Implementing Server-Side Ad Insertion Using Roku Adapters

Overview

The Roku Ad Framework (RAF) is responsible for a consistent ad experience across the Roku platform. For channels that implement ads using server-side ad insertion (SSAI), this can prove challenging due to some intricate steps of the process. Developers can, therefore, leverage the Roku-approved SSAI adapters to simplify this process (See the samples below).

Note: Before using the adapters, please refer to Requirements for Server Side Ad Insertion.

Roku Ad Framework Requirements

All channels that have video advertisements are required to meet Roku's certification requirements for RAF. Notably, the channel must always use client-side firing event (All SSAI providers support client-side firing events) through RAF.

The Roku adapters provide two options:

- Firing off all the metrics through the SSAI adapter and RAF
- The channel is responsible for firing the metrics through RAF with the adapter callbacks (See examples for using callback below)

It is encouraged that the developer uses the first option, but with either option, the metrics must be fired using RAF APIs.

The APIs used are different depending on the approach, but audience measurement must be dispatched on the client side. Roku recommends that the channel adopts the comScore VCE-inclusive audience measurement API.

RAFX SSAI Adapters

The **RAFX SSAI Adapters** provide interfaces to both SSAI manifest servers (stitchers) and RAF, including:

- Parsing of the masterURL response, and extraction of playURL, AdURL, and ad metadata
- Transforming SSAI ad metadata into RAF-usable ad metadata and configuring RAF for playback
- Observing stream events and timed metadata

- Matching the stream events and ad metadata and firing event pixels on time
- Pinging/Polling AdURL as required by the SSAI manifest server, parsing, and reconfiguring RAF

Server-Side Ad Insertion Playback

To playback an SSAI stream, the developer must follow these steps:

1. Initialize a playback Task
2. Make a request to the masterURL
3. Enable the client-side ad tracking and get the playURL, AdURL and/or ad metadata
4. Configure playback content and observe stream events
5. Start playing stream and fire event pixels on time as responding to observed events and ping/poll ad metadata

How to playback SSAI content using the adapters

Following are the instructions on how to playback server-side ads content using the adapters:

Note: For specific working instructions for a particular adapter, refer to the adapter samples below.

1) Loading the adapter

The following entry loads the adapter into the task:

At the beginning of the playback Task, instantiate the adapter with proper parameters and then initialize it. The valid values of the parameter **name** are uplynk, adobe, onceux, yospace, awsemt, and ggldai.

```
adapter = RAFX_SSAI({name:"uplynk"})  ' Supported: uplynk, adobeonceux, yospace, awsemt,
ggldai
adapter.init()
```

2) Make an initial request to the SSAI manifest server getting Ad metadata: Request Ad Metadata

```
request = {

    type: adapter.SreamType.VOD  ' Required, VOD or LIVE

    url:  "http://admanifest.ssai.com/api?assetid=abcdefg"' Ad metadata URL, provided by
SSAI

}

result = adapter.requestStream(request)
```

The value of the parameter **URL** depends on which SSAI manifest servers are to be integrated and which type of stream is used (The channel may query the initial request to SSAI manifest server by itself rather than using the adapter.requestStream() call). Valid values of the parameter type are VOD or LIVE. VOD is used when ad metadata is fetched before the playback starts; LIVE is used when ad metadata is provided as ping/poll/in-stream (such as X-MARKER) content playback.

3) Read Stream Info

The initial request to SSAI manifest servers returns content URL (like Adobe and Uplynk). The following entry gets the content URL:

```
streamInfo = adapter.getStreamInfo()

url = streamInfo["playURL"]
```

4) Enable Ads

Once the content playback URL is known, the adapter is ready to track ads. Pass the adapter **player** object and observe the position event on the video node.

The value of params.**player** is given to RAF internally as the second parameter of RAF.stitchedAdHandledEvent(). adapter.enableAds() parses ad metadata and/or configures additional settings such as observing timedMetadata2 of given video node. It then calls RAF.stitchedAdsInit() when valid ad metadata is found in the initial response from the SSAI manifest servers.

```
port = CreateObject("roMessagePort")
params = {player: {sgnode:m.top.video, port:port}}
adapter.enableAds(params)
m.top.video.observeField("position", port)
```

The channel can provide an optional parameter, params.**useStitched** = false. If the parameter is set to false, the channel is required to:

- Set the callback functions to proper AdEvents
- Fire pixels by using RAF.fireTrackingEvents()
- Run and manage Interactive ads

By default, params.**useStitched** is set to true. In this case:

- Setting callback functions is optional
- Pixels are fired internally by the adapter and RAF
- Interactive ads are ran and managed by the adapter and RAF

a) Optional: Enable Ads Without stitchedAdHandledEvent

```
params = {
   player: {sgnode:m.top.video, port:port},
   useStitched: false
         }

adapter.enableAds(params) ' adapter will not call RAF.stitchedAdHandledEvent() and RAF
will not play Interactive Ad
```

b) Optional: Set Callbacks

A callback parameter is an object with keys, event, and position.

When using PODS, the object has an additional key, adPods, and the value of the parameter is an array of the adPod metadata.

When using POD_START, the object has an additional key, adPod, and the value of the parameter is the current adPod metadata.

Supported callbacks are:

- PODS
- POD_START
- POD_END
- IMPRESSION
- FIRST_QUARTILE
- MIDPOINT
- THIRD_QUARTILE
- COMPLETE

Note: When params.useStiched = true or, not provided in the param (this is the default), setting callbacks is optional and for informational purposes only. The channel **MUST NOT** call RAF.fireTrackingEvents() in such a case.

When params.useStitched = false, it is required to set callbacks and the channel **MUST** call RAF.fireTrackingEvents().

Setting the callback functions to the Adapter:

```
' Set adapter callback functions

adapter.addEventListener(adapter.AdEvent.POD_START, rafxCallback)

adapter.addEventListener(adapter.AdEvent.POD_END, rafxCallback)

adapter.addEventListener(adapter.AdEvent.IMPRESSION, rafxCallback)

...

...

function rafxCallback(eventInfo as object) as void

    if adapter.AdEvent.POD_START = eventInfo.event

        m.top.adPlaying = true

    else if adapter.AdEvent.POD_COMPLETE = eventInfo.event

        m.top.adPlaying = false

    end if

    print "Callback at : ";eventInfo.position

end function
```

5) Enable Ad Measurements

When the channel is ready to start playback, configure RAF by enabling ad measurements:

It is recommended to use **enableAdMeasurements.**

```
adIface = Roku_Ads()

adIface.enableAdMeasurements(true) ' Required

adIface.setContentLength(...) ' Set app/content specific info

adIface.setNielsenProgramId(...) ' Set app/content specific info

adIface.setNielsenGenre(...) ' Set app/content specific info

adIface.setNielsenAppId(...) ' Set app/content specific info
```

6) Playback Loop

The developer can now start the playback and run the message loop:

```
video.control = "play"  ' start playback

while true

    msg = wait(1000, port)

    curAd = adapter.onMessage(msg)

    if invalid = curAd

        video.setFocus(true)  ' recommended

    end if

    '  exit while loop when condition met

    ...

end while
```

adapter.onMessage() calls RAF.stitchedAdHandledEvent() and returns the object as it is. It is thus recommended to evaluate the returned value and call setFocus() on the video node in case the interactive ad changes focus while playing.

When not using RAF.stitchedAdHandledEvent(), the app must fire AdEvent pixels in the callback functions.

Once the playback is completed, discard the adapter. Do **not** reference the adapter outside of the Task or re-use the same adapter instance.

Roku Ad Framework APIs

For channels playing SSAI streams, it is required to call RAF APIs using the guidelines below.

It is required for channels to call the following every time the content is played back in the Task:

- Enable ad measurement: **enableAdMeasurements**(true)
- Set content/app info: **setNielsenProgramId**(), **setNielsenGenre**(), **setNielsenAppId**(), **setContentLength**()

When useStitched is set to true, the adapter itself:

- Generates RAF ad metadata from SSAI specific format and calls **stitchedAdsInit**(), **stitchedAdHandledEvent**() to fire ad events and measurement pixels

When useStitched is set to false, the developer should make sure that the channel:

- Sets callback functions to the adapter
- Fires event pixels via: **fireTrackingEvents**() when called back

RAFX SSAI Adapter Samples

The following RAFX SSAI Adapter samples are built to provide client-side integration with SSAI, for both VOD and LIVE.

Note that **VOD** mode means ad metadata is fed from the SSAI server before content playback.

LIVE mode means ad metadata is given as a part of the content stream such as ID3 (HLS) and emsg (DASH), and/or periodic ping/poll request to SSAI server.

Before using the adapter samples, the developer must be familiar with the SSAI providers' documentation:

- Uplynk
- Adobe
- Brightcove
- Yospace
- Amazon

Note: The SSAI providers may have undocumented formats, parameters and API behaviors.

Uplynk Adapter

Uplynk Adapter	File

RAFX SSAI Adapter for Uplynk Preplay and Ping mode, showing ad rendering via stitchedAdsInit()/stitchedAdHandledEvent().
 	
The Uplynk Adapter provides the following services:
 	
• When Live-Ping
  • Request preplay and parse ads object
  • Ping and parse JSON, track with timestamps
  • Track ID3 tags and match ad objects
  • Configure RAF stitchedAdsInit() as ID3 tags
  • Track all ad events through stitchedAdHandledEvent()
  • Halt or append ads to current adPods as ID3 tags indicate
• When VOD-Preplay
  • Request preplay and parse ads object
  • Configure RAF stitchedAdsInit()
  • Track all ad events through stitchedAdHandledEvent()
	
	
	rsgupl.zip
	
	See UplynkTask.brs to find out how to use the adapter. Copy rafxssai.brs to your project and integrate it with the content playback Task.

Adobe Adapter

Adobe Adapter	File

RAFX SSAI Adapter for Adobe Manifest Server simple and x-marker mode, showing ad rendering via stitchedAdsInit()/stitchedAdHandledEvent().
 	
Adobe Adapter provides the following services:
 	
• When Live-x-marker
  • Observe ID3 tag: #EXT-X-MARKER
  • Parse ad metadata and configure RAF stitchedAdsInit()
  • Track ad events through stitchedAdHandledEvent()
• When VOD-simple
  • Request master URL and select stream
  • Request ad metadata with pttrackingposition=1, pttrackingmode=simple
  • Supported pttrackingversion=vmap and v2 JSON
  • Parse ad metadata and configure RAF stitchedAdsInit()
  • Track ad events through stitchedAdHandledEvent()
	
	
	rsgadb.zip
	
	See AdobeTask.brs to find out how to use the adapter. Copy rafxssai.brs to your project and integrate it with the content playback Task.

Brightcove/OnceUX Adapter

Brightcove/OnceUX Adapter	File

RAFX SSAI Adapter for OnceUX VOD mode, showing ad rendering via stitchedAdInit()/stitchedAdHandledEvent().

OnceUX Adapter provides the following services:

- When VOD
  - Request ad metadata and parse XML
  - Configure RAF stitchedAdsInit()
  - Track ad events through RAF stitchedAdHandledEvent()

When reading stream info, "playURL" field is not available because OnceUX provides a pair of video contentURL and metadata URL.

Read Stream Info:

```
...
streamInfo = adapter.getStreamInfo()
'  url = streamInfo["playURL"]        This field is NOT
available when OnceUX adapter.
...
```

Expand source

However, the returned value of getStreamInfo() includes a field called **tracking.** This returns a list of event info generated from XML element: <uo:contentImpressions><uo:Impression>. The app is responsible for sending those pixels when playback starts.

For example:

Sending Content Start Beacon:

```
...
m.top.video.control = "PLAY" ' Start video content
...
...
adIface = Roku_Ads()
for each evt in streamInfo.tracking
    if "Impression" = evt.event
        adIface.util.getNoResponseFromUrl(evt.url) ' send
beacon to OnceUX
    end if
end for
```

Expand source

rsgoux.zip

See OnceUXTask.brs to find how to use the adapter. Copy rafxssai.brs to your project and integrate it with the content playback Task.

Yospace Adapter

Yospace Adapter	File

RAFX SSAI Adapter for Yospace server, showing ad rendering via stitchedAdInit()/stitchedAdHandledEvent().
 	
Yospace Adapter provides the following services:
 	
• When VOD
  • Request masterURL, parse XML(DASH) or manifest(HLS), extract playbackURL and analyticsURL
  • Request ad metadata, parse XML and configure RAF stitchedAdsInit()
  • Track ad events through RAF stitchedAdHandledEvent()
• When LIVE
  • Request masterURL, parse XML(DASH) or manifest(HLS), extract playbackURL and analyticsURL
  • Observe timed metadata
  • As playback stream, ping Yospace server and parse XML
  • Match timed metadata YMID and ad metadata, configure RAF stitchedAdsInit() and stitchedAdHandledEvent()
	rsgyspc.zip
	
	See YospaceTask.brs to find how to use the adapter. Copy rafxssai.brs to your project and integrate it with the content playback Task.

AWS Adapter

AWS Elemental MediaTailor Adapter	File

RAFX SSAI Adapter for AWS Elemental MediaTailor(AWSEMT), showing ad rendering via stitchedAdInit()/stitchedAdHandledEvent() .

AWSEMT Adapter provides following services:

- Request masterURL, parse JSON, extract hls_url and tracking_url. For apps with known hls_url, use setStreamInfo() instead of requestStream() and getStreamInfo()

  Using setStreamInfo()

```
Expand source
if makingIntialRequest
    result = adapter.requestStream(...)
    streamInfo = adapter.getStreamInfo()
else
    streamInfo = {
        type: m.top.testConfig.type,
'Required
        tracking_url:
m.top.tracking_url, 'Required. App must
provide valid URL
        hls_url: m.top.hls_url
'Required. App must provide valid URL
    }
    adapter.setStreamInfo(streamInfo)
end if
```

- Poll ad metadata, parse JSON and configure RAF stitchedAdsInit()
- Track ad events through RAF stitchedAdHandledEvent()
- masterURL may require GET or POST. When POST request is required, fill request.body with {"adParams":{}}
  See AEMTTask.brs, function loadStream().

rsgemt.zip

See AEMTTask.brs to find how to use the adapter. Copy rafxssai.brs to your project and integrate it with the content playback Task.