad targeting
ROKU_ADS_CONTENT_LENGTH	Improves ad targeting by providing length of content (in number of seconds)
ROKU_ADS_USER_AGENT	Device model and firmware version
ROKU_ADS_DEVICE_MODEL	Device model
ROKU_ADS_EXTERNAL_IP	External IP address of the device
ROKU_ADS_DISPLAY_WIDTH	Width of device display
ROKU_ADS_DISPLAY_HEIGHT	Height of device display
ROKU_ADS_TIMESTAMP	Current timestamp value (number of milliseconds elapsed since 00:00:00 1/1/1970 GMT)
ROKU_ADS_CACHE_BUSTER	Makes the URL unique to avoid retrieving cached ad server responses, or to ensure proper counting of unique event tracking beacons
ROKU_ADS_KIDS_CONTENT	Mark ad requests as "directed towards children." This macro is designed to help your channel comply with the Children's Online Privacy Protection Act (COPPA)

Example

To make an ad request that requires the application ID, user agent, and timestamp values, call setAdUrl() with those parameters set:

setAdUrl example

```
rokuAds = Roku_Ads()
url =
"http://my.ad.server.net/?my_first_param=MyFirstValue&my_app_id=ROKU_ADS_APP_ID&my_user_
agent=ROKU_ADS_USER_AGENT&my_timestamp=ROKU_ADS_TIMESTAMP&other_param=SomeOtherValue"
rokuAds.setAdUrl(url)
```

Ad Structure

For a client application that must implement its own ad rendering, it is necessary to understand how the ad structure is represented in the BrightScript object returned from getAds(). The following is a description of the ad structure. Ad pods passed to showAds() must conform to this structure.

> Square brackets '[ ]' indicate BrightScript arrays, curly brackets '{ }' indicate associative arrays, and prefix '+' indicates a required data member.

Ad Structure

```
adPods : [{
         +viewed         : Boolean,
         +renderSequence : String ("preroll" | "midroll" | "postroll"),
         +duration       : Float (in s),
         +renderTime     : Float (in s),
          slots          : Int,
          backfilled     : Boolean,
         +tracking: [{
            +event: String,
            +url: String,
            +triggered: Boolean,
             valid: Boolean
          }],
         +ads : [{
                 +duration     : Float (in s),
                 +streamFormat : String,
                 +adServer     : String,
                  adId         : String,
                  adTitle      : String,
                  advertiser   : String,
                  creativeId   : String,
                  creativeAdId : String,
                  clickThrough : String (URL),
                 +streams : [{
                             +url      : String (URL),
                             +bitrate  : Int (in kbps),
                             +width    : Int,
                             +height   : Int,
                             +mimeType : String,
                              provider : String
                  }],
                 +tracking : [{
                              +event     : String,
                              +url       : String (URL),
                              +triggered : Boolean,
                               valid     : Boolean,
                               time      : Float (in s)
                  }],
                  companionAds: [{
                                 +url          : String (URL),
                                 +width        : Int,
                                 +height       : Int,
                                 +mimeType     : String,
                                  clickThrough : String (URL),
                                  provider     : String,
```

```
+tracking : [{
             +event     : String,
             +url       : String (URL),
             +triggered : Boolean,
              valid     : Boolean,
              time      : Float (in s)
 }]
```

```
                    }]
            }]
}]
```

The object returned from a new call to getAds() with no parameters is an array of adPods in this format.

Tracking

Tracking events are triggered automatically during ad rendering by showAds(). For client applications that perform their own ad rendering, the valid event types that must be handled are represented in the `tracking` array of the Ad Structure by:

Event name	Trigger condition
Impression	Start of ad render (e.g., first frame of a video ad displayed)
PodStart	Beginning of ad pod render
PodComplete	Completed rendering ad pod
FirstQuartile	25% of video ad rendered
Midpoint	50% of video ad rendered
ThirdQuartile	75% of video ad rendered
Complete	100% of video ad rendered
Error	Error during ad parsing or rendering (VAST 3.0)
Close	User exited out of ad rendering before completion
Skip	User skipped ad (if skippable)
Pause	User paused ad
Resume	User resumed ad
Rewind	User rewound ad
Mute	User muted ad
Unmute	User un-muted ad
AcceptInvitation	User launched another portion of an ad (for interactive ads)

Roku Genre Tags

Tagging content by genre via setContentGenre() is specific to the ad provider, and may not be uniformly implemented. For ads provided by the Roku ad service, there is currently a canonical set of genre tags that can be used to improve ad targeting:

- Action
- Adventure
- Animated

- Ballet
- Biography
- Children
- Comedy
- Comedy drama
- Crime drama
- Cuisine
- Dark comedy
- Docudrama
- Documentary
- Drama
- Entertainment
- Fantasy
- Historical drama
- Horror
- Martial arts
- Music
- Musical
- Musical comedy
- Mystery
- Performing arts
- Romance
- Romantic comedy
- Science fiction
- Special
- Suspense
- Talk
- Theater
- Thriller
- Travel
- War
- Western

Nielsen DAR Genre Tags

Tagging content by genre via setNielsenGenre() requires a single primary genre code for the selected content from the following set of values. Publishers should provide the most specific category applicable to the content for which ads are to be shown.

Description	Code
Adventure	A
Audience Participation	AP
Award Ceremonies & Pageants	AC
Children's Programming	CP
Comedy Variety	CV
Concert Music	CM
Conversation, Colloquies	CC
Daytime Drama	DD
Devotional	D

Documentary, General	DO
Documentary, News	DN
Evening Animation	EA
Feature Film	FF
General Drama	GD
General Variety	GV
Instructions, Advice	IA
Musical Drama	MD
News	N
Official Police	OP
Paid Political	P
Participation Variety	PV
Popular Music -Contemporary	PC
Popular Music -Standard	PS
Private Detective	PD
Quiz -Give Away	QG
Quiz -Panel	QP
Science Fiction	SF
Situation Comedy	CS
Sports Anthology	SA
Sports Commentary	SC
Sports News	SN
Sports Event	SE
Suspense/Mystery	SM
Western Drama	EW