ATSC Recommended Practice: Use of ATSC 3.0 Interactive Content

A/381:2026-03
11 March 2026

Revision History

Version	Date
A/381:2025 Recommended Practice approved	11 March 2026

Table of Contents

Index of Figures and Tables

ATSC Recommended Practice: Use of ATSC 3.0 Interactive Content

1. SCOPE

This document provides recommended practices for Broadcasters and Broadcaster Application (BA) developers using the interactive operating environment described in A/344 [2].

1.1 Introduction and Background

ATSC 3.0 is a suite of voluntary technical Standards and Recommended Practices for a digital terrestrial television broadcast system. ATSC 3.0 is fundamentally different from predecessor ATSC systems[1] and is therefore largely incompatible with them. Because of these differences and incompatibilities, an ATSC 3.0 service will not be recognizable to legacy Receivers or systems that are specific to ATSC 1.0. Consumers of ATSC 3.0 services are required to utilize new systems and processing for reception of the new signals. In general, this means that new Receivers are necessary to handle ATSC 3.0 broadcast and broadband signals and services.

Most of these new Receivers provide an interactive content environment comprised of a standard W3C User Agent with known characteristics, a WebSocket interface for obtaining information from the Receiver and controlling various Receiver functionality, and an HTTP interface for accessing files delivered over broadcast or broadband.

While one might expect that it would be possible to run an arbitrary web application (app) on a Receiver, several considerations require special attention when developing apps for the ATSC interactive content environment, such as:

- Lack of support for client-side Web framework (e.g., React)
- Different interfaces than those found on typical web client devices
- Limitations on available processing power
- Constraints on available storage or memory
- Other hardware-related constraints
- Temporary or permanent disconnection from the Internet (unconnected devices)

A BA developer can expect the following from a Receiver:

- An input/interface via remote control keypresses
- An application runtime environment as described in Section 4.1 of A/344 [2]
- That most client-side frameworks are not available (especially on unconnected devices)
- That the Receiver receives broadcast signals but may not be connected to the internet
- The Receiver provides minimum capabilities to support an HTML5 User Agent as described in section 5.2 of A/344 [2]
- HTML5/CSS support as described in CTA-5000 [7]

These expectations are discussed at length in this document. Considerations such as resource constraints and limited input devices are briefly discussed in this document, but developers are

[1] These systems include the original ATSC DTV system (retrospectively and informally referred to as "ATSC 1.0"), which is defined in ATSC A/53 [19] and other standards, and its backward-compatible updates referred to as "ATSC 2.0", as defined in ATSC A/107 [20] and other standards.

encouraged to perform their own evaluation of target devices to ensure their BAs perform as desired.

The A/344 [2] specification provides a set of APIs that BAs can use to interact with a Receiver. Developers of BAs may use this document to obtain guidance on how to best use the A/344 APIs to create smooth interactive experiences. Not all APIs are implemented by all Receivers (see section 4.2), which requires BA developers to implement contingency plans for the cases where APIs might not be available for their use on a specific Receiver.

The CTA provides Recommended Practice documents (e.g., CTA-CEB32.8 [10]) These documents include an informative list of assertions that are being tested. This information can be very useful for BA developers. In addition, CTA-5000 [7] provides guidance on writing media apps for use on consumer products.

Broadcasters can find advice on how to develop their protocol stack operation as well as data to be delivered to Receivers for use by Broadcaster Applications in section 8.

This document outlines several typical usage scenarios, including

- Closed captions, and which APIs to use
- Enabling the purchase of products and services via Receivers (T-commerce)
- Content replacement

For each use case or usage scenario, this document provides details on how to develop interactive Broadcaster Applications, and how Broadcasters can provide the data needed to maintain those BAs.

1.2 Organization

The document is organized as follows:

- Section 1 – Outlines the scope of this document and provides a general introduction to the interactive environment and examples of BA usage scenarios
- Section 2 – Lists references and applicable documents
- Section 3 – Provides definitions of terms, acronyms, and abbreviations for this document
- Section 4 – Overview of the general application operating environment in ATSC 3.0, including the application life cycle.
- Section 5 – Provides examples of how Receivers differ from application environments that web applications typically encounter. This includes devices that have no connection to the Internet (unconnected devices) and devices with streamlined interfaces.
- Section 6 – General guidance for BA developers, including recommendations on code hygiene and error handling.
- Section 7 – Recommendations on developing BAs for specific use cases and usage scenarios, include workflow and app lifecycle.
- Section 8 – Recommendations for Broadcaster data streams to support BAs.
- Section 9 – Describes approaches for using APIs that may not be supported by all Receivers in the field.
- Section 10 – Application testing recommendations.
- Annex A – Avoiding on-screen conflicts between BAs and closed captions, including examples and how to manage those screen updates.

2. REFERENCES

All referenced documents are subject to revision. Users of this Standard are cautioned that newer editions might or might not be compatible.

A summary of where information can be found, including other documents, is shown below.

- Guidance for broadcasters and BA developers: the present document
- Normative text for broadcasters: A/344 [2] and other ATSC standards
- Guidance for Receivers: A/344 [2]
- Recommendations for Receivers: CEB32.8 [10]

2.1 Informative References

The following documents contain information that may be helpful in applying this Standard.

[1] IEEE: "Use of the International Systems of Units (SI): The Modern Metric System," Doc. SI 10, Institute of Electrical and Electronics Engineers, New York, NY.

[2] ATSC: "ATSC Standard: ATSC 3.0 Interactive Content," Doc. A/344:2025-07, Advanced Television Systems Committee, Washington, DC, 17 July 2025.

[3] ISO/IEC: "Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," International Organization for Standardization, 15 May 2014.

[4] ATSC: "ATSC Standard: Signaling, Delivery, Synchronization, and Error Protection," Doc. A/331:2025-10, Advanced Television Systems Committee, Washington, DC, 16 October 2025.

[5] ATSC: "ATSC Standard: Service Announcement," Doc. A/332:2025-10, Advanced Television Systems Committee, Washington, DC, 27 October 2025.

[6] ATSC: "ATSC Standard: ATSC 3.0 Security and Service Protection," Doc. A/360:2025-07, Advanced Television Systems Committee, Washington, DC, 17 July 2025.

[7] CTA: "CTA Specification: Web Application Video Ecosystem – Web Media API Snapshot", Doc. CTA-5000-G, Consumer Technology Association, Arlington, VA, October 2024.

[8] ISO/IEC: "Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 1: MPEG media transport (MMT)," Doc. ISO/IEC 23008-1:2017(E), International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland.

[9] W3C: "Media Source Extensions," W3C Recommendation, World Wide Web Consortium, 17 November 2016.
https://www.w3.org/TR/media-source/

[10] CTA: "Recommended Practice for ATSC Television Sets, Application Runtime Environment (CTA-CEB32.8-D)," December 2025, https://www.cta.tech/standards/cta-ceb328-d/

[11] CTA: " Recommended Practice for ATSC 3.0 Television Sets, Overview (CTA-CEB32-D)," December 2025, https://www.cta.tech/standards/best-practices-for-atsc-30-tvs-audio/

[12] WHATWG: "HTML Living Standard," Section 9.3 "Web sockets," Web Hypertext Application Technology Working Group.
https://html.spec.whatwg.org/multipage/web-sockets.html

[13] HTML 4.01 Specification, Section 16: Frames
https://www.w3.org/TR/html401/present/frames.html

[14] ATSC: "ATSC Digital Television Standard, Parts 1 through 6 – Digital Television System," Doc. A/53, Advanced Television Systems Committee, Washington, DC, various dates.
[15] ATSC: "ATSC Standard: A/107 – ATSC 2.0 Standard," Doc. A/107:2015, Advanced Television Systems Committee, Washington, DC, 15 June 2015.
[16] ATSC: "ATSC Standard: Application Event Delivery," Doc. A/337:2025-10, Advanced Television Systems Committee, Washington, DC, 14 October 2025.

For WHATWG living standards, while it is recommended that implementations support the living standard, they must support the snapshot version of each WHATWG standard at the time of the earliest commit in 2020.

3. DEFINITION OF TERMS

With respect to definition of terms, abbreviations, and units, the practice of the Institute of Electrical and Electronics Engineers (IEEE) as outlined in the Institute's published standards [1] should be used. Where an abbreviation is not covered by IEEE practice or industry practice differs from IEEE practice, the abbreviation in question will be described in Section 3.3 of this document.

3.1 Compliance Notation

This section defines compliance terms for use by this document:

should – This word indicates that a certain course of action is preferred but not necessarily required.

should not – This phrase means a certain possibility or course of action is undesirable but not prohibited.

3.1.1 A/344-Specific Terms

Specific to A/344 [2] and to this Recommended Practice, the phrase "expected to" or the word "expected" are used to specify that the Receiver Reference Model and Broadcaster Application are expected to behave in a particular manner. Similarly, "not expected to" is used to specify that the Receiver or Broadcaster Application are not expected to behave in the described manner. Many of these requirements are described as recommendations in CTA CEB32.8 [10]. Note that neither A/344 nor CEB32.8 specify normative requirements for Receiver implementations. Furthermore, Broadcaster Application implementations are out-of-scope for ATSC standards, so conformance language is inappropriate in those cases.

3.2 Treatment of Syntactic Elements

This document contains symbolic references to syntactic elements used in the audio, video, and transport coding subsystems. These references are typographically distinguished by the use of a different font (e.g., restricted), may contain the underscore character (e.g., sequence_end_code) and may consist of character strings that are not English words (e.g., dynrng).

3.2.1 Reserved Elements

One or more reserved bits, symbols, fields, or ranges of values (i.e., elements) may be present in this document. These are used primarily to enable adding new values to a syntactical structure without altering its syntax or causing a problem with backwards compatibility, but they also can be used for other reasons.

The ATSC default value for reserved bits is '1'. There is no default value for other reserved elements. Use of reserved elements except as defined in ATSC Standards or by an industry standards-setting body is not permitted. See individual element semantics for mandatory settings and any additional use constraints. As currently reserved elements may be assigned values and meanings in future versions of this Standard, receiving devices built to this version are expected to ignore all values appearing in currently reserved elements to avoid possible future failure to function as intended.

3.3 Acronyms and Abbreviations

The following acronyms and abbreviations are used within this document.

AMP	Application Media Player
API	Application Programming Interface
BA	Broadcaster Application
BA-CC	Broadcaster Application - Closed Captions
CC	Closed Captions
CSS	Cascading Style Sheets
CTA	Consumer Technology Association
DASH	Dynamic Adaptive Streaming over HTTP [3]
DOM	Document Object Model
ESG	Electronic Service Guide [5]
HELD	HTML Entry pages Location Description [4]
HTML5	HyperText Markup Language, Fifth Version
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ID	Identifier
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
JSON-RPC	JSON Remote Procedure Call
MMT	MPEG Media Transport
MMTP	Multimedia Transport Protocol
MP	MMT Package [8]
MPD	Media Presentation Description [3]
MPEG	Moving Pictures Experts Group
MPU	Media Processing Unit
MSE	W3C Media Source Extensions [9]
NRT	Non-Real Time
OTA	Over The Air
RFC	Request For Comment
RMP	Receiver Media Player
ROUTE	Real-time Object Delivery over Unidirectional Transport
RP	Recommended Practice
RPC	Remote Procedure Call
STB	Set-Top Box
TV	Television
UI	User Interface

URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Universal Resource Name
UTC	Universal Time Coordinated
W3C	Worldwide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
WS	WebSocket [12]
XML	eXtensible Markup Language

3.4 Terms

The following terms are used within this document.

In the event of any discrepancy between the definitions that appear in this document and those that appear in a Standard, the definitions in the Standard are authoritative and take precedence. For terms defined in A/344, please refer to A/344 [2] for the authoritative definition.

Broadcaster Application – Broadcaster Application refers to the client-side functionality of the broader Web Application that provides the interactive service.

Receiver – An entity that implements the functions of the Reference Receiver Model.

Receiver Web Server – A conceptual component of a Receiver that provides a means for a User Agent to gain access to files that conceptually reside in the Application Context Cache.

Receiver WebSocket Server – Application that provides a means for a User Agent to access information about the Receiver and control features provided by the Receiver.

Reference Receiver Model – A conceptual Receiver device that can execute the APIs and behavior specified in A/344.

reserved – Set aside for future use by a Standard.

User Agent – Software provided by the Receiver that retrieves and renders Web content.

Web Application – A client/server program comprised of an HTML5 "Entry Page" and other resources referenced directly or indirectly by that document, all provided by a broadcaster in an ATSC 3.0 service accessed via the web using URLs.

4. OVERVIEW OF THE ATSC 3.0 APPLICATION OPERATING ENVIRONMENT

Broadcaster Applications (BAs) are applications designed to be executed in the HTML5 environment on a Receiver (e.g., a television). A BA can be delivered via the over-the-air broadcast (OTA) or via a Receiver's broadband connection (OTT). For details on application delivery, see Section 8.

ATSC 3.0 application runtime uses an HTML5 user agent in a device's native operating system, a separate "sandbox" for HTML5 commands to run. A/344 specifies it with an HTML5 protocol ([7]) that references W3C and Application Program Interface (API) calls intended for television usage scenarios where no keyboard or mouse hardware is available.

Broadcaster applications are signaled together with programs (HELD signaling), where the intent is for consumers to engage and interact with content.

While the runtime environment and signaling are defined, user experience specifications are not provided by ATSC A/344 nor any CTA guideline. ATSC A/344 describes an application lifecycle that includes expected behaviors of designed applications so that no harm occurs on devices and user experiences are enjoyable with no artifacts (e.g., no WebSocket 404 errors).

Webpage application designs do not easily port over to televisions as keyboard and mouse hardware are not available; the input device is usually a remote control with a few keys. While multi-frame HTML5 pages could be a preferred arrangement of pages in a frameset, navigation of such design can become complex with only remote keys available.

W3C describes framesets in its Frames in HTML documents [13]. BA developers are advised to be careful when using frameset designs when only remote control (UP / DOWN / LEFT / RIGHT / ENTER / BACK) key navigation is possible. Application workflow at a high level is described in Section 4.1. Applications are recommended to track history key presses for navigating through webpages.

Television device resources are limited, so minimum multi-threaded operations are recommended.

Two media players are available to a BA: Receiver Media Players (RMPs) and Application Media Players (AMPs). RMPs run directly on the Receiver hardware, while AMPs run within an application. Applications run in an HTML5 user agent, and user agents might not have access to all codecs in a native OS. It is recommended to use RMP APIs for rendering media rather than relying on the AMP.

Televisions are not likely to support simultaneous stream rendering, either in RMPs or AMPs, which means multi-channel video playback is not recommended.

4.1 High-level BA Workflow in ATSC 3.0

Basic broadcaster application launch procedure is depicted in A/344 [2], Figure 6.3. Application lifecycle is described in Annex A of A/344. Throughout this document, "Application" and "BA" refer to a Broadcaster Application per A/344. "Broadcast path" refers to OTA delivery within an ATSC 3.0 service. "Broadband" refers to Receiver-initiated IP connectivity external to the broadcast path.

Since applications are part of television programing, users need the option to engage with the interactive experience. This starts with an 'icon' pop-up on the display for a few seconds that users can select if desired. That 'icon' is a small graphic image described as BA Appear. It can be a simple graphic that a user can interact with via a remote control.

Only one BA runs at a time.

When users select the interactive environment, the application requests the device's information with DeviceInfo API and notes that information. An example is key value pairs of the remote. Applications are a guest of the native devices OS and cannot expect to have continuous access to remote control keys. Therefore, user navigation through the application requires key requests after each timeout period provided in Request Keys Timeout API. Figure 4.1 shows the high-level workflow of BAs in ATSC 3.0.

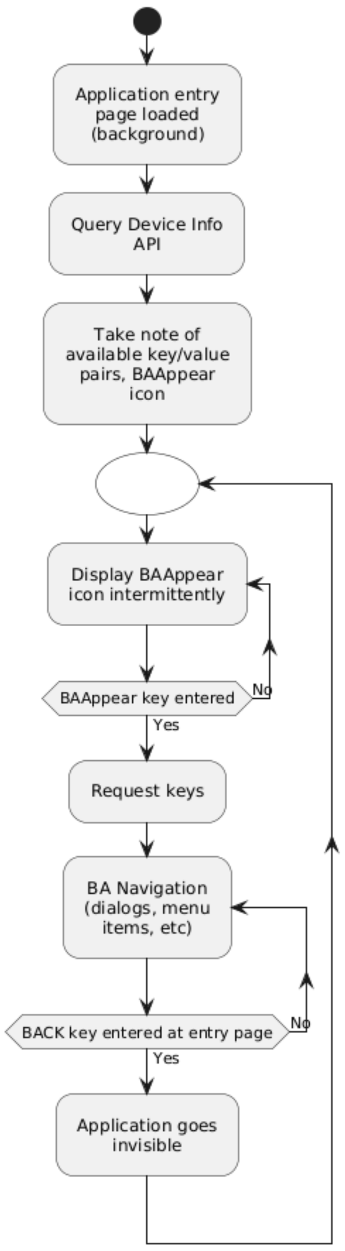

Application entry page loaded (background)
Query Device Info API
Take note of available key/value pairs, BAAppear icon
Display BAAppear icon intermittently
BAAppear key entered
No
Yes
Request keys
BA Navigation (dialogs, menu items, etc)
BACK key entered at entry page
No
Yes
Application goes invisible

Figure 4.1 High-level BA workflow.

4.2 API Support

Not all APIs defined in A/344 are necessarily supported across all devices. Refer to CTA-CEB32.8 [10] for a list of APIs that are required for Receivers to use the NextGenTV logo. Some Receivers, such as older Receivers, will not support all the APIs required for logo certification. See Section 9 for more details.

5. BA DEVELOPER CONSIDERATIONS

BA developers should take into account Receivers such as those described in this section. Developers of BAs need to be prepared to support Receivers that:

- Are temporarily or permanently disconnected from the Internet
- Have different user input interfaces than other platforms (e.g., no keyboard or mouse)
- Have limitations on available processing power
- Offer less available storage or memory than other platforms
- Are impacted by browser versions
- Lack of optimizations

Do not assume that a Receiver will provide more than:

- Remote control keypress as input/interface
- Minimum capabilities required to support the user agent
- The HTML5 and CSS features as described in CTA-5000 [7]

The BA will need to query the Receiver to find information on:

- What Receiver resources are available
- Whether the Receiver is connected to the internet
- Which Receiver interfaces are available to the BA
- The Receiver make and model
- How to interrogate the browser's navigator object at system level

5.1 Unconnected Receivers

Not all Receivers are connected to the Internet. For unconnected Receivers, all data for the BA, including the BA itself, is delivered via the broadcast (see Section 9 for details on broadcast stream support).

5.2 Interfaces

As noted elsewhere in this document, the main user input interface for the Receiver is a remote control.

Most remote controls do not include a point-and-click interface. Pointing devices are described in W3C [9], but this sort of input is not widely supported in Receivers. The Receivers that do are manufacturer-specific, so there are no A/344 APIs for such an interface with a coordinate system.

The BA should only request the input keys that are applicable to the current user interaction as described in A/344 [2] Section 9.11, Keys APIs. BA developers are encouraged to consider which keys will be needed by the BA, and how the BA will behave if the requested keys are not accepted by the Receiver.

BAs can also interface with events sent via the ATSC broadcast, such as those described in Section 9.6 of A/344. The A/344 standard also describes notifications that a BA can use to monitor changes to the Receiver, such as receipt of a new version of any Low-Level Signaling (LLS) table or Service-Level Signaling (SLS) fragment (Signaling Data Change Notification).

When a BA is leveraging the RMP, the BA can coordinate with RMP playback using the APIs defined in Section 9.13 of A/344.

5.3 Storage and Memory

Receivers typically have on the order of 1 GB of RAM.

BA developers are advised to keep in mind that other services will also require memory and that not all memory will be available for the BA. In addition, BA developers are advised to research limitations on app size for their target platforms. Common Receiver platforms limit apps to a few dozen MB for the entire TV app, and BAs must fit within the HTML5 sandbox that is within that TV app.

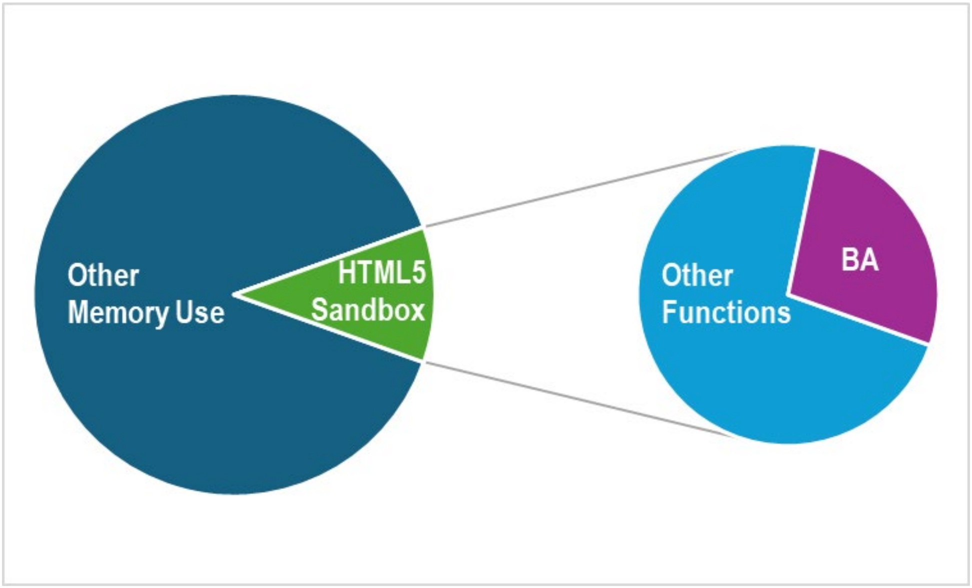


Figure 5.1 Example of resource availability.

5.4 Processing Power

Many Receivers have processing power that is similar to a 900 MHz quad-core 64-bit CPU.

As for storage and memory, the availability of the processing power allocated to the BA will depend on how many other services are running and BA developers are advised to keep in mind that the BA will not be able to use all the resources of the Receiver.

5.5 Key Label Mapping for Input Consistency

Inconsistencies have been identified between the key values provided in the deviceInfo.deviceInput object and those delivered to Broadcaster Applications through the DOM via KeyboardEvent, particularly in the key values (string labels), which may vary even when the keyCode values remain consistent. These differences may occur within the same Receiver or across different Receivers, where each Receiver may assign different string labels to the same key.

This issue is especially noticeable for commonly used keys such as "Back" and "Enter," where returned labels may include "Escape", "VK_BACK", "Backspace", "Enter", "OK", "Select", or may be undefined. Such inconsistencies can lead to unpredictable application behavior and reduce portability across Receivers.

To support consistent key handling, Broadcaster Applications are expected to use the keyCode values from deviceInfo.deviceInput as the main reference, in spite of the fact that W3C has deprecated keyCode. Applications can implement an internal mapping table that connects known keyCode values to the intended key label or logical action (e.g., 8 → "Back", 13 → "Enter"). When a key event is received through the DOM, the Broadcaster Application can use the event's keyCode to look up the corresponding action from this mapping, rather than relying on the KeyboardEvent.key string, which may vary by Receiver.

This approach is expected to provide consistent and predictable input behavior across a wide range of Receivers and supports reliable application performance in multi-device environments.

6. GENERAL GUIDANCE FOR BA DEVELOPERS

6.1 Lifecycle Behavior

Applications should implement the A/344 lifecycle described in A/344 [2] Annex A (launch, running, background, termination) and provide a clear path for the user to return to linear content. An "idle" or "data unavailable" presentation should be provided to maintain usability during transient asset loss or event gaps though this should be avoided whenever possible.

6.2 Code Hygiene

Authors should avoid dynamic code injection patterns (e.g., eval) and evaluate third-party libraries for offline behavior and compatibility with A/344 [2] user-agent expectations, as described in other sections of this document.

6.3 Error Handling and Recovery

Applications should provide user-visible, actionable error states and implement retry/backoff strategies that avoid tight reload loops or blank screens. Where appropriate, cached last-known-good content may be displayed with clear status messaging.

6.4 Graceful Degradation

Features requiring broadband (e.g., personalization, remote analytics) should be optional enhancements. Applications should remain useful and navigable using broadcast-only delivery.

7. DEVELOPING BAs FOR SPECIFIC USE CASES AND USAGE SCENARIOS

7.1 Content Replacement

Content replacement is a use case where a BA can instruct the Receiver to replace content with alternative content. In MMT, this is also known as asset replacement. The process of content or asset replacement differs depending on whether the broadcast is via MMT or ROUTE/DASH.

7.1.1 Asset Replacement in MMT

The MMT asset replacement process uses lightweight JSON-based messaging, which has minimal processing overhead for the Receiver. However, actual latency depends on factors such as Receiver processing speed and system performance. The Receiver's primary workload such as decoding and playback is unaffected as the API operations occur asynchronously with playback. Replacement via the AssetLink Resolved API only occurs if the BA explicitly provides content, otherwise the Receiver continues playing the original asset without interruption.

This process uses three APIs, defined in A/344 [2]:

1) AssetLink Resolution Notification (A/344 §9.17.1), which notifies the BA about assets with URLs before playback, ensuring early preparation for potential replacement, and provides critical details like `assetId` and `assetLink`, aligning perfectly with the MP Table.
2) RMP Media Asset Change Notification (A/344 §9.13.8), which informs the BA when the asset is detected for playback and includes `presentationTime`, enabling precise synchronization. It allows the BA to track asset transitions in real time.

3) AssetLink Resolved (A/344 §9.17.2), which enables the BA to replace an asset by providing a `replacementURL` or `replacementText` for playback integration with minimal perceptible impact.

7.1.1.1 Detailed Workflow

This workflow assumes that the BA is running and that it has successfully subscribed to the AssetLink Resolution notification. If the BA cannot successfully subscribe to the AssetLink Resolution notification, asset replacement cannot be performed.

1) Receiver Sends AssetLink Resolution Notification (A/344 §9.17.1)
    - When the Receiver parses the MP Table, it identifies an asset with:
        - A location (`assetLink`) that requires resolution.
        - An asset ID (`assetId`) uniquely identifying the asset.
    - The Receiver sends the AssetLink Resolution Notification to the BA after detecting an asset with a URL. The notification includes:
        - `assetId`: The unique ID of the asset (important for tracking across APIs).
    - This early notification allows the BA to prepare for content replacement and track the asset using its `assetId`.
2) Receiver Sends RMP Media Asset Change Notification (A/344 §9.13.8)
    - When the Receiver detects the asset in the live stream for playback through the MMPU box, it sends the RMP Media Asset Change Notification to the BA.
    - This notification includes:
        - `assetId`: The same ID provided in the AssetLink Resolution Notification, ensuring the BA knows it is the same asset, linking this notification to the notification from step 1.
        - `presentationTime`: The exact time when the asset will start playing. This allows the BA to know when the asset will play, so it can synchronize playback of downloaded or replacement content.
        - `duration`: How long the asset will be played.
3) BA Calls AssetLink Resolved API (A/344 §9.17.2)
    - After receiving the two previous notifications, the BA decides whether to replace the asset.
    - If replacement is required, the BA calls the AssetLink Resolved API with:
        - `assetId`: Same ID to identify the target asset.
        - `replacementURL`: A URL pointing to the replacement content (e.g., ad or updated video).
        - OR `replacementText`: Inline content encoded in Base64.
    - The Receiver (RMP) uses the assetId and replaces the correct asset in the live stream at the specified `presentationTime`, ensuring smooth playback.

Notes on Properties:

- The assetId is the same in all three APIs (A/344 §9.17.1, 9.13.8, and 9.17.2). This ensures the BA can track and manage the asset throughout its lifecycle, from initial notification (A/344 §9.17.1) to detection in live stream (A/344 §9.13.8) to replacement (A/344 §9.17.2).
- The presentationTime in RMP Media Asset Change Notification (A/344 §9.13.8) is essential for the BA to know exactly when the asset will start playing. This timing

information allows the BA to synchronize downloaded or replacement content with the live stream and minimize perceptible impact.

7.1.1.2 Subsequent Assets

The playback sequence follows the MP Table's schedule. Even when a replacement occurs, the MP Table remains the source for determining the next asset's timing and location.

If an asset is replaced by content downloaded from a URL (via the AssetLink Resolved API), the Receiver monitors the duration of the replacement content. Once the playback of the replacement content is complete, the Receiver refers back to the MP Table to determine the next asset. The MP table contains the list of all assets in the MMT package, including their Asset IDs, URL (`assetLink`), and Presentation times (`presentationTime`). The Receiver then transitions to the next scheduled asset in the playback order, using the `presentationTime` of the next asset as specified in the MP Table.

7.1.1.3 Error Cases

To ensure the BA has sufficient time to provide a replacement, the Receiver is expected to send the AssetLink Resolution Notification (A/344 §9.17.1) soon after detecting an asset with a URL in the MP Table. The RMP Media Asset Change Notification (A/344 §9.3.18) then provides the exact presentationTime, allowing the BA to determine when playback begins. At that point, the BA can replace the asset via the AssetLink Resolved API (A/344 §9.17.2). If the BA responds late (after the asset has already started playback), or if the replacement fails to download in time, the Receiver defaults to the original asset as specified in the MP Table. Late responses are ignored once playback begins, ensuring no disruptions.

If the replacement fails, playback does not stall as the Receiver defaults to the original asset or transitions to the next scheduled asset. If decoding fails mid-replacement, the Receiver may either attempt to resume the original asset if playback timing allows or move to the next scheduled asset to avoid disruptions.

7.1.1.4 Updates to the MP Table

The Receiver continuously monitors the MP Table in the live stream for updates. When a new MP Table update is detected, the Receiver re-parses it to identify any new assets or timing changes. If an update affects a currently playing asset, the Receiver ignores the change to avoid disrupting playback. If an update affects an upcoming asset, the Receiver provides the updates in subsequent AssetLink Resolution (A/344 §9.17.1) and RMP Media Asset Change (A/344 §9.13.8) notifications to the BA.

7.1.2 Content Replacement for DASH/ROUTE

In this use case, a BA can instruct the Receiver to replace an asset with alternative content (also known as content replacement). This replacement process uses lightweight JSON-based messaging, which has minimal processing overhead for the Receiver. However, actual latency depends on factors such as Receiver processing speed and system performance. The Receiver's primary workload such as decoding and playback is unaffected as the API operations occur asynchronously with playback. Replacement via the Set RMP URL API only occurs if the BA explicitly provides content, otherwise the Receiver continues playing the original MPD without interruption.

Event-based content replacement for MPEG DASH services requires the following components:

1) Source content with program boundary markers

2) Encoder capable of generating DASH Events
3) Receiver implementing the relevant A/344 APIs
4) Broadcaster Application with replacement logic
5) (Optional) Backend for determining content

The specific A/344 APIs utilized are the following:

1) Set RMP URL, startRmp operation, (A/344 §9.7.3), which enables the BA to request that the Receiver begin playback of a given MPD immediately or at a specific future time, relative to the current media presentation timeline.
2) Event Stream Subscribe (A/344 §9.6.1), which allows the BA to subscribe to Events present in the DASH stream, both in-band and MPD, when on a DASH-based service.
3) Event Stream Event (A/344 §9.6.3), which notifies the BA of the presence of an Event in the stream, including the media presentation time of the event, whenever an Event appears in the stream.
4) Integrated Subscribe API (A/344 §9.3.1.1), which is used to subscribe to RMP Media Time Change events.
5) RMP Media Time Change Notification (A/344 §9.13.5), which informs the BA that the current presentation time has changed.
6) Query Device Info (A/344 §9.12), which allows the BA to retrieve key information about the device, particularly the make/model, advertisingId, and supported APIs.
7) Cache Request DASH (A/344 §9.4.2), which allows the BA to request that the Receiver cache a specific broadband MPD and its associated content onto the local device storage.
8) Query Cache Usage (A/344 §9.5), which allows the BA to determine the total quota and current usage of the cache available to the BA.

7.1.2.1 Transmission Workflow

In order to perform frame-accurate content replacement, the broadcaster must transmit content with the proper markers in order to indicate the program boundaries with sufficient warning for the Broadcaster Application to perform the replacement workflow before the replacement point. This is typically done through the use of SCTE markers, which are inserted into the source feed, generally by the master control automation system. These markers will often contain information not only on the timing of the boundary, but also regarding the contents of the program, allowing the Broadcaster Application to determine when to replace content, and what that content will be replaced with. These markers are passed, normally in-band, to the encoder generating the DASH stream for transmission via ATSC 3.0.

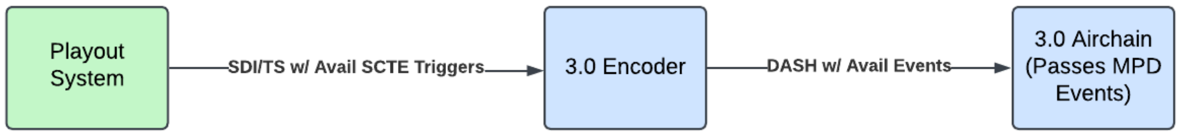

Playout System
SDI/TS w/ Avail SCTE Triggers
3.0 Encoder
DASH w/ Avail Events
3.0 Airchain (Passes MPD Events)

Figure 7.1 Transmission workflow.

The encoder then performs two key tasks: splicing the segments at the boundaries and inserting the Event data into the DASH stream. The former is done to ensure an I-frame is present at the start of each boundary, marked by a new Period, which allows the Receiver to have continuous, uninterrupted playback. The latter carries the actual body of the data, particularly the id, duration, and time of the Event, which are the bare minimum to perform a successful replacement. Typically, a broadcaster will use the `urn:scte:scte35:2014:xml+bin` or `urn:scte:scte35:2013:xml` schemeIdUris, which provide for the transmission of the full SCTE marker data within the body of the Event. Once this is generated, it is placed either in an EventStream element within the new Period in the MPD, or inband in the emsg box, with a corresponding InBandEventStream element. This is then passed through the remainder of the 3.0 airchain and out to the Receiver.

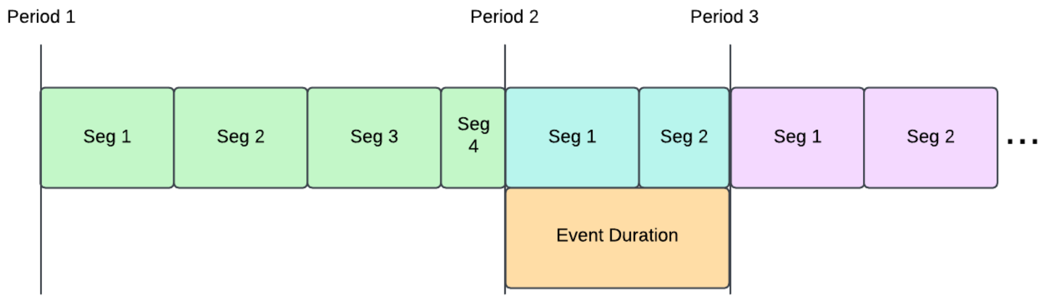


Figure 7.2 Segments.

It is key to utilize Early Available Periods (EAP) in this workflow, which allow the broadcaster to indicate that an Event is upcoming at a specific time, even if there is not yet content associated with that time. An EAP is a Period that is listed in the MPD, using either Period@start on the EAP, or Period@duration on the prior Period, to indicate the time that this Period will occur. This Period can contain elements such as EventStreams, which are then able to reference presentation times that occur following the beginning of the Period. This allows the usage of preroll triggers to be translated into DASH, giving the BA sufficient warning of upcoming content replacement opportunities.

7.1.2.2 Broadcaster Application Workflow

The BA forms the nexus of the system and is responsible for the actual logic of the content replacement. This ignores cases where APIs or features within those APIs are not supported. Figure 7.3 shows how the BA interacts with the Receiver in a sample system representative of common backend infrastructure. The BA to Receiver workflow assumes that the rest of the infrastructure portrayed is present. The DASH stream contains Events with boundary-aligned times, with sufficient preroll to allow for the insertion workflow. The BA communicates with an ad decision server, which in turn has knowledge of the both the business rules around content replacement and the identifying information in the SCTE-based DASH Events to determine what the contents of a given Event are. Lastly, the ad content server is accessible from the Receiver, typically via the internet.

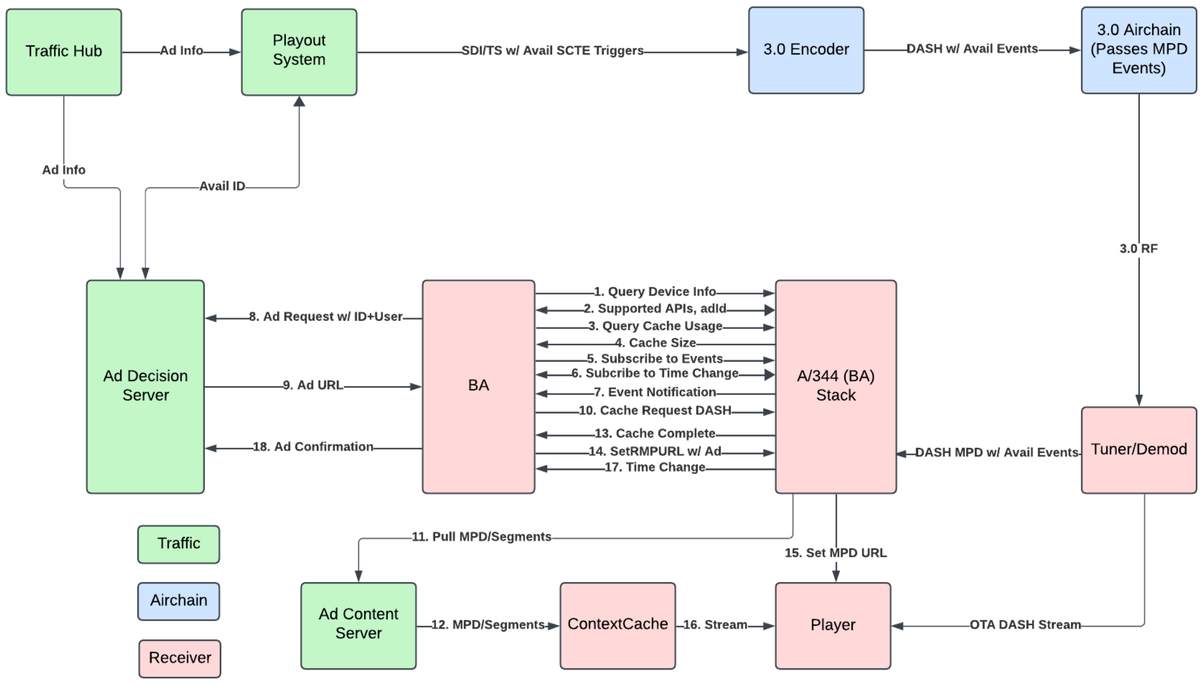


Figure 7.3 DASH Content Replacement workflow.

1) Call: Query Device Info
    a) Note Make/Model for device-specific behaviors.
    b) Ensure required APIs are in deviceSupportedWebSocketAPIs if provided.
    c) Note advertisingId if implementing this for ad insertion.
2) Call: Query Cache Usage
    a) Note available cache size.
3) Call: Event Stream Subscribe
    a) Set schemeIdUri to the URI used on the transmit side in the EventStream element, e.g. urn:scte:scte35:2014:xml+bin.
    b) Set value to the relevant value if looking for a specific Event, rather than general content replacement.
    c) Leave this on the default 'onReceive,' it allows for prior warning of replacement opportunities via EAPs.
4) Call: Integrated Subscribe API
    a) msgType = rmpMediaTimeChange
5) Notification: Event
    a) This contains the id, duration, and eventTime, along with any relevant data.
    b) id is used to determine the programming location, which is used by the decision server to choose whether to replace, and if so, what to replace.
    c) duration is used by the decision server to generate a playlist of appropriate length to fill the full time of the event.
    d) eventTime is used by the BA to determine when to set the rmpSyncTime for Set RMP URL.

   e) Data will often be application-specific data, such as SCTE marker info, which can help the decision server refine the content selection.
6) Internal: Decision
   a) The BA should utilize the Event information, in combination with local logic or a remote decision server, the advertisingId from device info, and any other relevant information, to determine what to replace the content with, if anything.
   b) The chosen content must be in the form of a DASH MPD as discussed in 7.1.2.4.
7) Call: Cache Request DASH
   a) This will generally be used with a DASH MPD, with the exception of the cases noted in 7.1.2.3.
   b) The BA should call this API repeatedly until the Receiver replies with cached = "true."
   c) The targetURL can be used to organize the cache in such a way that the BA can ensure no conflicts between multiple cache request files.
8) Call: Set RMP URL
   a) use the startRmp operation.
   b) set rmpurl to the local location of the cached DASH MPD.
   c) rmpSyncTime is a time relative to the media presentation time, which allows the BA to utilize the eventTime from the Event notification to ensure frame accuracy.
   d) default endOperation to resumeService, which will allow the Receiver to flow back to the prior content with minimal perceptible impact.
9) Notification: RMP Media Time Change
   a) The data in this can be parsed to determine if the content successfully was replaced, as the currentTime will, in almost all cases, change when the Receiver switches to the requested MPD.
10) Loop 7-9 until replacement is complete if multiple Set RMP URLs are required.
   a) The time change notification tells the BA that the content has begun playing, which means that in a case such as ad insertion where multiple MPDs are being used to cover a single Event, the next content replacement sequence can begin.

7.1.2.3 BA Developer Considerations

The Receiver is expected to receive the feed and process it in order to support the APIs listed in A/344. However, there are variances that must be considered on a Receiver by Receiver basis, beyond simply supporting these APIs. The most important is the behavior when Set RMP URL is utilized. Many Receivers do not begin buffering the MPD's content until the designated time is reached, which can create glitches in the viewer experience. Thus, it is important to vary the behavior of the BA based on the model identified using the Query Device Info API.

For those devices that do not buffer beforehand, use the Cache Request DASH API. As the replacement content is decided upon, the Receiver would send the request to the Receiver to download the media for that content's MPD. This request must be repeated until cached = "true" is received, at which point the BA can substitute the remote MPD URL in the Set RMP URL call with the local location of the cached content. This ensures the Receiver does not need to wait for segments to buffer before beginning playback of the new MPD, as the segments are then already on the device.

However, there is an additional consideration that must be made, as Receivers often have limited cache capacity to work with. This can be determined through the Query Cache Usage API.

For particularly long sections, the storage required may exceed what is available in the cache. In this scenario, such as a two-minute ad break consisting of four 30 second spots, the BA can send a cache request for the first spot, then call Set RMP URL when that cache is completed. Once the first spot has begun playing, indicated by the Media Time Change Notification, the BA can then cache the second spot and call Set RMP URL for it. This ensures the minimum cache space is used at any given time, while also remaining compliant to the requirement that only one Set RMP URL be queued at a time.

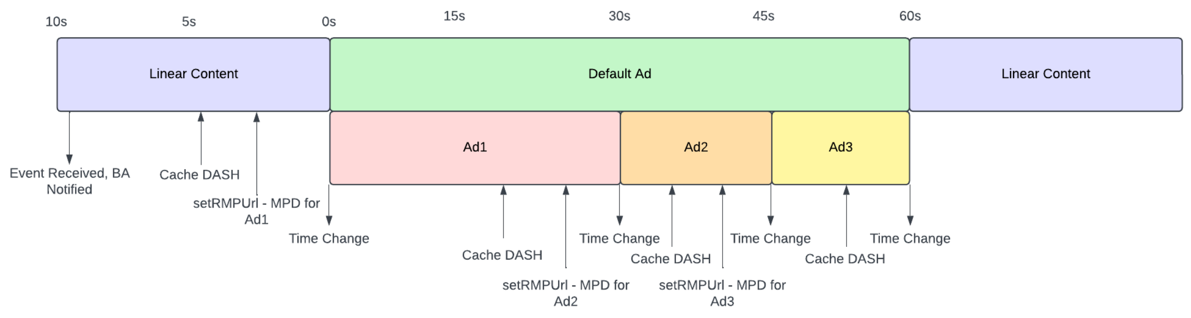


Figure 7.4 Timing.

In the event a set of content cannot be broken up into discrete MPDs, it can instead be split within a single MPD. The Cache Request DASH API allows the BA to request a specific Period be cached, rather than the entire MPD, which in this case can be used to load in the initial section of an MPD, while the other Period(s) point to the broadband location. This ensures the content switch still can take advantage of the caching to ensure no glitch in playback, then reverts to the RMP-managed buffer model after that first Period.

7.1.2.4 Backend Considerations

As noted in 7.1.2.2, the backend provides a certain set of conditions for the insertion workflow to properly execute:

- Only DASH URLs can be utilized
- The backend system must guarantee DASH sourcing by either:
  - Filtering the ad exchange such that it only supplies DASH-based spots, or
  - Converting non-DASH sources such as HLS or MP4 to DASH on the fly (e.g., by the ad exchange or the decision server). In a CMAF system, this is as simple as generating the new MPD using the existing CMAF segment; but if not using CMAF, the segments will need to be remixed to DASH, maintaining the existing codecs to improve the processing speed.
- Replacement Opportunity IDs: These are typically generated by the broadcaster, potentially in combination with EIDR or other similar IDs to identify the specific content. These are unique enough to be able to conclusively determine where along the linear transmission the opportunity is occurring, so that it can be associated with the relevant business rules. It is important to give care to the timeshifted viewership use case given the proliferation of DVRs, as repeated identifiers could cause errors in content replacement if a viewer playing a recorded stream encounters a replacement opportunity. Note that in

addition to the DASH event ID, additional identifiers are usable within the provided Base64-encoded SCTE message, for example:

7.2 Avoiding BA – Closed Caption Conflicts

BAs render UI components over video, such as interactive menus, graphics, or prompts intended for viewer interaction. Closed captions (CC), which are rendered separately by the Receiver, can appear in the same visual area of the screen as the BA content. Because the BA and the Receiver operate independently in terms of layout, overlapping screen regions might occur.

Examples of Overlapping Scenarios include:

- Captions rendered in regions already occupied by UI components from the BA, such as menus or prompts. This can reduce the readability of both the captions and the BA content.
- BA UI components appearing in regions commonly used for caption placement, which can block portions of the caption text.

To support coordination between BAs and Receiver-rendered captions, A/344 defines several APIs that can be used to query screen layout, adjust the video region, and declare UI component areas. These APIs enable the BA to build layouts that are aware of and responsive to caption rendering behavior. Annex A provides examples of BA layout designs that use these APIs to avoid display conflicts.

The following three APIs are relevant to managing screen layout in the presence of captions:

- **Query Display Components API** allows the BA to learn where captions are currently rendered and, if desired, move its visual output to avoid conflict.
- **Video Scaling and Positioning API** enables the BA to request changes to the video rendering region. If supported by the Receiver, this can be used to create space for captions or UI components.
- **Graphics Display Regions API** allows the BA to declare which parts of the screen are being used by UI components, to inform caption placement. If supported, the Receiver can use this information to avoid placing captions in the same regions, helping reduce visual conflicts.

These APIs can be used independently or together, depending on the Receiver's capabilities and the BA's layout requirements. Of these three APIs, only the Video Scaling and Positioning API is currently "Recommended" by CTA's logo testing program (see API list in [10]).

7.2.1 Query Display Components API

The Query Display Components API allows a BA to obtain information about the Receiver's current display configuration, specifically with respect to the caption display region and video window scaling capabilities. This information can help the BA adapt its visual layout to reduce the likelihood of conflicts with closed captions.

This API provides the BA with:

- A list of caption regions currently active on the screen
- Support status for video window scaling
- Support status for caption scaling relative to video scaling

While the API returns coordinate and scaling information, it is recommended that developers note the following considerations.

7.2.1.1 Caption Regions Are Not Frame-Accurate Snapshots

The activeCaptionRegions reported by the Receiver are not real-time representations of the caption text on screen. Instead, they represent the maximum expected area where captions might appear based on current font size, layout settings, and text capacity. These regions can be larger than the area used by individual caption lines at any moment in time.

7.2.1.2 Regions Can Be Empty

If no captions are enabled or currently active, the activeCaptionRegions array can be empty. It is not recommended that BAs assume captions are always present.

7.2.1.3 Multiple Regions Can Be Returned

The API supports multiple caption regions, which might be present in cases such as dual-language captioning or side-by-side captioning. It is recommended that BAs iterate over all entries in the array and avoid placing UI components in any of the reported areas.

7.2.1.4 Units Are in Pixels

All coordinates and dimensions in activeCaptionRegions are reported in absolute screen pixel values, relative to the top-left corner of the screen.

7.2.1.5 Re-query on Caption Changes

The caption display configuration can change during playback if the user adjusts caption settings (e.g., font size or language). It is recommended that BAs monitor the Caption Display Preferences Change Notification (A/344 [2] Section 9.3.6) and re-issue the query when changes occur.

7.2.1.6 Optional Support

Not all Receivers are required to support this API. If the API is not supported, the BA can avoid placing critical UI components near the bottom of the screen, as captions are frequently displayed in that area by default on many Receivers.

7.2.1.7 Caption Scaling Support Flag

The captionScalingSupported flag informs whether the caption renderer will scale proportionally when the video window is resized. This helps determine whether video scaling (via the Video Scaling and Positioning API) will affect the position of captions as well.

7.2.2 Video Scaling and Positioning API

The optional Video Scaling and Positioning API enables a BA to request the Receiver Media Player (RMP) to scale and reposition the video window within the display area. If supported by the Receiver, this functionality can be useful for creating space for captions or UI components so that they do not overlap with the video.

This API allows the BA to:

- Scale the video display to a smaller size (from 100% down to a Receiver-supported minimum).
- Position the scaled video window at a defined location within the screen.

This API can help the BA establish a caption-aware layout where the video is shifted or resized to reduce visual conflicts.

It is recommended that developers note the following implementation guidance.

7.2.2.1 Scaling and Positioning Behavior

Each request applies the new scaling and position to a reset full-screen state not to the result of a previous transformation. The Receiver applies the scale first, then positions the video.

7.2.2.2 Receiver Capabilities Vary

Not all Receivers support video scaling. It is recommended that BAs first check the videoWindowScalingSupported flag using the Query Display Components API before issuing any scaling request. If unsupported, the Receiver will return an error.

7.2.2.3 Minimum Size Limits

The Receiver might reject requests using a scaleFactor smaller than supported. In such cases, the error response might include the minimumScaleFactor field to indicate the smallest acceptable value. BAs can retry with an adjusted value or choose an alternative layout.

7.2.2.4 Impact on Captions

Requesting that the program video be resized does not guarantee that captions will move. Whether captions reposition or scale with the video depends on the Receiver's support for captionScalingSupported. If unsupported, captions will remain in their default location even if the video is moved.

7.2.3 Graphics Display Regions API

The optional Graphics Display Regions API allows a BA to inform the Receiver of which areas of the screen are occupied by its graphical layout. If supported, the Receiver can use this information to avoid placing captions in the same regions, helping reduce visual conflicts. This declaration is particularly useful when caption repositioning is supported by the platform.

It is recommended that developers note the following implementation guidance.

7.2.3.1 Grid Model

The screen is divided into 25 numbered regions arranged in a 5×5 grid below. Numbering starts at the top-left (Region 1) and proceeds left-to-right, top-to-bottom:

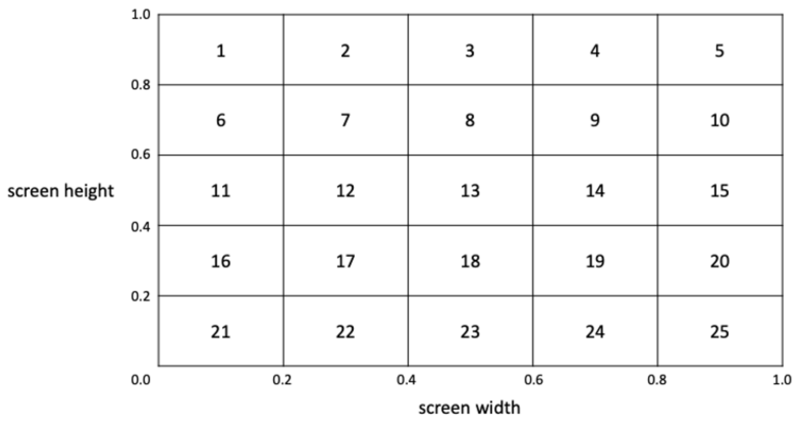


Figure 7.5 Screen Grid Numbering.

Each bit position corresponds to a region number, with Region 1 assigned to bit 0 (least significant bit), Region 2 to bit 1, and so on, up to Region 25 assigned to bit 24:

The seven (7) most significant bits (bits 25–31) are always set to zero (0).

To set a specific bit corresponding to a region number, developers use the bitwise left shift operator <<, which shifts the binary value 1 to the left by the specified number of positions. For example,

- for Region 1: 1 << 0 → binary 000...0001 → decimal 1 and

- for Region 2: 1 << 1 → binary 000...0010 → decimal 2 and
- for Region 3: 1 << 2 → binary 000...0100 → decimal 4,
- finally for Region 25: 1 << 24 → binary with a 1 in the 25th position → decimal 16777216

To set multiple regions, combine the results using the bitwise OR (|) operator. For example,

- for Regions 6, 7, and 8: (1 << 5) | (1 << 6) | (1 << 7) → 000…1110000 → decimal = 224
- (32 + 64 + 128)

7.2.3.2 When to Send Updates

The BA can call the Graphics Display Regions API at any point during its lifecycle. The Graphics Display Regions API can be invoked at any point during the BA's runtime. It is useful to send updates when the BA initially presents its layout, when graphical elements are added or removed (such as menus or overlays), or when caption settings change.

7.2.3.3 Coordinate Mapping Is Approximate

The Graphics Display Regions API does not support pixel-accurate coordinate declarations. Instead, it defines a fixed 5×5 grid across the screen, where each region represents approximately 20% of the screen width and height. The BA is expected to indicate which regions are partially or fully covered by its UI components. If any part of a UI element overlaps a region, that region is marked as occupied. The grid acts as a simple signaling method for layout coordination, not as a detailed spatial map.

7.2.3.4 Error-Free by Design

The API does not define any specific errors beyond general JSON-RPC failure handling. A successful call returns an empty object {}, regardless of whether the Receiver uses the declared data. If the API is unsupported, a JSON-RPC error is returned.

7.2.3.5 Optional Support

Not all Receivers are required to support this API. If the API is not supported, the BA can avoid placing critical UI components near the bottom of the screen, as captions are frequently displayed in that area by default on many Receivers.

8. BROADCASTER STREAMS TO SUPPORT BAs

This section provides guidance for the preparation, signaling, and broadcast delivery of BAs as specified in A/344 [2]. The objective is to promote robust Receiver behavior, predictable user experience, and operational efficiency when applications are delivered via the OTA broadcast path as part of an ATSC 3.0 service.

8.1 Authoring and Packaging Practices

8.1.1 Self-Containment

An application should include all assets required for initial rendering and basic interaction (HTML, CSS, JavaScript, fonts, images, configuration). Relative paths should be used so the package operates via the broadcast HTTP interface defined by A/344 without reliance on broadband at launch.

8.1.2 Initial Payload Minimization

Authors should minimize the "app shell" (entry document, essential styles, bootstrap scripts) to reduce time-to-first-interaction. Non-critical modules and heavy assets may be deferred to later carousel cycles. Authors should avoid runtime fetches that are not guaranteed to be available over broadcast.

8.1.3 Deterministic Versioning and Cache Behavior

Receiver caches and broadcast object caches can retain files longer than expected. Versioned paths avoid stale asset mixtures during updates.

Applications should employ versioned directory paths (e.g., /apps/news/v1.04/...) and, where appropriate, hashed filenames for long-lived assets. These are recommended to allow both Receivers and broadcast caches to distinguish new releases from cached content they have chosen. Reusing a stable URL for different content is discouraged to avoid inconsistent cache behavior across Receivers.

8.2 Signaling and Lifecycle Coordination

Note: Receivers vary in how they surface application state changes. Conservative signaling reduces visible disruptions.

8.2.1 Availability Windows

Application availability should be scheduled in alignment with program or service periods. Where availability changes mid-program, broadcasters should coordinate signaling in accordance with A/331 and, if applicable, communicate state transitions using A/337 events.

8.2.2 Signaling Stability

Broadcasters should avoid unnecessary signaling churn (e.g., frequent changes to application descriptors) during a program, as some Receivers may restart or reset application state on signaling updates. When updates are required, Broadcasters should be batch and time the updates to minimize user impact.

8.3 ROUTE/DASH Delivery Considerations

8.3.1 Conformance to Interoperability Points

Application delivery over ROUTE/DASH should conform to the applicable DASH profile and ATSC 3.0 interoperability constraints to minimize Receiver variability.

8.3.2 Object Sizing and Cadence.

Broadcasters should prefer multiple small-to-moderate objects over few large ones to improve recovery after packet loss and to reduce perceived carousel latency. Critical startup objects (entry HTML, essential CSS/JS) should be repeated more frequently than non-critical assets.

8.3.3 Prioritization

The ROUTE object schedule should prioritize first-paint assets and essential configuration. Large images and optional modules may be assigned longer refresh periods. Where practical, the MPD and object priorities should reflect the desired user-perceived loading order. A/344 [2] only states that the Entry Package required to be cached so this package must contain all the code and content to start a basic Broadcaster Application. See the overall discussion in A/344 Section 6.2 and Section 6.2.3 describing the Entry Package.

Note: Shorter object cycles reduce the time to acquire missing critical files and limit the user's exposure to partial UI states.

8.4 Security, Signing, and Integrity

8.4.1 Integrated Signing Workflow

Broadcasters should incorporate application and signaling signing (as defined in A/360 [6]) into regular release pipelines, maintain secure key handling procedures, and document certificate rotation and incident response steps. For OTT delivery of BAs, Broadcasters use TLS.

8.4.2 Content Security Policy (CSP)

If a Content Security Policy (CSP) is employed, it should permit operation when all resources are served via the broadcast HTTP mechanism. Any broadband domains permitted by policy should be explicitly listed and operationally monitored.

8.5 Updates and Rollback

8.5.1 Atomic Release Strategy

Broadcasters should publish new application versions to new versioned paths and verified on-air prior to updating signaling to reference the new entry point. Broadcasters should keep previous versions available for a transition period that accounts for Receiver caching and carousel cadence.

8.5.2 Rollback Preparedness

The Broadcaster should retain a last-known-good version, with documented procedures for rapid signaling reversion if field monitoring indicates defects in the new release.

Note: Maintaining parallel availability of N and N-1 versions reduces user impact if a defect is discovered post-deployment.

9. UNSUPPORTED APIs

The A/344 specification provides a set of APIs that BAs can use to interact with a Receiver. However, not all APIs are implemented by all Receivers, which requires BA developers to implement contingency plans for the cases where APIs might not be available for use on a specific Receiver. For example:

- A Receiver that does not allow BAs to change the audio volume will respond to an Audio Volume API request with error code -32601, "Method not found".
- A Receiver that stops the BA as soon as the user accesses parental controls will not support a request for notifications regarding changes to parental controls.

Due to hardware longevity, different generations of Receivers will be active in the field at the same time. Developers need to keep in mind that Receivers may be using older versions of hardware, frameworks, and browsers, and that some features are not available in older Receivers. In addition, different versions of Receivers can support different APIs or different versions of APIs.

10. BA TESTING RECOMMENDATIONS

CTA's CEB32 document [11] and its sub-documents such as CEB32.8 [10] provide details on test assertions that are used to determine whether a Receiver can use the NextGenTV logo. The logo testing evaluates Receiver behavior and support for A/344 APIs but does not test Broadcaster Applications.

Not all assertions listed in the CEB32 documents are mandatory. The assertions also may not be the most current due to release cycle timing. When developing and testing a BA, refer to the assertions to see what Receivers are likely to support. The assertions in the CEB32 documents might not be a comprehensive list of all assertions applicable in the testing of an ATSC 3.0 Receiver. They are subject to change.

10.1 Startup Time

Broadcasters should establish and test internal targets for time-to-first-interaction after service tune, and test against representative Receivers. Meeting those targets typically requires minimizing initial bytes, prioritizing critical objects, and avoiding synchronous broadband fetches during startup.

10.2 Receiver Diversity

Ideally, BA test plans exercise a representative set of Receiver implementations and software versions, including A/344 [2] broadcast HTTP file access, caching and reload behavior, lifecycle transitions, and A/344 WebSocket interactions.

10.3 On-Air Rehearsal

Where operationally feasible, use a rehearsal or low-audience service to validate the end-to-end chain (signaling, scheduling, Receiver behavior) prior to promotion to primary services.

10.4 Event Timing

When BA behavior is synchronized to program time or advertising, drive timing using mechanisms consistent with A/337 [16], and measure drift/latency under on-air conditions.

10.5 Network Availability

Test the BA with and without an internet connection to ensure that the BA can both function with no internet connection and update its behavior gracefully when network conditions change (connected to unconnected or vice versa).

Annex A Avoiding On-Screen Conflict with Broadcaster Application Graphics

A.1 INTRODUCTION

BAs can include interactive menus and selections for supplemental content whose graphics can be displayed on the NEXTGEN TV screen. The BA graphics can be rendered on the screen along with the other more traditional TV components, like the programming video and closed captioning (CC). Receiver manufacturers use display designs to manage the rendering of the video window, the CC, and graphics that are drawn on the screen from applications running natively on the Receiver (e.g. program guide).

The purpose of this annex is to provide guidelines for implementing display management techniques so that conflict between the display of the video window, CC, and graphics from a broadcaster application can be avoided.

A.2 CONFLICT SCENARIOS AND SOLUTION GUIDELINES

Several examples of conflict scenarios, where the display of BA graphics conflict with that of the video window, CC, or both, are presented in this section along with guidelines for how the display components can be better managed.

A.2.1 L-BAR

A current conflict scenario that has been seen on Receivers in the marketplace is one where BA graphics are displayed on the screen in an L-bar layout design, the video window is scaled and positioned so that it is squeezed-back up and to the right, and CC, if enabled, are displayed underneath the BA graphics. Figure A.1 shows an example of this type of conflict scenario.

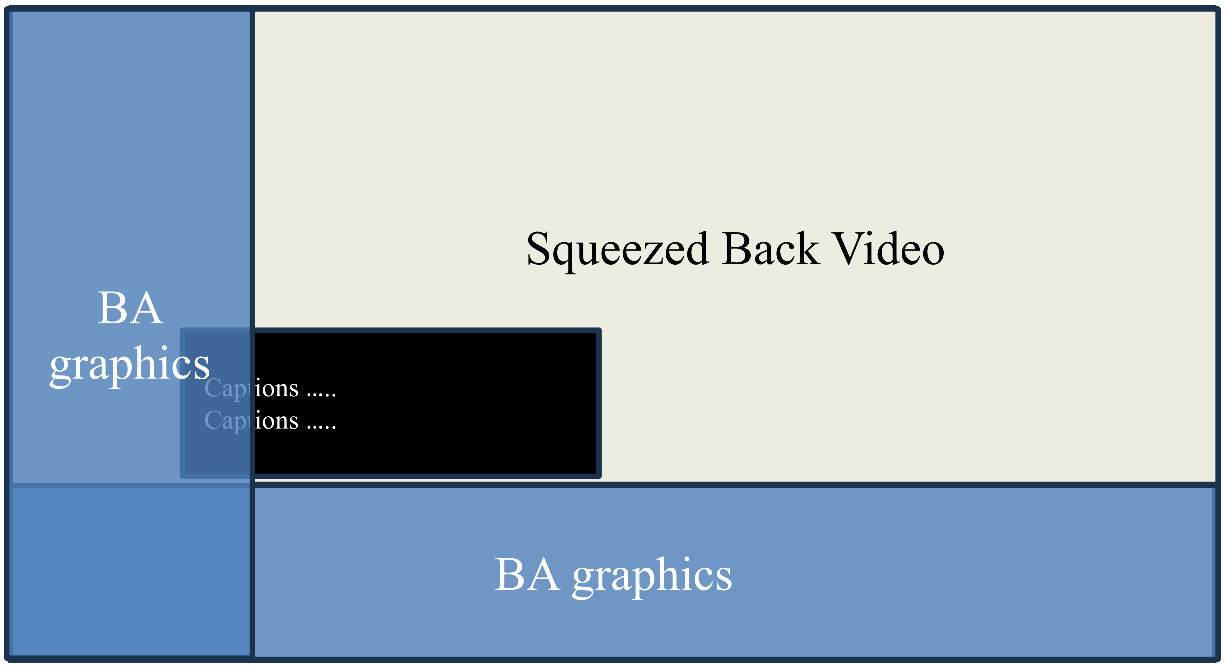


Figure A.1 L-Bar conflict.

The Receiver can resolve this conflict scenario by following these guidelines:

1) Render CC on the top layer of the display.
2) Support both the request and response of the ATSC A/344 Video Scaling and Positioning API.
3) Render CC with its positioning and scaling relative to the video window and not relative to the display.

The BA can use the Video Scaling and Positioning API to request the Receiver to scale and position the video window so that BA graphics do not overlay the video. When the Receiver follows the request from the BA via the Video Scaling and Positioning API and the guidelines above, this conflict scenario will be resolved so that the resulting display looks like Figure A.2.

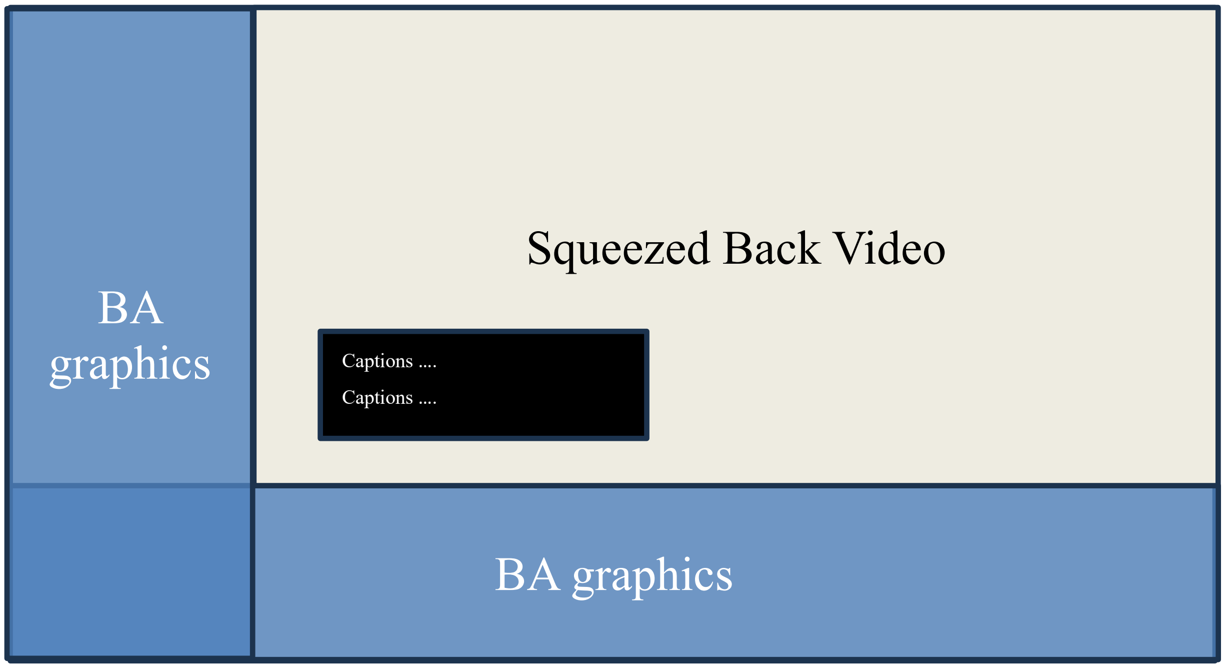


Figure A.2 L-Bar solution.

The video window has been scaled and positioned so that the BA graphics, in this case configured as an L-bar, do not overlap. The CC has scaled and shifted in position in relation to the scaled and shifted video window so that they are no longer obscured by the BA graphics.

The Receiver determines whether the BA has requested the video window to be scaled down so far that the resulting CC will be too small to read. If the CC rendering would result in size that is too small to read, the Receiver sends a response with a JSON object and an "error" object with a "code" set to -8 and the following "message":

```
<-- {
"jsonrpc": "2.0",
"error": {"code": -8, "message": "Requested scaleFactor value is not supported"},
"id": 589
}
```

When the BA design receives this error message, it might make a new Video Scaling and Positioning API request with a lower scaleFactor value appropriate for an alternate graphical layout that takes up less screen area.

A.2.2 CALL-TO-ACTION

Avoid the conflict scenario where CC are enabled for rendering and the BA design results in a call-to-action (CTA) presented on the screen. The CTA is expected to be a minimal graphics representation that appears overlayed on an edge or corner of the video window temporarily and then disappears. The CTA might also be recalled for display temporarily by a remote-control key press by the user. The BA design might use the CTA to activate an enhancement feature with user selection. In the case that CC are enabled, there is a possibility that the CTA graphics might conflict with the CC. If the CC are displayed on top of the CTA, as recommended at all times, the conflict could prevent or make it difficult for the user to select or engage with the CTA. Figure A.3 shows this conflict scenario. Note that the partially-obscured yellow star represents the CTA.

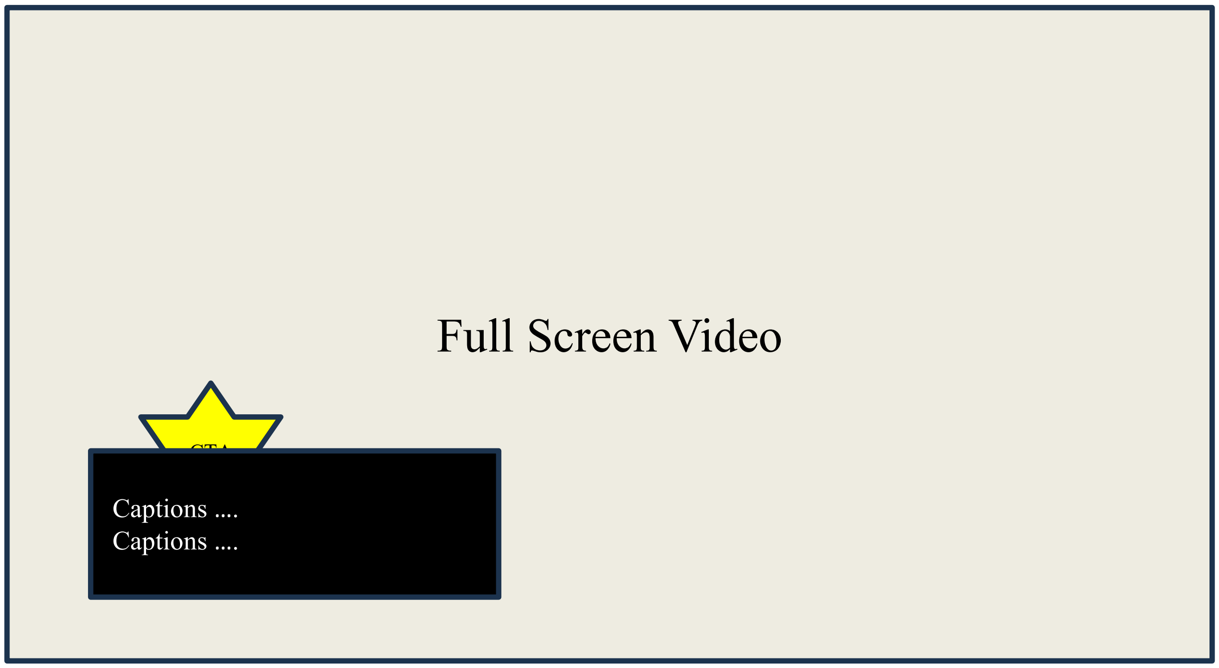


Figure A.3 Call-to-Action conflict.

The Receiver can resolve this conflict scenario by following these guidelines:

1) Render CC on the top layer of the display.
2) Receivers are expected to support both the request and response of the ATSC A/344 Query Closed Captions Enabled/Disabled API.
3) Receivers are expected to support both the request and response of the ATSC A/344 Query Caption Display Preferences API.

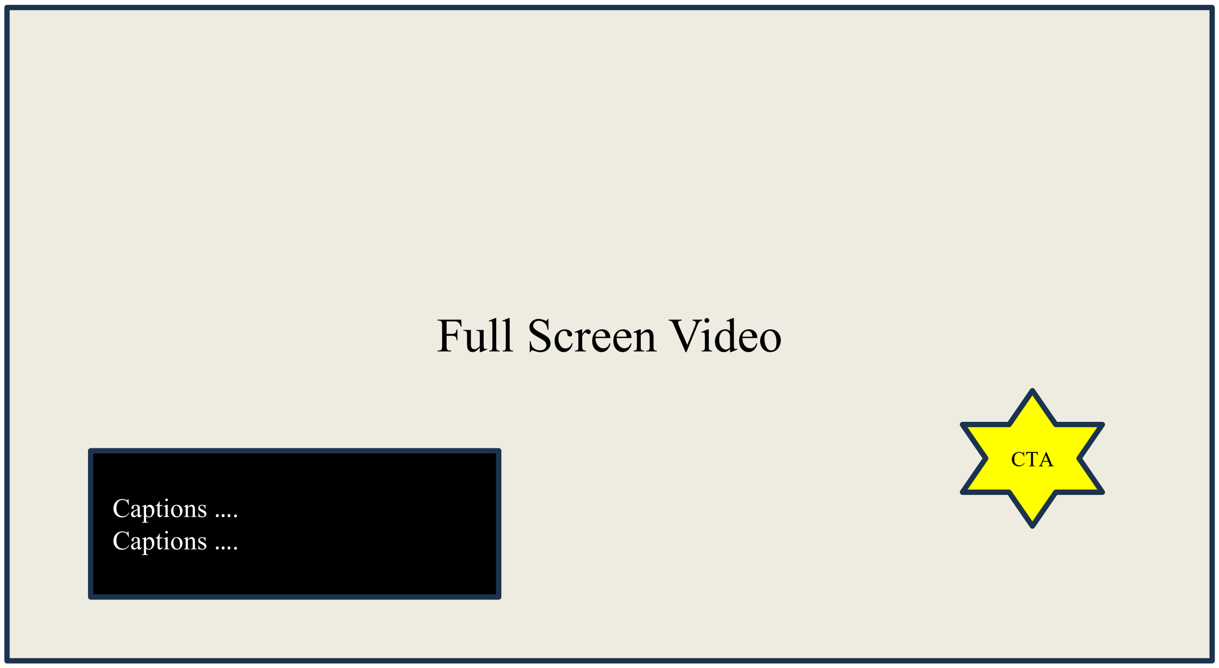


Figure A.4 Call-to-Action solution.

The BA will use the Query Closed Captions Enabled/Disabled API to request from the Receiver if CC are enabled. If CC are enabled, the BA will use the Query Caption Display Preferences API to request CC positioning information to know where the CC are being displayed in reference to the video window. The BA design will result in the CTA being displayed in an alternate area on the screen so that there is no conflict with the CC. Figure A.4 shows the result of the BA design using the Query CC APIs to know where the CC are being displayed and presenting the CTA using an alternate layout.

The BA has determined that the CC have been enabled and where they are being displayed relative to the video window and has presented an alternate layout so that no conflict between the CC and the CTA results.

A.2.3 FULL SCREEN VIDEO ONLY

Another conflict scenario to avoid is one where BA graphics are being displayed but the Receiver does not support scaling of the video window. Since the Receiver is only able to display the video window full-screen, there is no screen area where BA graphics can be rendered so that it is not overlayed on the video window, and possibly in conflict with CC. Figure A.5 shows this conflict scenario.

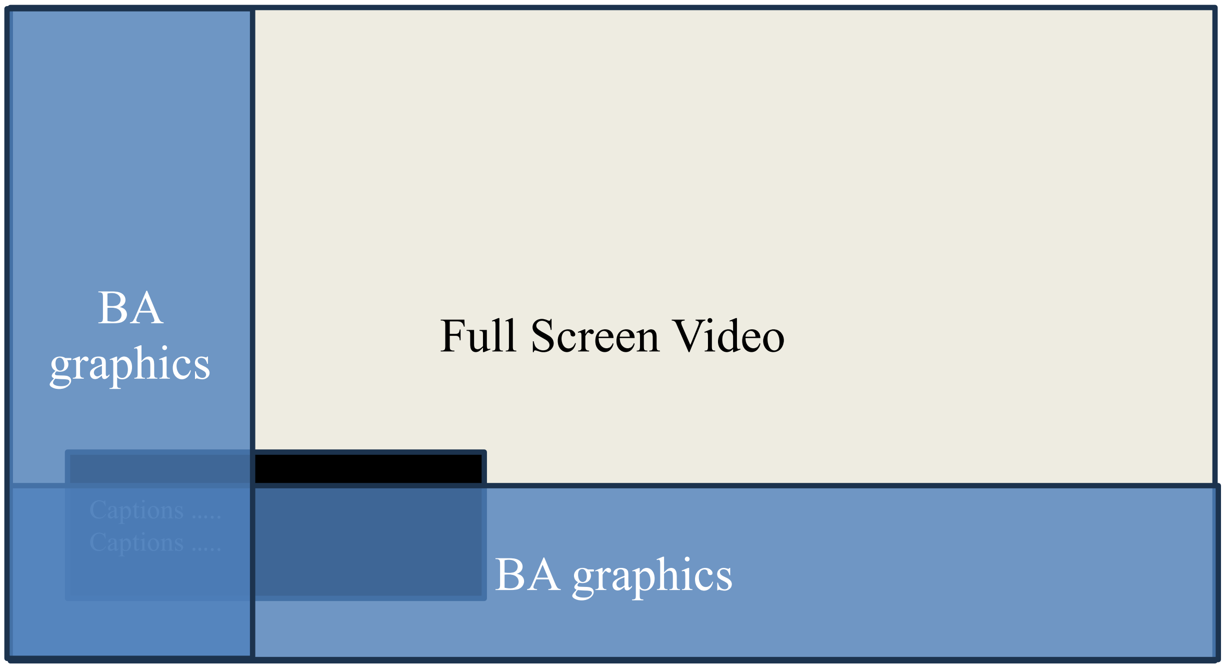


Figure A.5 Full Screen Video Only conflict.

A Receiver that does not support scaling or repositioning of the video window can resolve this conflict scenario by following these guidelines:

1) Render CC on the top layer of the display.
2) Render CC in the center lower-third of the screen as a default.
3) Maintain the CC rendering in a fixed position.
4) Support both the request and response of the ATSC A/344 Video Scaling and Positioning API.

The BA design will use the Video Scaling and Positioning API to request the Receiver to scale and position the video window to try to make some screen space available for BA graphics.

Since the Receiver does not support scaling or repositioning of the video window, the Receiver must send a response with a JSON object and an "error" object with a "code" set to -8 and the following "message":

```
<-- {
"jsonrpc": "2.0",
"error": {"code": -8, "message": "Video scaling and positioning is not supported"},
"id": 589
}
```

The BA design will determine that the Receiver does not support scaling and positioning of the video window after receiving this -8 error message (or if there is no response at all from the Receiver because this scenario's guidelines #4 or 5 are not supported). The BA design will know that the CC will be rendered in the center lower-third of the screen and will output an alternate graphical layout to avoid potential conflicts. Figure A.6 shows the result of the BA design determining that the Receiver cannot support scaling and positioning of the video window and presenting an alternate layout to avoid conflict with CC rendering.

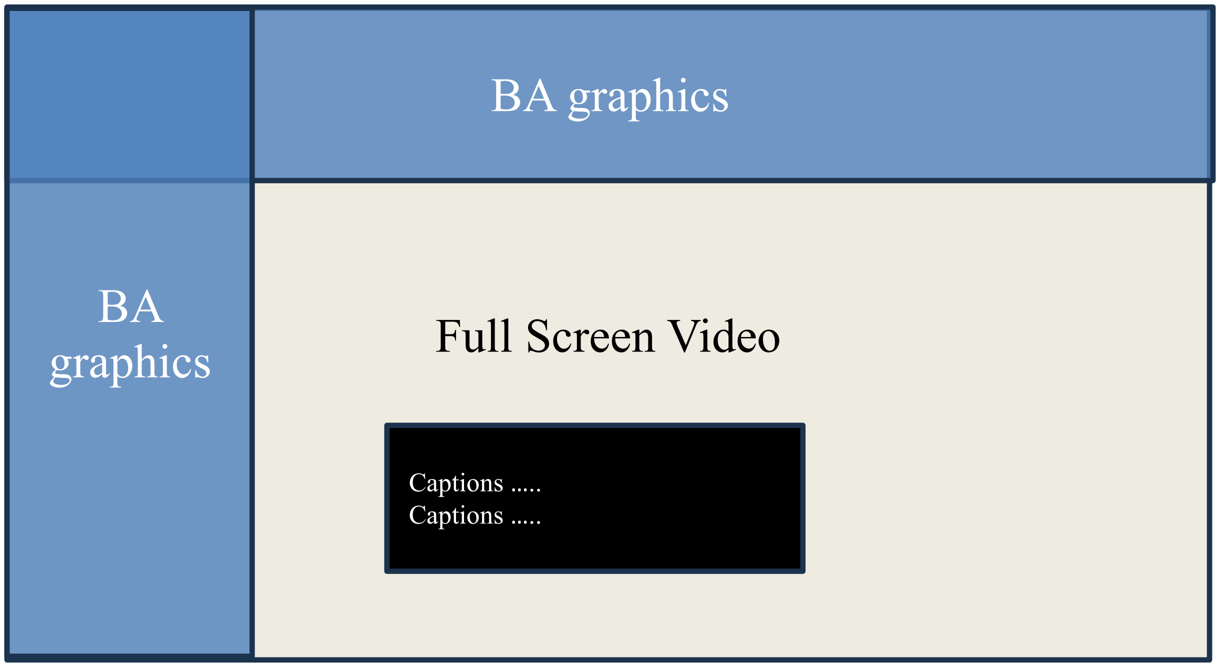


Figure A.6 Full Screen Video Only solution.

The BA design determines that the Receiver does not support scaling and positioning of the video window, so it presents an alternate graphical layout. The layout alternative from what is shown in Figure A.6 shifts the bottom part of the L-bar to the top of the screen so that it is not obscured by the CC. The Receiver renders the CC in the center lower-third of the screen so there is no need for the BA design to avoid conflict by shifting the left part of the L-bar to the right side of the screen.

A.2.4 SCALING OF CC NOT SUPPORTED

The conflict in this scenario is not between the BA graphics and the CC but rather between the CC and the video as both of these display components are scaled and positioned to clear screen space for the BA graphics. This scenario arises when the Receiver does not support the relative scaling of CC as the video window is being scaled. As the video window scales (as described in the L-bar scenario) the Receiver moves the CC with relative positioning to the shifting video window but the size remains the same. As the size of the video window decreases while the size of the CC display region is maintained, the CC display region increasingly covers the majority of the video window. Figure A.7 shows this conflict scenario.

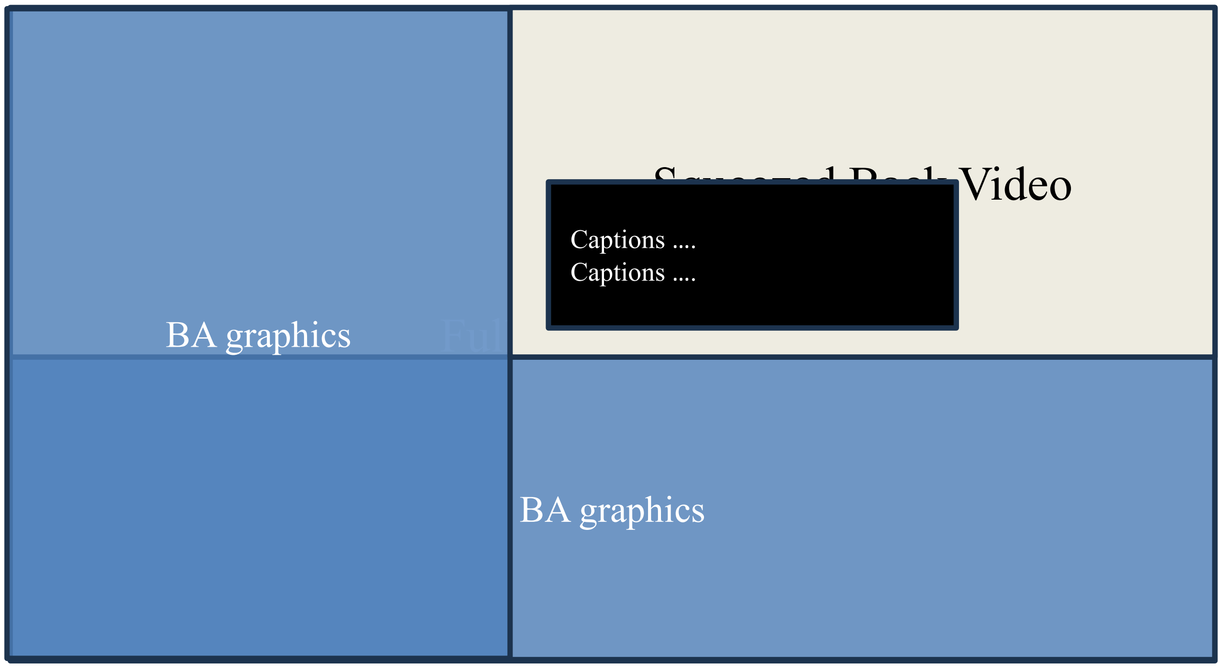


Figure A.7 Scaling of CC Not Supported conflict.

A Receiver that does not support scaling of CC can resolve this conflict scenario by following these guidelines:

1) Render CC on the top layer of the display.
2) Support both the request and response of the ATSC A/344 Video Scaling and Positioning API.
3) Render CC with its positioning relative to the video window and not relative to the display.

The BA will use the Video Scaling and Positioning API to request the Receiver to scale and position the video window so that BA graphics do not overlay the video. A Receiver that does not support scaling of the CC must determine that the BA has requested the video window to be scaled down so far that the resulting CC display region would be too large and overwhelm the video window. If the CC display region would result in a size that covers too much of the video window, the Receiver must send a response with a JSON object and an "error" object with a "code" set to -8 and the same "message" defined for the L-bar scenario(Section A.2.1), provided again here:

```
<-- {
"jsonrpc": "2.0",
"error": {"code": -8, "message": "Requested scaleFactor value is not supported"},
"id": 589
}
```

When the BA design received this error message, it might make a new Video Scaling and Positioning API request with a lower scaleFactor value appropriate for an alternate graphical layout that takes up less screen area. Figure A.8 shows the result of the BA design adjusting to an alternate graphical layout and making a new request to the Receiver using the Video Scaling and Positioning API and a smaller scaleFactor value.

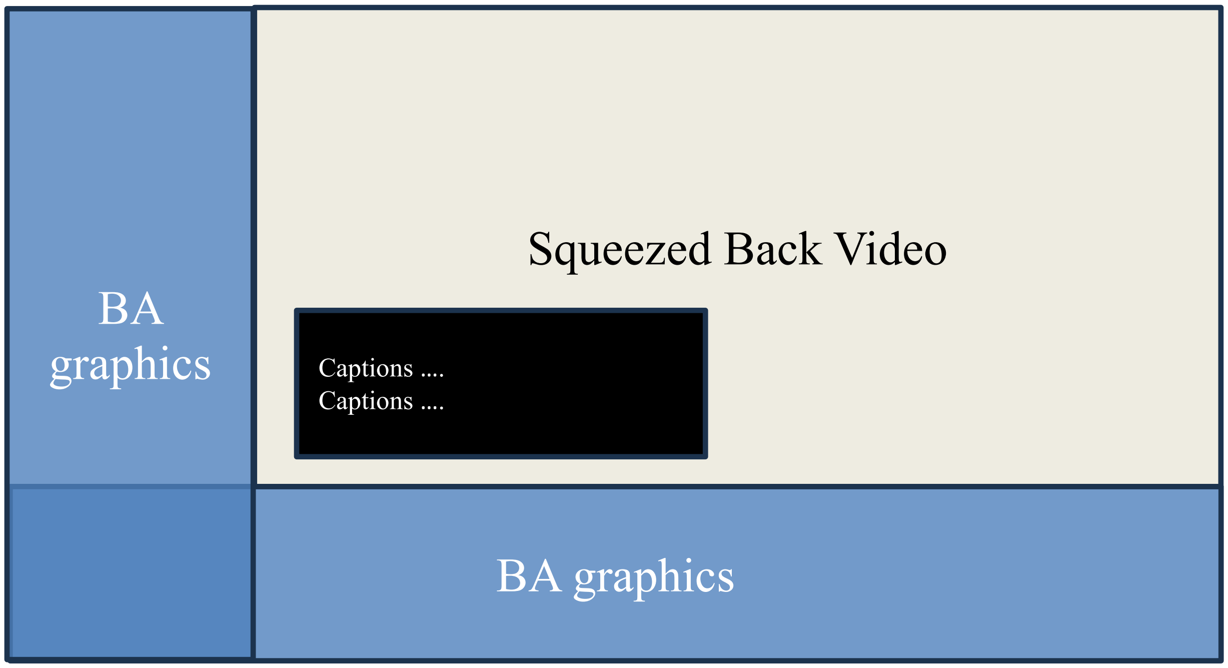
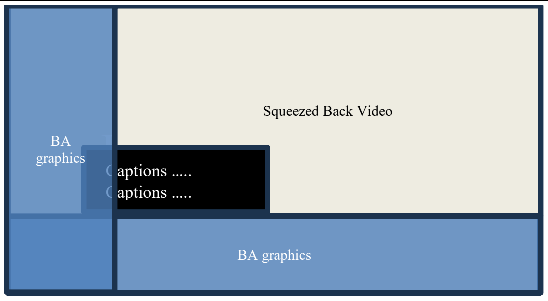
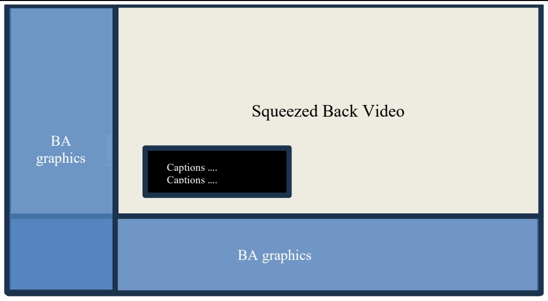
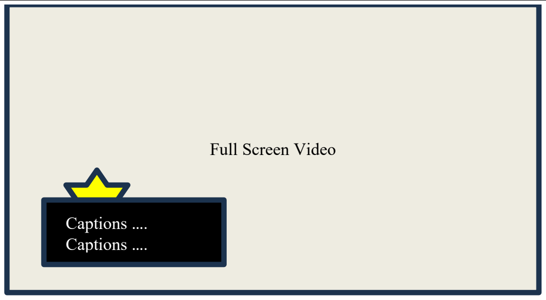
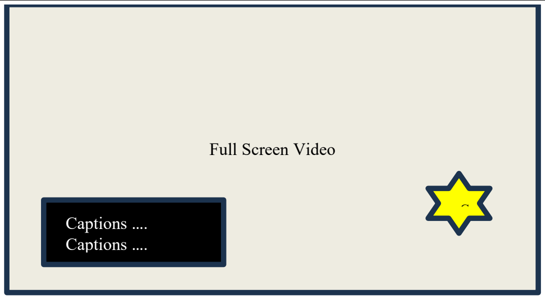
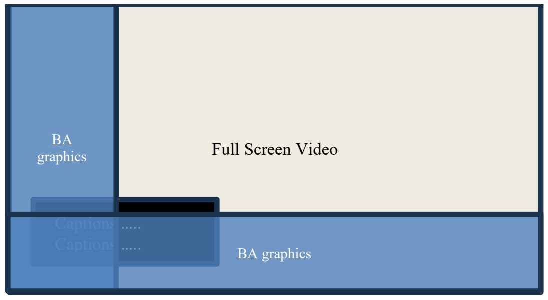
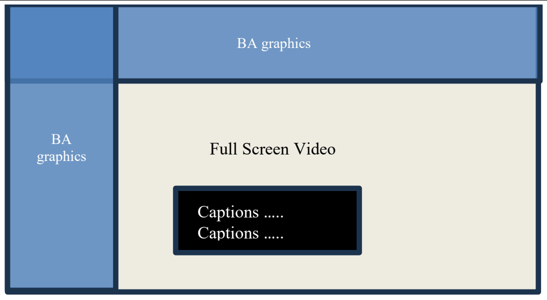
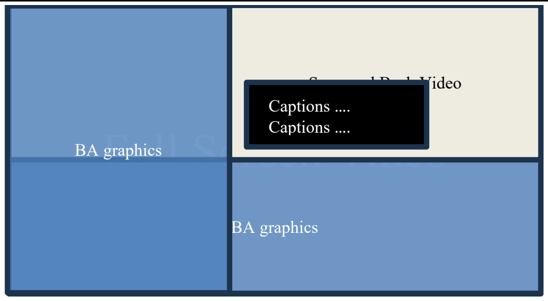
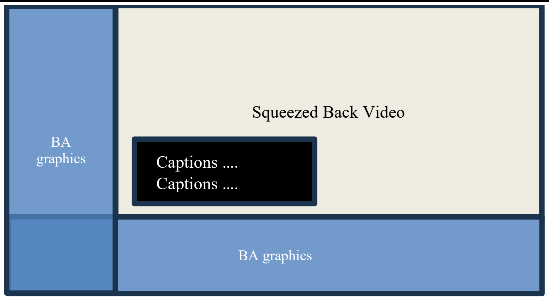


Figure A.8 Scaling of CC Not Supported solution.

The BA design has requested the Receiver to scale and position the video window using a smaller scaleFactor value and has presented the Receiver with an alternate graphical layout that does not overlap the video window. The resulting relative size of the CC display region does not overwhelm the scaled video window.

A.3 CONCLUSION

The Receiver implementation must follow the guidelines for each conflict scenario that are provided above, in Section A.2, to minimize the possibility of conflict between each of the BA graphics, CC, and the video window. The BA design plays a part in avoiding conflict but can only be effective at doing so if the Receiver implementation follows these guidelines.

Don't do this …	Do that
L-Bar	
 Squeezed Back Video BA graphics Captions ..... Captions ..... BA graphics	 Squeezed Back Video BA graphics Captions .... Captions .... BA graphics
Call-to-Action	
	
Full Screen Video Only	
 BA graphics Full Screen Video Captions ..... Captions ..... BA graphics	 BA graphics BA graphics Full Screen Video Captions ..... Captions .....
Scaling of CC Not Supported	
 Squeezed Back Video Captions .... Captions .... BA graphics BA graphics	 Squeezed Back Video BA graphics Captions .... Captions .... BA graphics

– End of Document –