# ATSC Candidate Standard: ATSC 3.0 Interactive Content (A/344)

S34-230r1
29 December 2016

**Advanced Television Systems Committee**
1776 K Street, N.W.
Washington, D.C. 20006
202-872-9160

The Advanced Television Systems Committee, Inc., is an international, non-profit organization developing voluntary standards for digital television. The ATSC member organizations represent the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

Specifically, ATSC is working to coordinate television standards among different communications media focusing on digital television, interactive systems, and broadband multimedia communications. ATSC is also developing digital television implementation strategies and presenting educational seminars on the ATSC standards.

ATSC was formed in 1982 by the member organizations of the Joint Committee on InterSociety Coordination (JCIC): the Electronic Industries Association (EIA), the Institute of Electrical and Electronic Engineers (IEEE), the National Association of Broadcasters (NAB), the National Cable Telecommunications Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE). Currently, there are approximately 150 members representing the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

ATSC Digital TV Standards include digital high definition television (HDTV), standard definition television (SDTV), data broadcasting, multichannel surround-sound audio, and satellite direct-to-home broadcasting.

---

*Note*: The user's attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. One or more patent holders have, however, filed a statement regarding the terms on which such patent holder(s) may be willing to grant a license under these rights to individuals or entities desiring to obtain such a license. Details may be obtained from the ATSC Secretary and the patent holder.

---

This specification is being put forth as a Candidate Standard by the TG3/S34 Specialist Group. This document is a revision of the Working Draft (S34-230r0) dated 16 November 2016. All ATSC members and non-members are encouraged to review and implement this specification and return comments to cs-editor@atsc.org. ATSC Members can also send comments directly to the TG3/S34 Specialist Group. This specification is expected to progress to Proposed Standard after its Candidate Standard period.

**Revision History**

| Version | Date |
|---|---|
| Candidate Standard approved | 29 December 2016 |
| Cyan highlight and text in *blue italics* identify areas that are under development in the committee.<br>Feedback and comments on these points from implementers is encouraged. | |
| Standard approved | Insert date here |

# Table of Contents

1. SCOPE (TO BE COMPLETED) ..... 1
1.1 Introduction and Background 1
1.2 Organization 1
2. REFERENCES ..... 1
2.1 Normative References 1
2.2 Informative References 3
3. DEFINITION OF TERMS ..... 3
3.1 Compliance Notation 3
3.2 Treatment of Syntactic Elements 4
3.2.1 Reserved Elements 4
3.3 Acronyms and Abbreviations 4
3.4 Terms 5
4. OVERVIEW ..... 6
4.1 Application Runtime Environment 6
Action requested by the application 7
API used by the application 7
4.2 Receiver Media Player Display 7
5. ATSC RECEIVER LOGICAL COMPONENTS AND INTERFACES ..... 8
5.1 Introduction 8
5.2 User Agent Definition 9
5.2.1 HTTP Protocols 9
5.2.2 Receiver WebSocket Server Protocol 9
5.2.3 Cascading Style Sheets (CSS) 9
5.2.4 HTML5 Presentation and Control: Image and Font Formats 10
5.2.5 JavaScript 10
5.2.6 2D Canvas Context 10
5.2.7 Web Workers 10
5.2.8 XMLHttpRequest (XHR) 10
5.2.9 Event Source 10
5.2.10 Web Storage 10
5.2.11 Cross-Origin Resource Sharing (CORS) 10
5.2.12 Mixed Content 11
5.2.13 Web Messaging 11
5.2.14 Opacity Property 11
5.2.15 Transparency 11
5.2.16 Full Screen 11
5.2.17 Media Source Extensions 11
5.2.18 Encrypted Media Extensions 11
5.3 Origin Considerations 11
6. BROADCASTERS APPLICATION MANAGEMENT ..... 14
6.1 Introduction 14
6.2 Local Receiver Cache Management 15
6.2.1 Local Receiver Cache Hierarchy Definition 15
6.2.2 Active Service Local Receiver Cache Priority 17

6.2.3 Cache Expiration Time 18
6.3 Broadcaster Application Signaling 18
6.3.1 Broadcaster Application Launch 18
6.3.2 Broadcaster Application Events (Static / Dynamic) 18
6.4 Broadcaster Application Delivery 19
6.4.1 Broadcaster Application Packages 19
6.4.2 Broadcaster Application Package Changes 19
6.5 Security Considerations 19
6.6 Companion Device Interactions 19
7. MEDIA PLAYER 20
7.1 Receiver Media Player 20
7.2 Application Media Player 20
7.2.1 Pull-Model Broadcast Media Streaming 21
7.2.2 Push-Model Media Streaming 21
7.3 Receiver Media Player 21
7.4 Media Content Access 22
7.4.1 Streaming Broadcast Media Content 22
7.4.1.1 Streaming Broadcast Media Content Utilizing RMP 22
7.4.1.2 Streaming Broadcast Media Content Utilizing AMP 23
7.4.2 Streaming Broadband Media Content 24
7.4.3 Streaming Downloaded Media Content 26
7.4.3.1 Playback of Downloaded Media Content Utilizing RMP 26
7.4.3.2 Playback of Downloaded Media Content Utilizing AMP 26
7.5 Downloading Media Content from Broadcast or Broadband 27
8. ATSC 3.0 WEB INTERFACES 29
8.1 Introduction 29
8.2 Interface binding 30
8.2.1 WebSocket Servers 31
8.3 Data Binding 32
8.3.1 Error handling 33
9. SUPPORTED METHODS 34
9.1 Receiver Query APIs 35
9.1.1 Query Content Advisory Rating API 35
9.1.2 Query Closed Captions Enabled/Disabled API 36
9.1.3 Query Service ID API 37
9.1.4 Query Language Preferences API 38
9.1.5 Query Caption Display Preferences API 39
9.1.6 Query Audio Accessibility Preferences API 41
9.1.7 Query MPD URL API 42
9.1.8 Query Receiver Web Server URI API 43
9.2 Asynchronous Notifications of Changes 44
9.2.1 Rating Change Notification API 45
9.2.2 Rating Block Change Notification API 45
9.2.3 Service Change Notification API 46
9.2.4 Caption State Change Notification API 47
9.2.5 Language Preference Change Notification API 47

9.2.6 Personalization Change Notification API 48
9.2.7 Caption Display Preferences Change Notification API 48
9.2.8 Audio Accessibility Preference Change Notification API 51
9.2.9 MPD Change Notification API 53
9.3 Event Stream APIs 53
9.3.1 Event Stream Subscribe API 54
9.3.2 Event Stream Unsubscribe API 55
9.3.3 Event Stream Event API 57
9.4 Request Receiver Actions 59
9.4.1 Acquire Service API 59
9.4.2 Video Scaling and Positioning API 60
9.4.3 XLink Resolution API 62
9.4.4 Subscribe MPD Changes API 63
9.4.5 Unsubscribe MPD Changes API 64
9.4.6 Set RMP URL API 64
9.4.7 Audio Volume API 66
9.5 Media Track Selection API 68
9.6 Media Segment Get API 69
9.7 Mark Unused API 69
10. DASH AD INSERTION ..... 71
10.1 Dynamic Ad Insertion Principles 71
10.2 Overview of XLinks 72
*ANNEX A* : OBSCURING THE LOCATION OF AD AVAILS ..... 74
A.1 Obscuring the Location of Ad Avails 74

# Index of Figures and Tables

**Figure 4.1** Rendering model for application enhancements using RMP. 8
**Figure 5.1** ATSC 3.0 receiver logical components. 9
**Figure 5.2** Application Context Identifier Conceptual Model 13
**Figure 6.2** Example local receiver cache hierachy. 16
**Figure 7.1** Using the RMP to render live broadcast streams. 23
**Figure 7.2** AMP live streaming using pull model. 24
**Figure 7.3** AMP live streaming using push model. 24
**Figure 7.4** Streaming broadband media content utilizing AMP. 25
**Figure 7.5** Streaming broadband media content utilizing RMP. 25
**Figure 7.6** Playback downloaded media content utilizing RMP. 26
**Figure 7.7** Streaming downloaded media content using pull model. 27
**Figure 7.8** Streaming downloaded media content using push model. 27
**Figure 7.9** Downloading media content from broadband. 28
**Figure 7.10** Downloading media content from broadcast. 29
**Figure 8.1** Communication with ATSC 3.0 receiver. 30
**Figure 9.1** RMP audio volume. 67
**Figure 10.2** Example period with XLink. 72
**Figure 10.3** Example remote period. 72
**Figure A.1.1** Public and private MPDs. 74
**Figure A.1.2** Example public MPD. 74
**Figure A.1.3** Example MPD equivalent. 75
**Figure A.1.4** Example ad replacement. 76

**Table 4.1** Application Actions and APIs 7
**Table 8.1** WebSocket Server Functions and URLs 31
**Table 8.2** API Applicability 32
**Table 8.3** JSON RPC Reserved Error Codes 34
**Table 8.4** JSON RPC ATSC Error Codes 34
**Table 9.1** Asynchronous Notifications 44

# ATSC Candidate Standard:
# ATSC 3.0 Interactive Content

## 1. SCOPE (TO BE COMPLETED)

*Section 1 includes the scope, purpose, and organization as individual subheads (as needed).*

### 1.1 Introduction and Background

*Introductory and background text is included in this section if needed.*

### 1.2 Organization

This document is organized as follows:

- Section 1 – The scope, introduction, and background of this specification
- Section 2 – Normative and informative references
- Section 3 – Compliance notation, definition of terms, and acronyms
- Section 4 – Overview of the runtime environment from the system level
- Section 5 – Specification of the runtime environment

## 2. REFERENCES

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

### 2.1 Normative References

The following documents, in whole or in part, as referenced in this document, contain specific provisions that are to be followed strictly in order to implement a provision of this Standard.

[1] ATSC: "ATSC Candidate Standard: Signaling, Delivery, Synchronization, and Error Protection," Doc. A/331, Advanced Television Systems Committee, 21 September 2016. (Work in process.)

[2] ATSC: "ATSC Working Draft: Application Signaling," Doc. A/337, Advanced Television Systems Committee. (Work in process.)

[3] DASH-IF: "Guidelines for Implementation: DASH-IF Interoperability Point for ATSC 3.0," DASH Industry Forum. (Work in process.)

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

[5] IETF: "Tags for Identifying Languages," RFC 5646, Internet Engineering Task Force, September 2009. https://tools.ietf.org/html/rfc5646

[6] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Authentication," Doc. RFC 7235, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7235

[7] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Caching," Doc. RFC 7234, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7234

[8] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests," Doc. RFC 7232, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7232

[9] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," Doc. RFC 7230, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7230

[10] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Range Requests," Doc. RFC 7233, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7233

[11] IETF: "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," Doc. RFC 7231, Internet Engineering Task Force, June, 2014. https://tools.ietf.org/html/rfc7231

[12] IETF: "The Web Origin Concept," RFC 6454, Internet Engineering Task Force, December 2011. https://tools.ietf.org/html/rfc6454

[13] IETF: "The WebSocket Protocol," RFC 6455, Internet Engineering Task Force, December 2011. https://tools.ietf.org/html/rfc6455

[14] IETF: "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, Internet Engineering Task Force, January 2005. https://tools.ietf.org/html/rfc3986

[15] ISO/IEC "Information technology – Coding of audio-visual objects – Part 22: Open Font Format," Doc. ISO/IEC 14496-22:2015, International Organization for Standardization, Geneva, 1 October 2015.

[16] ISO/IEC: ISO/IEC 23009-1:2014, "Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats," International Organization for Standardization, 2nd Edition, 5/15/2014.

[17] W3C: "CSS Backgrounds and Borders Module Level 3," W3C Candidate Recommendation, Worldwide Web Consortium, 9 September 2014. https://www.w3.org/TR/2014/CR-css3-background-20140909/

[18] W3C: "CSS Color Module Level 3," W3C Recommendation. Worldwide Web Consortium, 7 June 2011. https://www.w3.org/TR/css3-color/

[19] W3C: "CSS Transforms Module Level 1," W3C Working Draft, Worldwide Web Consortium, 26 November 2013. http://www.w3.org/TR/2013/WD-css-transforms-1-20131126/

[20] W3C: "Definition of User Agent", Worldwide Web Consortium, 16 June 2011. https://www.w3.org/WAI/UA/work/wiki/Definition_of_User_Agent

[21] W3C: "Encrypted Media Extensions," W3C Editor's Draft, World Wide Web Consortium, 3 September 2015, https://w3c.github.io/encrypted-media/

[22] W3C: "Geolocation API Specification," W3C Editor's Draft, World Wide Web Consortium, 11 July 2014. http://dev.w3.org/geo/api/spec-source

[23] W3C: "HTML5: A vocabulary and associated APIs for HTML and XHTML," W3C Recommendation, World Wide Web Consortium, 28 October 2014. http://www.w3.org/TR/html5/

[24] W3C: "ISO BMFF Byte Stream Format," World Wide Web Consortium, 9 March 2015. http://www.w3.org/TR/media-source/isobmff-byte-stream-format.html

[25] W3C: "Media Queries," W3C Recommendation, Worldwide Web Consortium, 19 June 2012. http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/

[26] W3C: "Media Source Extensions," W3C Proposed Recommendation, World Wide Web Consortium, 4 October 2016. https://www.w3.org/TR/media-source/

[27] W3C: "Media Source Extensions Byte Stream Format Registry," World Wide Web Consortium, 9 March 2015. http://www.w3.org/TR/media-source/byte-stream-format-registry.html
[28] W3C: "Mixed Content," W3C Candidate Recommendation, Worldwide Web Consortium, 8 October 2015. http://www.w3.org/TR/2015/CR-mixed-content-20151008/
[29] W3C: "WOFF File Format 1.0," W3C Recommendation, Worldwide Web Consortium, 13 December 2012. http://www.w3.org/TR/2012/REC-WOFF-20121213/
[30] W3C: "XML Linking Language (XLink)," Recommendation Version 1.1, Worldwide Web Consortium, 6 May 2010. http://www.w3.org/TR/xlink11/

### 2.2 Informative References

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

[31] CTA: "Digital Television (DTV) Closed Captioning," Doc. CTA-708, Consumer Technology Association, Arlington, VA.
[32] JSON-RPC: "JSON-RPC 2.0 Specification," JSON-RPC Working Group. http://www.jsonrpc.org/specification
[33] W3C: "CSS Multi-column Layout Module," W3C Candidate Recommendation, Worldwide Web Consortium, 12 April 2011. http://www.w3.org/TR/2011/CR-css3-multicol-20110412/
[34] W3C: "CSS Namespaces Module Level 3," W3C Recommendation, Worldwide Web Consortium, 29 September 2011. http://www.w3.org/TR/2014/REC-css-namespaces-3-20140320/
[35] W3C: "CSS Text Module Level 3," W3C Last Call Working Draft, Worldwide Web Consortium, 10 October 2013. http://www.w3.org/TR/2013/WD-css-text-3-20131010/
[36] W3C: "CSS Writing Modes Level 3," W3C Candidate Recommendation, Worldwide Web Consortium, 5 December 2015. https://drafts.csswg.org/css-writing-modes/
[37] W3C: "Geolocation API Specification," W3C Recommendation, Worldwide Web Consortium, 24 October 2013. http://www.w3.org/TR/2013/REC-geolocation-API-20131024/

## 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 [4] shall 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:

**shall** – This word indicates specific provisions that are to be followed strictly (no deviation is permitted).

**shall not** – This phrase indicates specific provisions that are absolutely prohibited.

**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.

## 3.2 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
**ATSC** Advanced Television Systems Committee
**CDN** Content Delivery Network
**CSS** Cascading Style Sheets
**DASH** Dynamic Adaptive Streaming over HTTP
**ESG** Electronic Service Guide
**HELD** HTML Entry pages Location Description
**HTML** HyperText Markup Language
**HTTP** HyperText Transfer Protocol
**IPTV** Internet Protocol Television
**JSON** JavaScript Object Notation
**MPD** Media Presentation Description
**RMP** Receiver Media Player
**ROUTE** Real-Time Object Delivery over Unidirectional Transport
**SSM** Service Signaling Manager
**URL** Uniform Resource Locator
**W3C** Worldwide Web Consortium
**XML** eXtensible Markup Language

## 3.4 Terms

The following terms are used within this document.

**Application Context Identifier** – The Application Context Identifier is a URI that allows Broadcaster Applications to be grouped. Resources associated with a Broadcaster Application and hence an Application Context Identifier will be made available to another Broadcaster Application if and only if each has the same Application Context Identifier. Details of the Application Context Identifier syntax are specified in the HELD [2].

**Broadcaster Application** – A Broadcaster Application is used herein to refer to the functionality embodied in a collection of files comprised of an HTML5 document referencing other HTML5, CSS, JavaScript, image and multimedia resources that are provided by a broadcaster in an ATSC 3.0 service. The Broadcaster Application refers to the client-side functionality of the broader Web Application that provides the interactive service. The distinction is made because the broadcaster only transmits the client-side documents and code. The server-side of this broader Web Application is implemented by an ATSC 3.0 receiver and has a standardized API for all applications. No server-side application code can be supplied by the broadcaster. The broadcaster may provide Web-based documents and code that work in conjunction with the Broadcaster Application over broadband making the Broadcaster Application a true Web Application. The collection of files making up the Broadcaster Application can be delivered over the web in a standard way or over broadcast as a ROUTE file package.

**Entry Page** – The Entry Page is the initial HTML5 document referenced by application signaling that should be loaded first into the User Agent. It is part of or comprises the entire Launch Package.

**Launch Package** – The Launch Package contains one or more files that comprise the functionality of the Broadcaster Application. The Launch Package includes the Entry Page and perhaps additional supporting files include JavaScript, CSS, image files and other content. Note that if the Entry Page is delivered separately it may still be referred to as the Launch Package.

**Local Receiver Cache** – The Local Receiver Cache is a conceptual storage area where information from the broadcast is collected for retrieval through the Receiver Web Server. This document refers to the Local Receiver Cache as if it were implemented as actual storage though this is for convenience only. The actual implementation of the Local Receiver Cache is beyond the scope of the present document.

**Persistent Storage** – TBD.

**Receiver** – TBD.

**Receiver Web Server** – The Receiver Web Server provides a means for a User Agent to gain access to files delivered over ROUTE that conceptually reside in the Local Receiver Cache.

**reserved** – Set aside for future use by a Standard.

**User Agent** – Defined by W3C in [20] as "… any software that retrieves, renders, and facilitates end user interaction with Web content, or whose user interface is implemented using Web technologies."

**Web Application** – A Web Application is a client/server program accessed via the web using URLs. The client-side software is executed by a User Agent.

# 4. OVERVIEW

## 4.1 Application Runtime Environment

This specification defines the details of an environment that is required for broadcaster's applications to run. The broadcaster's application is an HTML5 application running in a User Agent, which utilizes the User Agent APIs specified in this document. A broadcaster-provided HTML5 application, referenced herein as a "Broadcaster Application," consists of HTML pages and other resources such as JavaScript, CSS, XML and multimedia files. The collection of these files may be packaged as one file and via broadcast utilizing ROUTE, or can be fetched via broadband. These pages and resources are then made available to the User Agent. In the broadband environment, launching of an application behaves the same as in a normal web environment with no specialized behavior or intervention from a receiver.

The Broadcaster Application executes inside a W3C-compliant User Agent accessing some of the graphical elements of the receiver to render the user interface or accessing some of the resources or information provided by the receiver. If a Broadcaster Application requires access to resources such as information known to the receiver, or if the application requires the receiver to perform a specific action that is not defined by the User Agent APIs, the Broadcaster Application sends the request by using the receiver's Receiver WebSocket Server utilizing a set of JSON-RPC messages, as defined in this specification.

The JSON-RPC messages defined in this specification provide the APIs that are required by the Broadcaster Application to access the resources that are otherwise not reachable. These JSON-RPC messages allow the Broadcaster Application to query information that is gathered or collected in the receiver, to receive notifications that are signaled via broadcast signaling, and to request performing of actions that are not otherwise available via the standard JavaScript APIs.

There are noteworthy differences between an HTML5 application deployed in a normal web environment and one deployed in an ATSC 3.0 broadcast environment. In the ATSC 3.0 broadcast environment, a Broadcaster Application can:

- Access resources from broadcast or broadband;
- Request receivers to perform certain functions that are not otherwise available via the JavaScript APIs, such as:
  - Utilizing the media player provided by the receiver (called the Receiver Media Player) to
    - Stream media content via broadcast signaling delivery mechanism
    - Stream media content (i.e. unicast) via broadband delivery mechanism
    - Playback media content that has been downloaded via broadcast or broadband delivery mechanism;
  - Utilizing MSE and EME to play media content streamed over broadcast or broadband
- Query information that is specific to the reception of TV services, for example, the status of closed caption display and language references, and receive notifications of changes in this information;
- Receive notifications of "stream events" that are embedded in the media content or signaling, when that content is being played by the Receiver Media Player.

Another noteworthy difference between the two models is that in the normal web environment, the viewer is in direct control of launching an HTML5 application by specifying the URL of a desired website. In the ATSC 3.0 environment, although the user still initiates the

action by selecting a Service, the actual application URL is not explicitly selected by the viewer and instead is provided via a broadcast signaling. In this case, it is the responsibility of the receiver using its User Agent to launch or terminate the application referenced by a URL provided in broadcast signaling.

The Broadcaster Application relies on a set of features that are provided via the User Agent. Although it is beyond the scope of this specification to describe how the pages of a Broadcaster Application are provided to the User Agent, it is recommended that standard web technologies should be used to serve the pages.

Table 4.1 shows which type of API a broadcaster-provided application uses to access the features provided by the receiver.

**Table 4.1** Application Actions and APIs

| Action Requested by the Application | API Used by the Application |
|---|---|
| Requesting to download a media file from broadband | W3C APIs provided via the user-agent |
| Query information related to user display and presentation preferences, including languages, accessibility options, and closed caption settings | Receiver WebSocket Server APIs, described in this specification in Section 9.1 |
| Requesting to stream downloaded media file from broadcast | Via push or pull model, described in this specification in Section Sections 9.1 and 9.3.2 |
| Requesting to stream downloaded media file from broadband | Via push or pull model, described in this specification in Section Sections 9.1 and 9.3.2 |
| Requesting the Receiver Media Player to play a broadband-delivered media stream | Receiver WebSocket Server APIs, described in this specification in Section Sections 9.1 and 9.3.2 |
| Subscribing (or un-subscribing) to stream event notifications that are sent as part of ROUTE/DASH over broadcast | Receiver WebSocket Server APIs, described in this specification in Sections 9.1 and 9.3.2 |
| Querying the receiver to learn the identity of the currently-selected broadcast service | Receiver WebSocket Server APIs, described in this specification in Section 9.1.3 |
| Receiving notice of changes to above user display and presentation preferences | Receiver WebSocket Server APIs, described in this specification in Section 9.1.7 |
| Receiving stream event notifications that are sent as part of ROUTE/DASH over broadcast | Receiver WebSocket Server APIs, described in this specification in Section 9.3.3 |
| Requesting the receiver to select a new broadcast service | Receiver WebSocket Server APIs, described in this specification in Section 9.4.1 |

## 4.2 Receiver Media Player Display

The RMP presents its video output behind any visible output from the broadcaster application. Figure 4.1 illustrates the relationship and the composition function performed in the receiver. The aspect ratio of the video display window shall be 16:9 (width:height) and the “width” media feature of CSS MediaQuery [21] shall align with the width of the video display window.

Figure 4.1 illustrates two examples. In the example on the left, the graphical output from the Broadcaster Application is overlaid onto the full-screen video being rendered by the Receiver Media Player. For the linear A/V service with application enhancement, the application may instruct the Receiver Media Player to scale the video, as it may wish to use more area for graphics. A JSON-RPC message as described in Section 10.8 is used to instruct the RMP to scale and position the video it renders. This scenario is illustrated in the example shown on the right side of the figure. The application will likely want to define the appearance of the screen surrounding the video inset. It can do that by defining the background in such a way that the rectangular area where the RMP video is placed is specified as transparent.

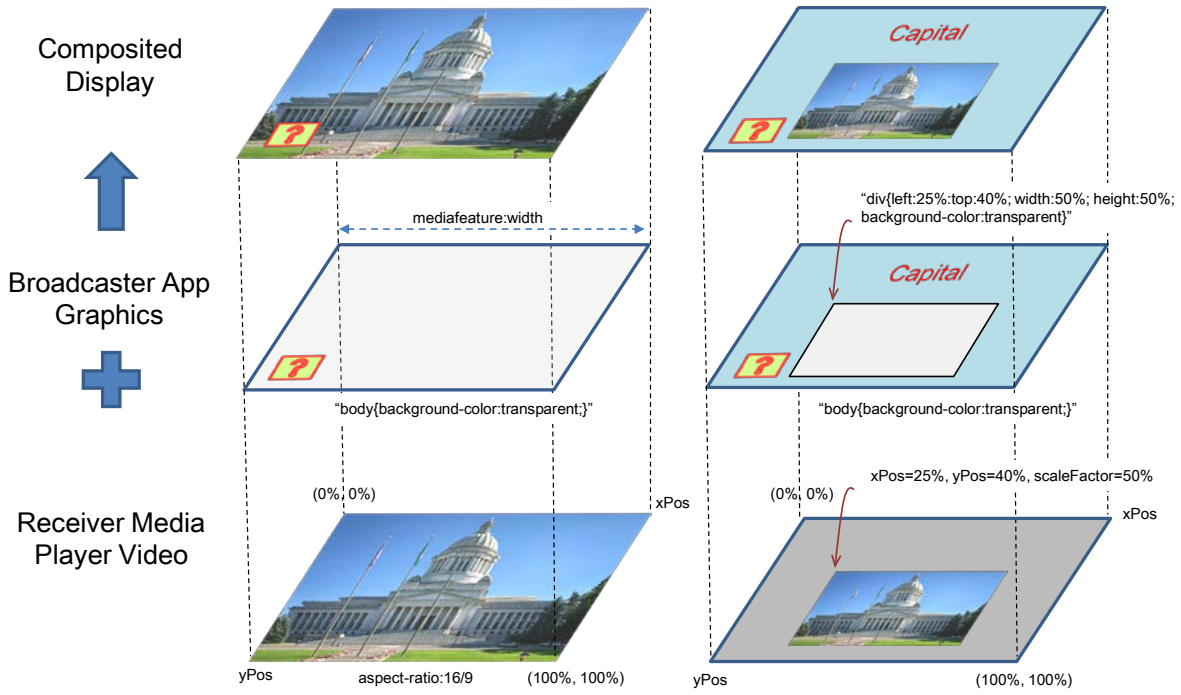


**Figure 4.1** Rendering model for application enhancements using RMP.

The receiver shall only display the area of the HTML5 page corresponding to the video display (16:9). Thus, receivers do not support vertical or horizontal scrolling.

Note that the display of closed captioning will be directly related to the current audio selected and is expected to be presented on top of all other content and video. However, the display of closed captioning is the responsibility of the receiver and out of the scope of the present document.

## 5. ATSC RECEIVER LOGICAL COMPONENTS AND INTERFACES

### 5.1 Introduction

An ATSC 3.0 receiver may be composed of several logical components, which are described in this section. Details about the internal structure of the components, the internal communication mechanisms or their implementation are beyond the scope of this document. Also, in practice several of the given logical components can be combined into one component or one logical component can be divided into multiple components. Figure 5.1 shows the logical components of an ATSC 3.0 receiver. Although the software stack shows a layering architecture, it does not necessarily mean one module must use the layer below to access other modules in the system, with the exception of the Broadcaster Applications, which are run in the User Agent implementation provided by the receiver, which complies with the APIs specified in this specification.

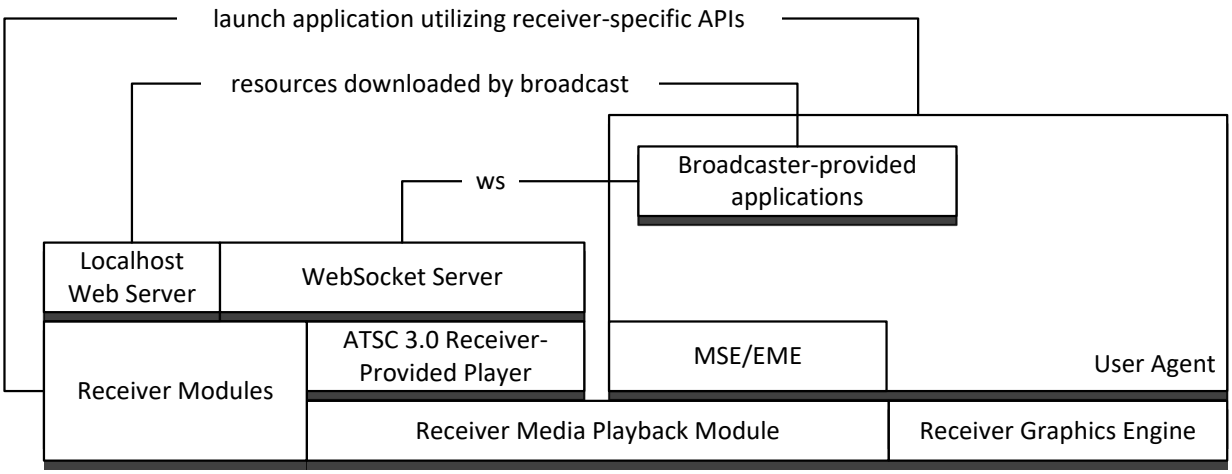


**Figure 5.1** ATSC 3.0 receiver logical components.

## 5.2 User Agent Definition

Terminals shall implement an HTML5 User Agent that complies with all normative requirements in the W3C HTML5 Specification [23]. In addition, the features described in the following sections shall be supported.

### 5.2.1 HTTP Protocols

The User Agent shall implement the HTTP protocols specified in RFC 7230 through RFC 7235, references [9], [11], [8], [10], [7], and [6]. User Agents that implement the HTTP protocol must implement the Web Origin Concept specification and the HTTP State Management Mechanism specification (Cookies) as well. These are referenced in [23] as [HTTP], [ORIGIN], and [COOKIES].

### 5.2.2 Receiver WebSocket Server Protocol

The User Agent shall support the WebSocket protocol, referenced as [WSP] in [23].

### 5.2.3 Cascading Style Sheets (CSS)

The User Agent shall support the `text/css` media type of the [CSS] reference as defined in [23] as follows:

1) It must satisfy CSS Level 2 Revision 1 conformance requirements, and
2) If the Receiver implements the capability controls, or is capable of direct interface to a screen, then it must support the screen CSS media type.

The User Agent shall support the features defined by the following CSS Level 3 modules:

- W3C CSS Background and Borders [17];
- W3C CSS Transforms [19]; and
- The [CSSUI], [CSSANIMATIONS], and [CSSTRANSITIONS] references as defined in [23].

The User Agent shall support the features defined by the following CSS Level 3 modules:

- CSS Image Values and Replaced Content [CSSIMAGES] in [23]
- CSS Multi-Column Layout [33]
- CSS Namespaces [34]
- CSS Selectors, referenced as [SELECTORS] in [23]

- CSS Text [35]
- CSS Values and Units [CSSVALUES] in [23]
- CSS Writing Modes [36]

The User Agent shall support the `@font-face` rule in the context of using the `text/css` media type of the [CSSFONTS] in [23].

To allow the application to adjust the document size to match the device screen size, the User Agent shall implement CSS3 Media Queries [25], referenced as [MQ] in [23].

#### 5.2.4 HTML5 Presentation and Control: Image and Font Formats

The User Agent shall support the following image formats:

- `image/svg` media type of the [SVG] reference as defined in [23]
- `image/jpeg` media type as defined by the [JPEG] reference as defined in [23]
- `image/png` media type as defined by the [PNG] reference as defined in [23]
- `image/gif` media type as defined by the [GIF] reference as defined in [23]

The User Agent shall support the `application/font-woff` media type as defined by W3C in [29] for use with the `@font-face` rule, and, more specifically, shall support the OpenType font format as defined by ISO/IEC 14496-22 [15] when encapsulated in a WOFF file.

The User Agent shall support the `application/opentype` media type as defined by ISO/IEC 14496-22 [15] for use with the `@font-face` rule; and, further, shall support any media resource in such context regardless of its media type, if it can be determined that it (the media resource) conforms to the OpenType font format defined by ISO/IEC 14496-22 [15].

#### 5.2.5 JavaScript

The User Agent shall support JavaScript as defined in the [ECMA262] reference in [23].

#### 5.2.6 2D Canvas Context

The User Agent shall support the "`2d`" canvas context type as defined by the [CANVAS2D] reference in [23].

#### 5.2.7 Web Workers

The User Agent shall support the `SharedWorkerGlobalScope`, `DedicatedWorkerGlobalScope`, and related interfaces of the [WEBWORKERS] reference in [23].

#### 5.2.8 XMLHttpRequest (XHR)

The User Agent shall support the `XMLHttpRequest` and related interfaces of the [XHR] reference in [23].

#### 5.2.9 Event Source

The User Agent shall support the `EventSource` interface and related features of the [EVENTSOURCE] reference in [23].

#### 5.2.10 Web Storage

The User Agent shall support the `WindowSessionStorage` interface, `WindowLocalStorage` interface, and related interfaces of the [WEBSTORAGE] reference in [23].

#### 5.2.11 Cross-Origin Resource Sharing (CORS)

The User Agent shall support the [ORIGIN] specification referenced in [23].

#### 5.2.12 Mixed Content

The User Agent shall handle fetching of content over unencrypted or unauthenticated connections in the context of an encrypted and authenticated document according to the W3C Mixed Content specification [28].

#### 5.2.13 Web Messaging

The User Agent shall support the Web Messaging specification referenced as [WEBMSG] in [23].

#### 5.2.14 Opacity Property

The User Agent shall support the opacity style property of the [CSSCOLOR] as defined in [23]. In addition, it must support the `<color>` property value type as defined therein in any context that prescribes use of the CSS `<color>` property value type.

#### 5.2.15 Transparency

The background of the User Agent's drawing window is transparent by default. Thus, for example, if any element in the web page (such as a table cell) includes a CSS style attribute `"background-color: transparent"` then video content presented by the Receiver Media Player (see Section 4.2) can be visible. Note that certain areas can be specified as transparent while others are opaque.

#### 5.2.16 Full Screen

The display aspect ratio of the application presentation window shall be 16:9. The Receiver shall present the document such that it is fully visible.

#### 5.2.17 Media Source Extensions

The User Agent shall support Media Source Extensions [26].

#### 5.2.18 Encrypted Media Extensions

The User Agent shall support Encrypted Media Extensions [21].

### 5.3 Origin Considerations

Each file that is delivered via broadband has the usual absolute URL associated with it. Each file that is delivered via broadcast has a relative URL reference associated with it, signaled in the broadcast, and it also has one or more Application Context Identifiers associated with it, signaled in the broadcast. As specified below, receivers assign to each broadcast file a prefix that converts the relative URL reference to one or more absolute URLs, taking its Application Context Identifier(s) into account.

The origin of a web resource is defined in RFC 6454 [12]. The algorithm used by an ATSC 3.0 receiver to generate the portion of a URI that determines the origin of a broadcast file is an implementation detail and is out of scope of this specification except that the portion of the URI shall conform to the restrictions specified below. Note that a resource from a broadband source will have an origin defined by the web server hosting the resource.

The URI provided by the receiver shall have the following form for any resource within an Application Context Identifier environment:

`<prefix>/<pathToResource>`

where:

`<prefix>` is defined as `<scheme>://<authority>[/<pathPrefix>]`

`<scheme>` and `<authority>` are standard URI elements as defined in RFC 3986 [14],

`<pathPrefix>` is an optional path prefix prepended to the path; and

`<pathToResource>` is the relative path supplied by ROUTE (e.g., `Content-Location` from the EFDT [1])

As described in Section 6, when delivered via broadcast, a Broadcaster Application will be launched with a URI defined by the receiver. The *prefix* of the URI, that is, the portion of the URI prior to the relative path supplied by ROUTE transmission, shall uniquely correspond to the Application Context Identifier, described later in this section. The algorithm used by the ATSC 3.0 receiver to create the prefix of the URI used to launch the Broadcaster Application is an implementation detail and out of scope of the present document.

While the algorithm used to generate the URI for a given Application Context Identifier should be consistent, that is, the algorithm should repeatedly produce the same URI over time, Broadcaster Applications cannot rely on the uniqueness of either origin or path prefix provided by the receiver as part of the Entry Page URI. The Broadcaster Application will need to manage a local name space for setting cookies and other local User Agent storage elements.

If the current Application Context Identifier remains the same, even though the Entry Page may change, all the associated resources that may have been cached for that context will be available through the Receiver Web Server. This is referred to herein as the Application Context Identifier environment. Entry Page changes are signaled in application signaling as described in Section 6.3.1.

If the Application Context Identifier changes, the receiver may reuse an environment previously created for that Application Context Identifier or a new environment may be created. The receiver may elect to maintain any previous Application Context Identifier environment, albeit inaccessible to Broadcaster Applications with differing Application Context Identifiers, on the presumption that these previous Application Context Identifier environments may be needed soon. Alternatively, the receiver can free the resources associated with the previous Application Context Identifier environment. If a file is not cached, the Receiver Web Server may respond to the request by waiting for the next delivery of the file or with an error code. The decisions on caching files are entirely left to the receiver implementation and are out of scope.

The Application Context Identifier is a URI that allows Broadcaster Applications to be grouped. Resources associated with a Broadcaster Application and hence an Application Context Identifier shall be made available to another Broadcaster Application if and only if each has the same Application Context Identifier. Details of the Application Context Identifier syntax are specified in the HELD [2].

An Application Context Identifier shall be associated with at least one and perhaps many Broadcaster Applications. An individual Broadcaster Application shall be associated with a single Application Context Identifier. Each Application Context Identifier forms a unique conceptual environment in which the receiver is expected to comingle resources for use by the associated Broadcaster Applications.

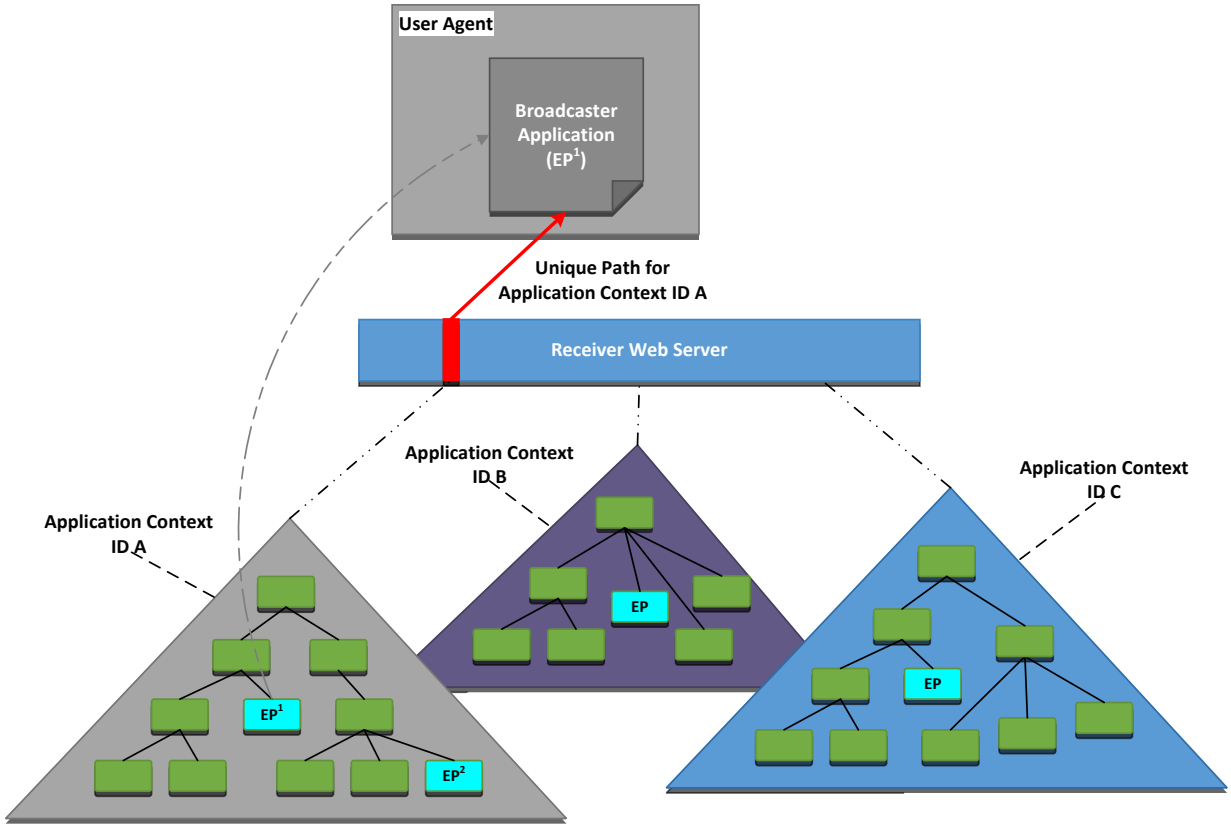


**Figure 5.2** Application Context Identifier Conceptual Model.

Figure 5.2 provides a conceptual model of how Application Context Identifiers are related to the Broadcaster Application and broadcast files. The diagram provides an example of how resources (files and directories) are made available to a Broadcaster Application through the Receiver Web Server using URIs unique to a given Application Context Identifier. In the figure, the Broadcaster Application is shown operating in the User Agent having been launched using Entry Page, ***EP*$^{1}$**. At some point while ***EP*$^{1}$** is active, application signaling could launch the Entry Page designated as ***EP*$^{2}$**. In this case, the Application Context Identifier environment and access to it would remain constant with the User Agent loaded with ***EP*$^{2}$**. The receiver may or may not provide access to the other environments corresponding to different Application Context Identifiers. The availability of these environments while another Application Context Identifier environment is active is receiver dependent and out of scope. Broadcaster Applications shall restrict access to resources within their own Application Context Identifier environment as provided by the receiver, or to the Internet if broadband is available.

Broadcaster Applications delivered on services spanning multiple broadcasts may have the same Application Context Identifier allowing receivers with caching or persistence capabilities (see Section TBD) to maintain resources across tuning events. This allows broad flexibility in delivering resources on multiple broadcasts for related Broadcaster Applications.

# 6. BROADCASTERS APPLICATION MANAGEMENT

## 6.1 Introduction

A Broadcaster Application is a set of documents comprised of HTML5, JavaScript, CSS, XML, image and multimedia files that may be delivered separately or together within one or more packages.

This section describes, how a Broadcaster Application package is

- Downloaded,
- Signaled,
- Launched and
- Managed

Additionally, it describes how a Broadcaster Application can access the resources made available by the receiver.

Figure 6.1 diagrams the relationships between various concepts within a generalized reference receiver architecture—whether distributed, i.e., the Receiver Web Server is in a separate physical device from the User Agent, or not. It is not intended to define a particular receiver implementation but to show relationships between the various elements discussed in this section.

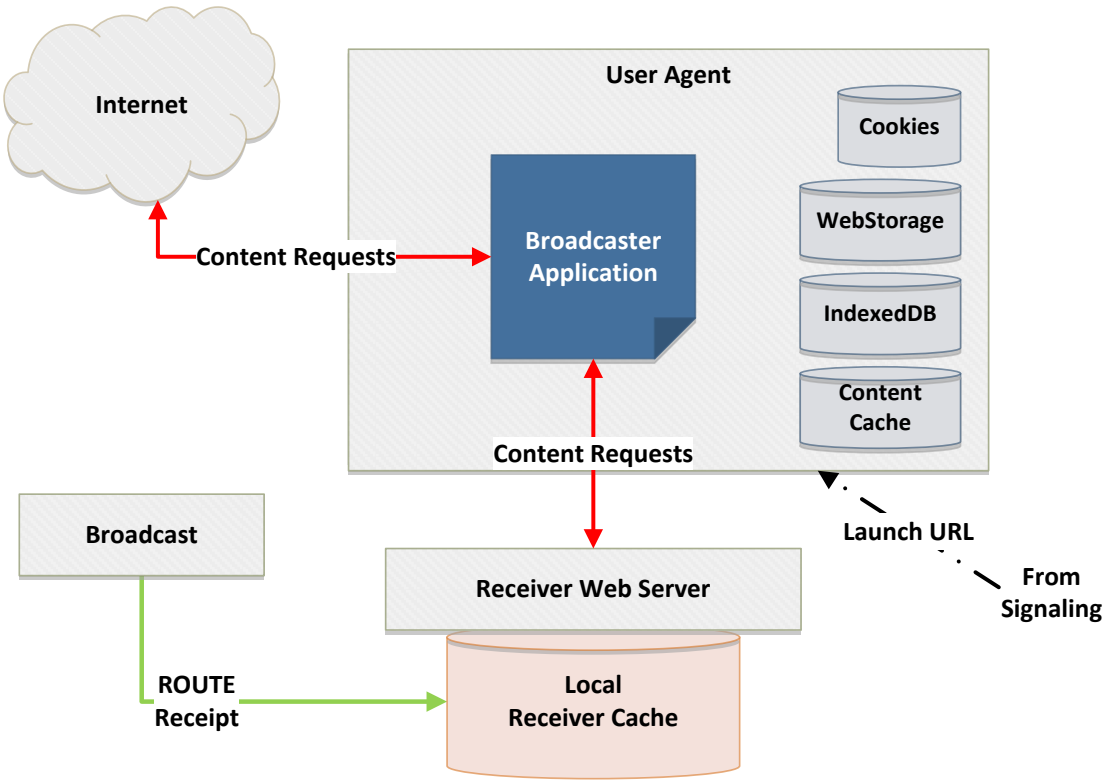


**Figure 6.1** Receiver Conceptual Architecture.

The Broadcaster Application is launched after the receiver receives application signaling information (see Section 6.3 below) and then forwards the launch URL to the User Agent, which, in turn, loads the main Broadcaster Application document from the URL. Note that the URL may

point to Internet or to the Receiver Web Server depending on how it is formatted in the service application signaling. The specific mechanism of communicating the Broadcaster Application launch URL to the User Agent is a receiver implementation detail and beyond the scope of this document.

Once the main Broadcaster Application document has been loaded, it will begin requesting content from various local or external URLs. This may be done through JavaScript or standard HTML5 href requests in the normal W3C fashion. It is assumed that any content received over broadcast via ROUTE file delivery will be saved into the Local Receiver Cache and accessed using the Receiver Web Server. This specification makes no assertions as to how this is done nor how any cache or storage is implemented. It does, however, describe how the Broadcaster Application can access the resources using URL requests to the Receiver Web Server.

Note that the User Agent supports various local W3C storage mechanisms according to Section 5.2. The User Agent may also perform internal caching of content. These are shown in Figure 6.1 as Cookies, WebStorage, IndexedDB, and Content Cache. The internal W3C-compatible storage mechanisms implemented within the User Agent should not be confused with the Local Receiver Cache shown separately in Figure 6.1. The availability and management of the internal User Agent storage areas is beyond the scope of this specification.

## 6.2 Local Receiver Cache Management

It is anticipated that receiver storage capabilities within a given broadcast area will run the gamut from receivers having no storage available for applications to receivers having effectively unlimited storage. It is presumed that a typical receiver will not be able to store all content sent to it for all services it is capable of receiving. It is also anticipated that the storage will be a limited commodity further constrained by the intended target uses and type of the receiver. Thus, the management of this storage will be a key function of the core receiver software.

Based on the above assumptions, a Broadcaster Application must view the underlying storage of the receiver environment as a cache where content and resource management are ultimately under control of the receiver as opposed to the Broadcaster Application itself. In fact, in some cases, the receiver cannot rely on the Broadcaster Application to manage various elements of even its own resources so ultimately the entire storage management must be undertaken by the receiver implementation.

Cache management is very usage-specific and this will be the case for receiver cache management as well. It is beyond the scope of this standard to specify the cache management schemes used by the receiver; however, certain expected behaviors are described below.

Receivers may provide Persistent Storage on various types of physical media (spinning disks, solid-state drives, etc.). Any Persistent Storage is deemed separate from the Local Receiver Cache storage. Management of any such Persistent Storage is considered proprietary and beyond the scope of this specification.

### 6.2.1 Local Receiver Cache Hierarchy Definition

All broadcast data is transmitted via ROUTE as files or packages of files (Section 6.4). All files and packages received for a particular Application Context ID shall be placed in a single hierarchy accessible to the Broadcast Application. How the base URI is formatted is beyond the scope of the present document as described in Section 5.3.

The choice of whether a package or file is immediately stored to the Local Receiver Cache on receipt within the ROUTE file stream or if the receiver chooses to defer storage until the

particular broadcast data element is referenced is an implementation decision and beyond the scope of the present standard. However, if the underlying Receiver Web Server cannot provide the requested content, an HTTP status code within the 400-series Client Error or 500-series Server Error will be returned indicating that some error condition occurred [32]. Typically, this will be either `404 Not Found`, `408 Request Timeout` or `504 Gateway Timeout` error, however, Broadcaster Applications should be constructed to deal with any HTTP status code when referencing resources outside of the Launch Package (6.2.2).

Similarly, the present document does not specify how frequently files and packages needed for the Broadcaster Application are transmitted nor how frequently application signaling metadata is sent. These decisions will depend on a number of factors such as what time the Broadcaster Application functionality is actually needed during the program, how quickly the receiver needs to access the Broadcaster Application after the service is selected, and the overall bandwidth needed to carry Broadcaster Application resources, to list a few. These and other factors will likely be different for every Broadcaster Application depending on its overall purpose.

The broadcaster shall be responsible for defining and managing any hierarchy below the Application Context ID root directory through use of the directory and path mechanisms, specifically the HTTP `Content-Location` header element [11], defined by the ROUTE file delivery protocol, as described in A/331 [1]. Any hierarchy below the Application Context ID level can be constructed as desired.

An example of how such a hierarchy could defined is described below. Figure 6.2 shows an example of such a hierarchy for **`AppContextID`** `A.xyz.com`.

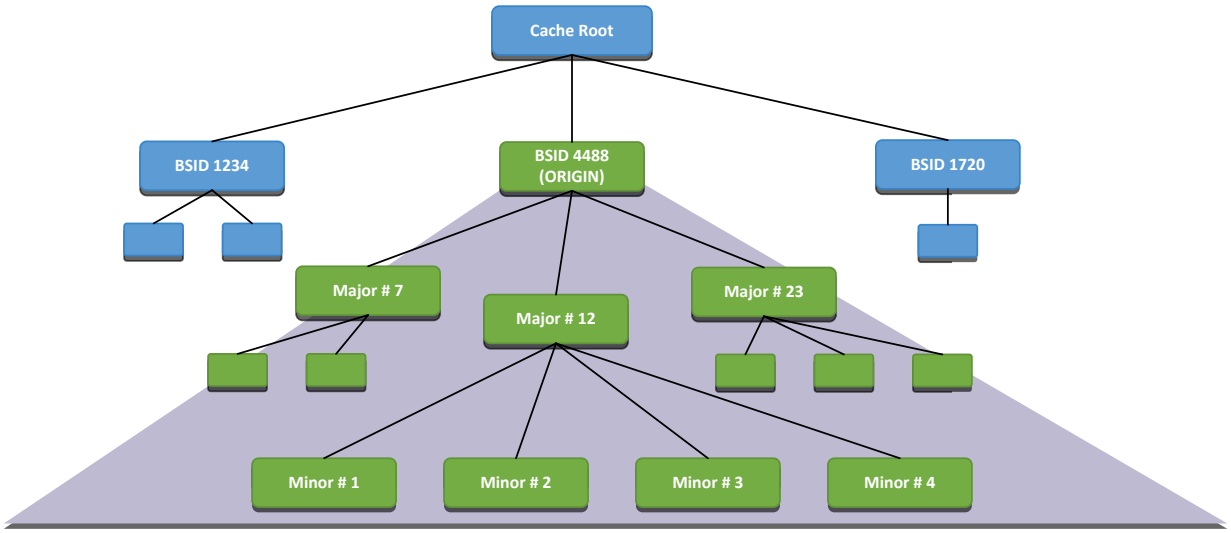


**Figure 6.2** Example local receiver cache hierachy.

As an example, presume that the broadcaster transmitted the ROUTE data such that the Broadcaster Application files are placed within the “Minor #2” directory in the hierarchy shown in Figure 6.2. Further, assume that the Broadcaster Application main document is called `"main.html"`. To launch the Broadcaster Application, the application signaling will provide the relative URI of `"12/2/main.html"` to address this main page. Note that the local path is a

convention defined by the broadcaster; it could just as easily have been called `"red/green"`. The actual URL of the Broadcast Application would be

**`<root>`**`/12/2/main.html`

Note that the immutable root is highlighted as bold in the URL. The Broadcaster Application would be able to reference any directory within its **`AppContextID`** hierarchy as designated by the green boxes in Figure 6.2.

The receiver shall not allow a Broadcaster Application to access any other areas of the Local Receiver Cache outside its **`AppContextID`** root. In effect, this means that a Broadcaster Application may only access content received through an LCT channel with which its **`AppContextID`** is associated. The receiver shall respond with a `403 Forbidden` error when a Broadcaster Application attempts to access a local URL outside of its allowed hierarchy.

### 6.2.2 Active Service Local Receiver Cache Priority

Once the packaged resources, known as the Launch Package, of the Broadcaster Application (see Section 6.3) have been received in the Local Receiver Cache, the Broadcaster Application can assume that the Launch Package resources, that is, those files delivered along with the Broadcaster Application Launch File in a single package, shall remain available as long as the receiver is tuned to the particular service. Note that if the Broadcaster Application Launch File is delivered as a separate file, it will be treated as the Launch Package without necessarily having been delivered as an explicit ROUTE multi-file package. If the receiver cannot completely store the entire Launch Package of the Broadcaster Application in the Local Receiver Cache, the Broadcaster Application shall not be launched with the User Agent. From a generic cache management standpoint, the current service is the owner of the Broadcaster Application and its associated Launch Package resources, and those particular files shall not be removed while the service (aka the current owner) is still active.

Once a file has been successfully received into the Local Receiver Cache (Section 6.2.1), it shall remain available while services with the same Application Context ID remain actively selected. Note that the receiver may have to clear other portions of the cache that are not active to accommodate files in the currently-active hierarchy. Indeed, it may be necessary for the receiver to purge the entire cache to accommodate the present active service. Since the user actively selected the current service, the receiver will assume that the current service content preempts all other content from the user perspective.

Note that a Local Receiver Cache may not be able to hold all of the content delivered over the broadcast ROUTE file delivery for the particular Application Context ID. In these cases, the Local Receiver Cache management may elect to make files unavailable, aside from the Launch Package, from the Local Receiver Cache once accessed by the User Agent. If a broadcaster desires that files should be available from the Receiver Web Server to the Broadcaster Application as long as the service is selected, the `Cache-Control` HTTP header tag shall be used. In this case, the Local Receiver Cache will continue to make the file available while the service is selected. Note that this may also result in the User Agent not caching the associated file.

An application can mark particular content as unused to hint to a receiver that the resources are no longer needed. A WebSocket API is provided (see Section 9.7) that allows files or entire directory hierarchies to be marked as unused. Subsequent access to resources that are marked as unused is indeterminate.

### 6.2.3 Cache Expiration Time

Every file delivered via broadcast using ROUTE, as described in A/331 [1], can specify an expiration time in the associated HTTP header. The `Expires` tag shall be used to define the time when the associated element becomes obsolete and can be safely removed from the cache. Receivers may elect to leave the element within the cache or purge it whenever is convenient after the expiration date and time has been passed, depending on other cache management criteria. Broadcaster Applications shall receive a `404 Not Found` error when attempting to access content that has expired. HTTP header tags and error codes are defined in RFC 7231 [11].

Setting the `Expires` HTTP tag shall indicate that the broadcaster expects the associated element to remain in the cache until the expiration time has been reached regardless of whether the service or broadcast channel is currently tuned. However, the storage requirements of the active service take precedence over the `Expires` tag, so the receiver may be forced to delete elements prior to their `Expires` time to provide storage to the current service. The Broadcaster Application must be prepared to deal with the possibility that an element may not be available.

Note that for resources that are part of the Launch Package, the Broadcaster Application can assume that those files will be available at startup and remain available as long as the Broadcaster Application is running regardless of the `Expires` time as described in Section 6.2.2.

## 6.3 Broadcaster Application Signaling

The ATSC 3.0 Receiver is responsible for retrieving the Broadcaster Application files and resources from the location and transport mechanism indicated by ATSC 3.0 application signaling. A service must be successfully selected and, for local references, the Launch Package shall be completely stored within the Local Receiver Cache (Section 6.2.1) in order for the ATSC 3.0 Receiver to launch the Broadcaster Application. For external broadband references, the URI supplied by the signaling shall be launched directly.

### 6.3.1 Broadcaster Application Launch

When a new service is selected and there is a Broadcaster Application URI present in the application signaling (see Section 6.3) for that service, there are two possible conditions to consider:

1) **No Current Broadcaster Application** – when there is no Broadcaster Application currently active in the User Agent, the receiver will launch the Broadcaster Application specified by the URI in the application signaling for the new service.
2) **Current Broadcaster Application** – when there is a Broadcaster Application previously loaded and the URI from the present service signaling matches, then the same Broadcaster Application has been requested. The current Broadcaster Application will receive a notification that a new service has been selected via the API described in Section 9.2.3. It is up to the Broadcaster Application design whether to reload its Launch Page or remain on the currently-active page.

If the Broadcaster Application URI from the newly-selected service application signaling does not match the presently-loaded Broadcaster Application URI, the new Broadcaster Application will be launched as if no previous Broadcaster Application was loaded.

### 6.3.2 Broadcaster Application Events (Static / Dynamic)

[Brief summary with reference to A/337]

## 6.4 Broadcaster Application Delivery

The file delivery mechanism of ROUTE, described in A/331 [1], provides a means for delivering a collection of files either separately or as a package over the ATSC 3.0 broadcast. The ROUTE-delivered files are made available to the User Agent via a Receiver Web Server as described in Section 6.2. The same collection of files can be made available for broadband delivery by publishing to a receiver-accessible web server. The application signaling [2] determines the source of the Broadcast Application files and packages:

1) A relative URI indicates that the source of the Broadcaster Application is broadcast ROUTE data,
2) An absolute URL indicates that the Broadcaster Application should be sourced from broadband,
3) Or all of the above.

### 6.4.1 Broadcaster Application Packages

The Broadcaster Application package is a collection of files such as HTML5, CSS, JavaScript, and image files, comprising a set of interactive functionality. The Launch File is an HTML5 document that then directly or indirectly refers to all other files of the Broadcaster Application. This Launch File will be referenced by the application signaling (see Section 6.3) and will be the file loaded into the User Agent first to launch the Broadcaster Application.

Note that it is not required that all resources used by the Broadcaster Application be delivered in a single ROUTE package over broadcast. The broadcaster may choose to send a relatively small Launch File which then performs a bootstrapping operation to determine what other resources have been delivered or are accessible via the broadcast delivery path and, in turn, which resources need to be obtained using broadband requests. Since the Launch Package containing the Launch File shall be received in its entirety before launching the Broadcaster Application (Section 6.2.2), the Broadcaster Application can forgo any checks for basic resources and perhaps speed the initial startup time. It is conceivable that Broadcaster Applications may have incremental features based on the availability of resources on the receiver. In other words, the Broadcaster Application may add features and functions as more resources are available.

In addition, the Broadcaster Application may request resources and content from or perform other activities with broadband web servers making the Broadcaster Application a true Web Application in the traditional sense. A Broadcaster Application should be aware that all receivers may not contain sufficient storage for all the necessary resources or may not have a broadband connection and should deal with these situations accordingly.

### 6.4.2 Broadcaster Application Package Changes

Broadcaster Application resource files and packages may be updated at any time. The broadcaster may send an Event Stream notification to let the Broadcaster Application know that something has been changed. The Broadcaster Application determines how such changes should be addressed based on the Event Stream notification.

## 6.5 Security Considerations

TBD

## 6.6 Companion Device Interactions

TBD [Brief overview with reference to A/338]

# 7. MEDIA PLAYER

In the ATSC 3.0 receiver environment, there are two software components that are capable of collecting media segments from either broadcast or broadband and forwarding them to the receiver's media codec. For the purposes of this specification, they are referred to as Application Media Player (AMP) and Receiver Media Player (RMP) and described further in this section. The AMP is JavaScript code (e.g. DASH.js), which is part of the HTML5 application, while the RMP is receiver-specific software or hardware. Details of the RMP design and implementation are beyond the scope of this specification.

## 7.1 Receiver Media Player

The Receiver Media Player (RMP) may use any appropriate mechanism to retrieve media content and stream it to the receiver's decoders. The media content can be fetched or streamed from either broadcast or broadband. Either RMP or AMP can be utilized to play media content streamed over broadcast or broadband, but ultimately, it is the receiver's responsible to decode and render the media content to the display, regardless of which media player is used or from where the content is originated.

In some conditions and use cases, a Broadcaster Application may request receiver to use the RMP to play media content using the Receiver WebSocket Server APIs, as defined by this specification.

## 7.2 Application Media Player

The Application Media Player (AMP) uses W3C-standardized technologies such as Media Source Extensions (MSE) and the `HTMLMediaElement` (`<video>`) to render DASH media content (segments or sub-segments), which can be downloaded from broadband or broadcast.

In this discussion, "streamed" means the file playback can commence while the full media content is being retrieved. The term "fetched" refers to the retrieval of a single file (which might be a full mp4 media file or DASH Media Segment).

The AMP can be utilized to access any of the following media content types:

- Media content downloaded from broadcast and streamed by the receiver to the AMP
- Media content downloaded from broadband and streamed by the receiver to the AMP
- Media content streamed from broadcast (i.e., live streaming) and streamed by the receiver to the AMP

The AMP can retrieve the DASH media content via either a push model, utilizing Receiver WebSocket Server APIs defined in this specification, or a pull model, utilizing W3C APIs such as XHR, and sends the collected segments to the receiver media codec, as described in subsections below. The AMP may optionally use W3C-standarized technologies such as Encrypted Media Extensions (EME) to provide a license to a CDM in the receiver to enable decryption of the retrieved segments.

Since either a Receiver Media Player or an Application Media Player can collect and render the live broadcast media content, service layer signaling indicates whether application is intended to playback the media content, so the receiver will not use its own player, upon a service selection. If the application is required for rendering of live media content, it is recommended that the application appear sufficiently frequently in the broadcast emission such that it can be accessed in a timely manner upon service acquisition. Further details of the broadcast emission rate of the Broadcaster Application are out of scope of this specification.

### 7.2.1 Pull-Model Broadcast Media Streaming

In the Pull-Model Broadcast Media Streaming, a Broadcaster Application pulls broadcast media segments by leveraging an MPD and a JavaScript instantiation of a DASH client (e.g. DASH.js – see https://github.com/Dash-Industry-Forum/dash.js/wiki). DASH client design and implementation details are beyond the scope of this specification.

The Broadcaster Application can retrieve the MPD file using a URL discovered by the Query MPD URL WebSocket API specified in Section 9.1.7.

The MPD should only include segment addresses that can be accessed by the DASH client. This may require receiver overwriting of addresses in the MPD sent as part of the broadcast emission to point to segments available locally. For instance, a receiver that can fetch content via broadcast may perform the following:

1) Download the MPD from the broadcast emission
2) Inspect the MPD for segment access
3) Determine the best method for downloading the segment (broadcast or broadband) and download the segment
4) Overwrite the corresponding MPD address with the local address of the segment and make it available at the location indicated in the API

The receiver is expected to allow the application only to access the live media content associated with the Service from which it was launched.

### 7.2.2 Push-Model Media Streaming

As described in the previous section, the Broadcaster Application may attempt to open one of the binary WebSocket interfaces. If the connection is successfully opened, the broadcast receiver may push media as it is received leveraging the WebSocket binary data transmission capabilities (see https://www.w3.org/TR/2011/WD-websockets-20110929/#feedback-from-the-protocol).

The Broadcaster Application can access the broadcast MPD through the Query MPD URL API described in Section 9.1.7. It is not required for the application to have access to the MPD prior to playback of media blobs. However, the MPD may be leveraged by the application for track or ad insertion. The application may also request the broadcast receiver to retrieve specific segments on its behalf (based on information in the MPD) using the Media Segment Get command (Section 8.4.5) in order to pre-fetch segments such as advertisements that can be inserted at the appropriate time relative to the live media.

## 7.3 Receiver Media Player

Receiver Media Player (RMP) may use any mechanism appropriate for their platform to retrieve media content and stream them to the media codec. Similar to the AMP, the media content can be downloaded or streamed from either broadcast or broadband. Since either RMP or AMP can be utilized to render the DASH content, the receiver implementation relies on the service layer signaling to indicate which media player to use. The receiver is ultimately responsible for decoding and rendering the media content to the display, regardless of which media player is used or from where the content is originated.

In some conditions and use cases, a Broadcaster Application may request to use the RMP to play back media content using the Receiver WebSocket Server APIs, as defined by this specification.

## 7.4 Media Content Access

Either the AMP or the RMP can be utilized to access any of these types of content:

- Media content downloaded from broadcast and streamed by the receiver to the AMP
- Media content downloaded from broadband and streamed by the receiver to the AMP;
- Media content fetched from broadband and then streamed by the receiver to the AMP;
- Media content streamed from broadband to the AMP directly;
- Media content streamed from broadcast (i.e. live streaming) and delivered to the AMP directly.

### 7.4.1 Streaming Broadcast Media Content

A broadcaster application may wish to utilize either the AMP or RMP to play media content sent over broadcast. For app-enhanced services (specified as **`SLT.Service`**`@serviceCategory="1"` in the A/331 [1] SLT), a flag in the service layer signaling indicates whether to use the AMP or RMP to stream the broadcast content. Receiver implementation uses the value in this flag to determine whether to use its own player to stream and decode the live broadcast media content or to provide the segments to the broadcaster's AMP associated with the service to render, as described in the previous section utilizing either the push or pull model. If the service does not have any associated Broadcaster Application but the flag in the service indicates that the AMP must be used, the broadcast media content stream is not viewable and the RMP cannot be used to playback the broadcast streaming content.

#### 7.4.1.1 Streaming Broadcast Media Content Utilizing RMP

When the flag in the service signaling indicates RMP is to be used to play broadcast media content, the receiver software receives the media segments from the broadcast and utilizes its own player to stream and decode the broadcast media content. Figure 7.1 shows the interaction between the logical components of the system. As it is shown in this diagram, the AMP has no role or responsibility in the streaming of such broadcast media content.

The receiver implementation provides the necessary mechanism for a viewer to select different audio/video tracks and enable or disable closed captions. Details of how a Receiver Media Player implements such features and exposes the necessary user interface to the viewer is out of the scope of this specification.

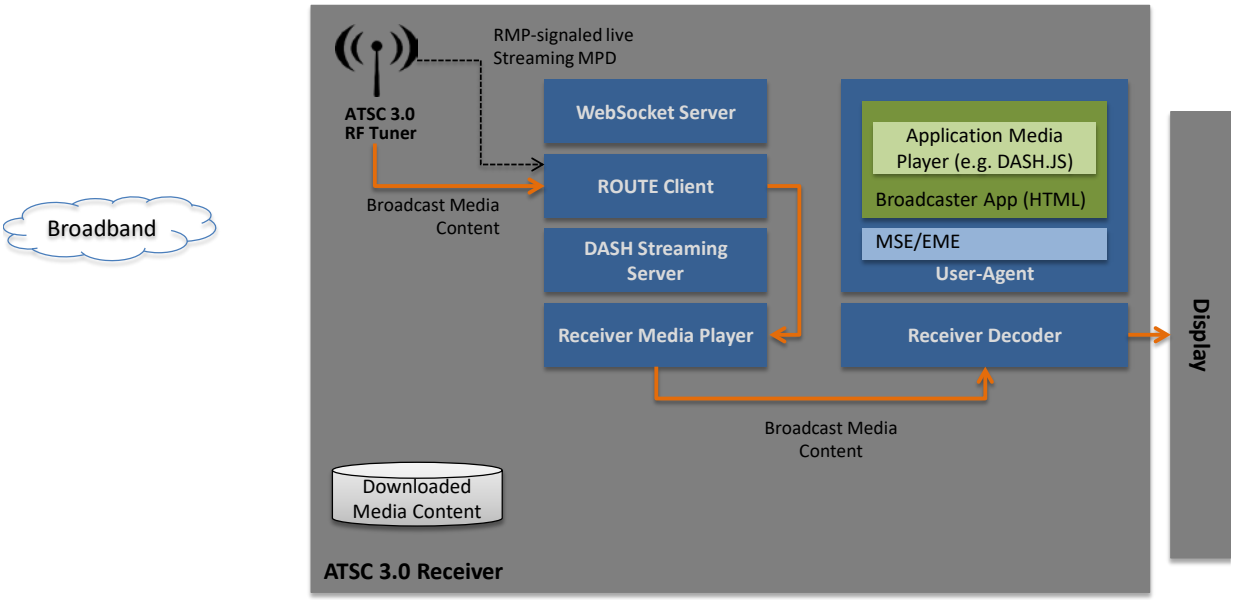


**Figure 7.1** Using the RMP to render live broadcast streams.

#### 7.4.1.2 Streaming Broadcast Media Content Utilizing AMP

The **`HELD.HTMLEntryPage`**`@linearSvcEnabling` flag in the service signaling [1] indicates, when true, that the broadcaster requests that the AMP to be used to render the broadcast streaming media content associated with the Service. If the receiver implementation supports AMP playback of broadcast media content, it provides the broadcast stream media segments to the application to play the stream.

As described previously, the APIs provide two mechanisms to allow the broadcast segments to be sent to the Broadcaster Application for playback: (1) pull mode, using standard methods such as XML HTTP Request (XHR), and (2) push model, using the Receiver WebSocket Server interface.

In the pull model, the Broadcaster Application provides a DASH client streaming media player. The receiver provides the MPD at a well-known URL for the application to access and the receiver collects the broadcast media segments from the RF tuner and provides the segments as requested by the client media player, as signaled in the MPD. The MPD provided by the receiver will provide the URL to the locally cached segments.

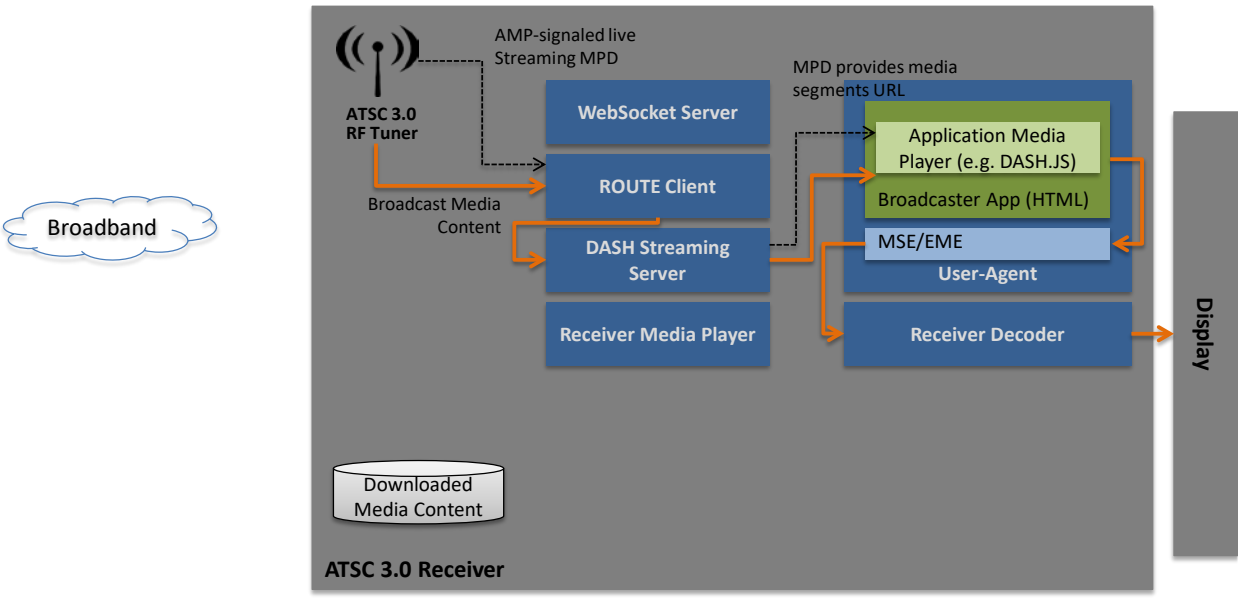


**Figure 7.2** AMP live streaming using pull model.

In the push model, the Broadcaster Application provides a streaming media player (e.g. AMP). Receivers that support AMP playback of broadcast media content provide broadcast media segments collected from the RF tuner via a WebSocket API.

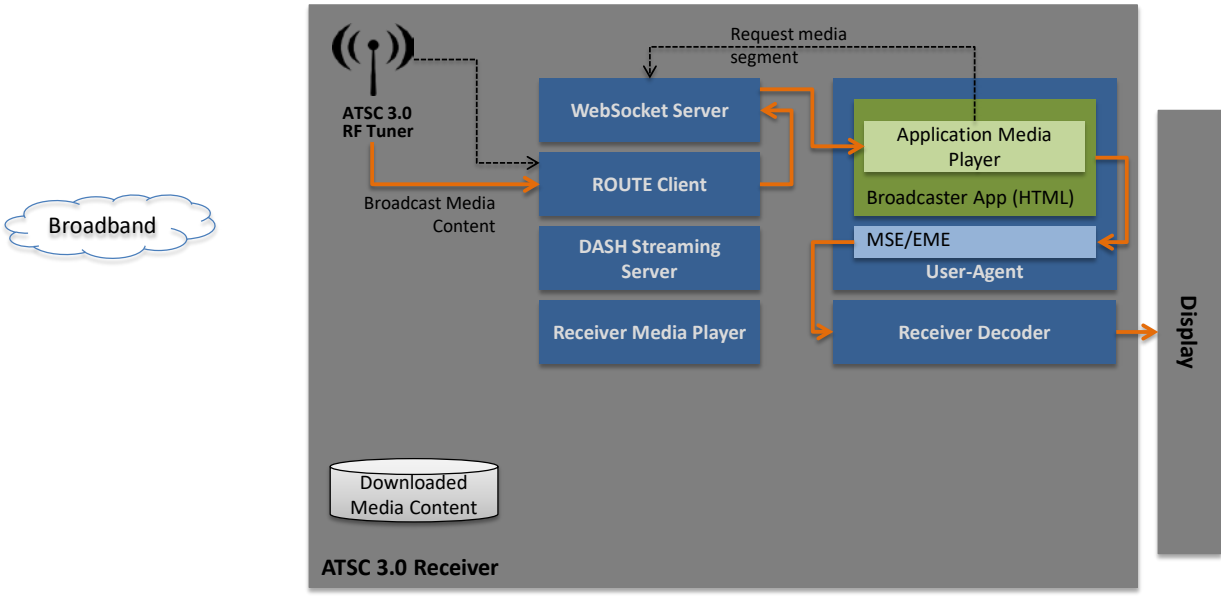


**Figure 7.3** AMP live streaming using push model.

### 7.4.2 Streaming Broadband Media Content

In the case where the media content is delivered over broadband, the Broadcaster Application may either request the receiver to play the content using RMP, or it may use its own AMP to play the media.

In the case where the Broadcaster Application wishes to use its own AMP, it collects the segments and processes it using MSE/ME. The following diagram shows this interaction. In this case, the RMP is not involved, as shown below:

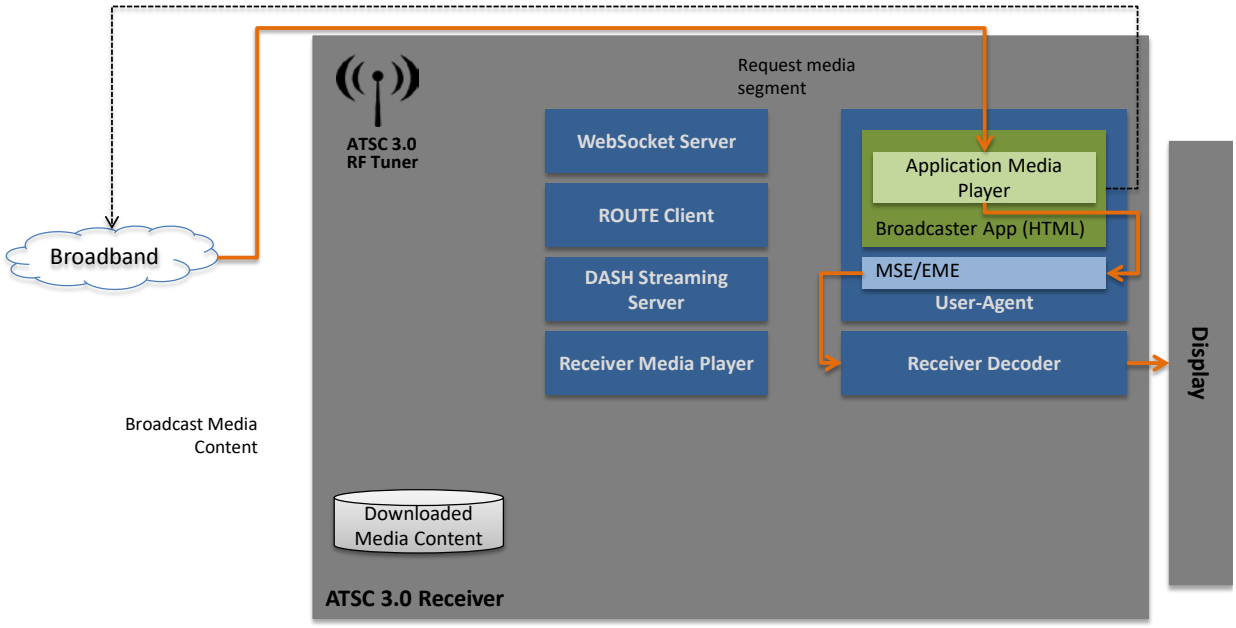


**Figure 7.4** Streaming broadband media content utilizing AMP.

In the case where the Broadcaster Application wishes to use the RMP to stream a broadband media-streaming source, the Broadcaster Application uses Receiver WebSocket Server API to send the request to RMP to play the content using the Set RMP URL API defined in Section 9.4.6.

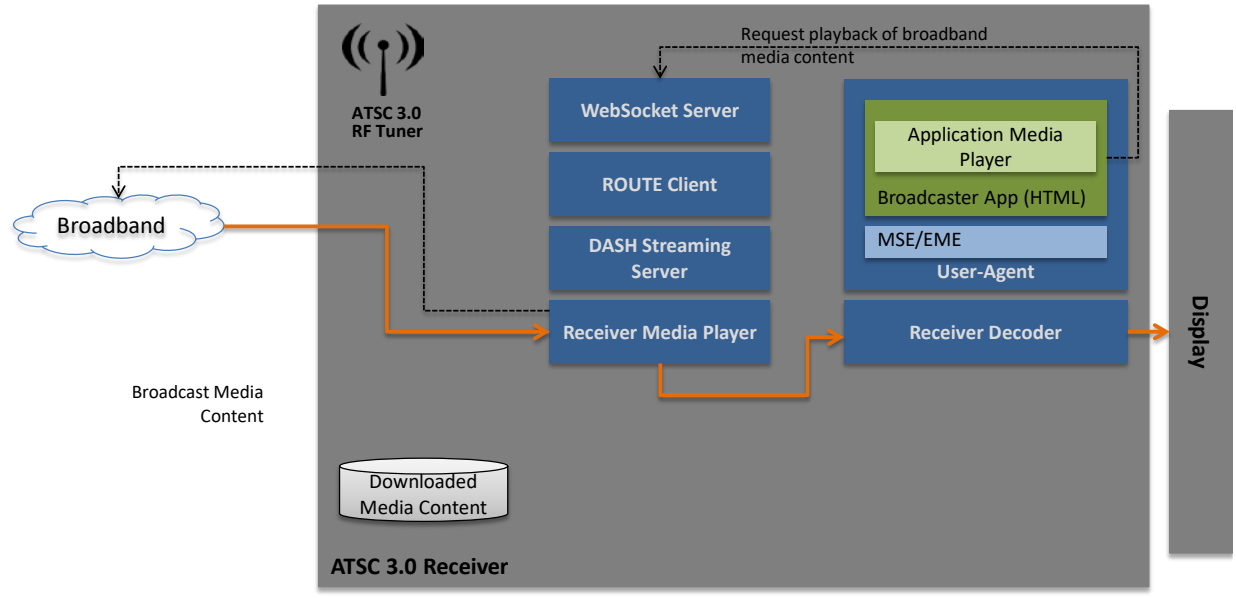


**Figure 7.5** Streaming broadband media content utilizing RMP.

### 7.4.3 Streaming Downloaded Media Content

A Broadcaster Application may wish to utilize either the AMP or the RMP to play media content that has been downloaded from either broadcast or broadband. The details of how media content is downloaded from broadband or broadcast are described in Section TBD.

#### 7.4.3.1 Playback of Downloaded Media Content Utilizing RMP

Once media content is downloaded and stored, the application may request the RMP to play the downloaded media content. The Broadcaster Application utilizes Receiver WebSocket Server APIs to request the playback of the media content using the RMP.

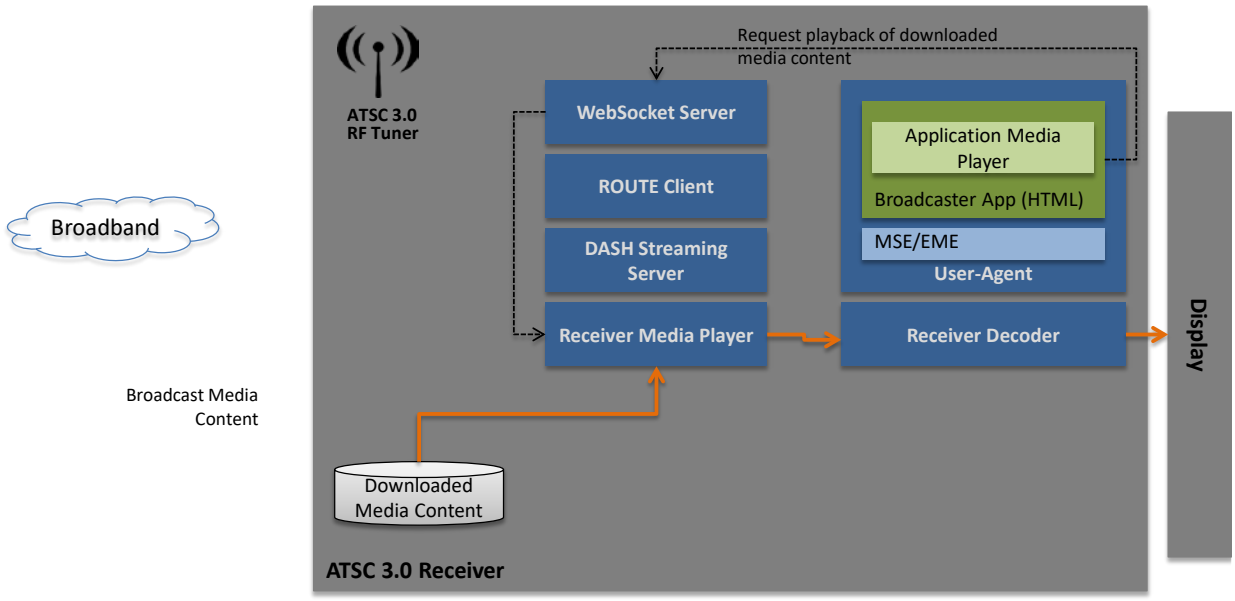


**Figure 7.6** Playback downloaded media content utilizing RMP.

#### 7.4.3.2 Playback of Downloaded Media Content Utilizing AMP

As described previously, two mechanisms are specified to allow the downloaded media content segments to be sent to the Broadcaster Application for playback: (1) pull-mode, using standard methods such as XML-over-HTTP-Request (XHR), and (2) push-mode, using Receiver WebSocket Server APIs. All receivers are expected to support pull mode operation.

Similar to playing the broadcast media content, in the pull model, the Broadcaster Application provides a DASH client streaming media player along with its application. The receiver provides the URL referencing the broadcast MPD for the application to access using the API specified in Section 9.1.7. The Broadcaster Application retrieves and parses the MPD to identify the URLs of desired Media Segments, and receiver makes appropriate requests for these files to the receiver. The receiver sends media segments locally cached the requested Media Segments retrieved from Persistent Storage, as requested by the client media player, as signaled in the MPD. The MPD will provide the URL to the locally cached segments.

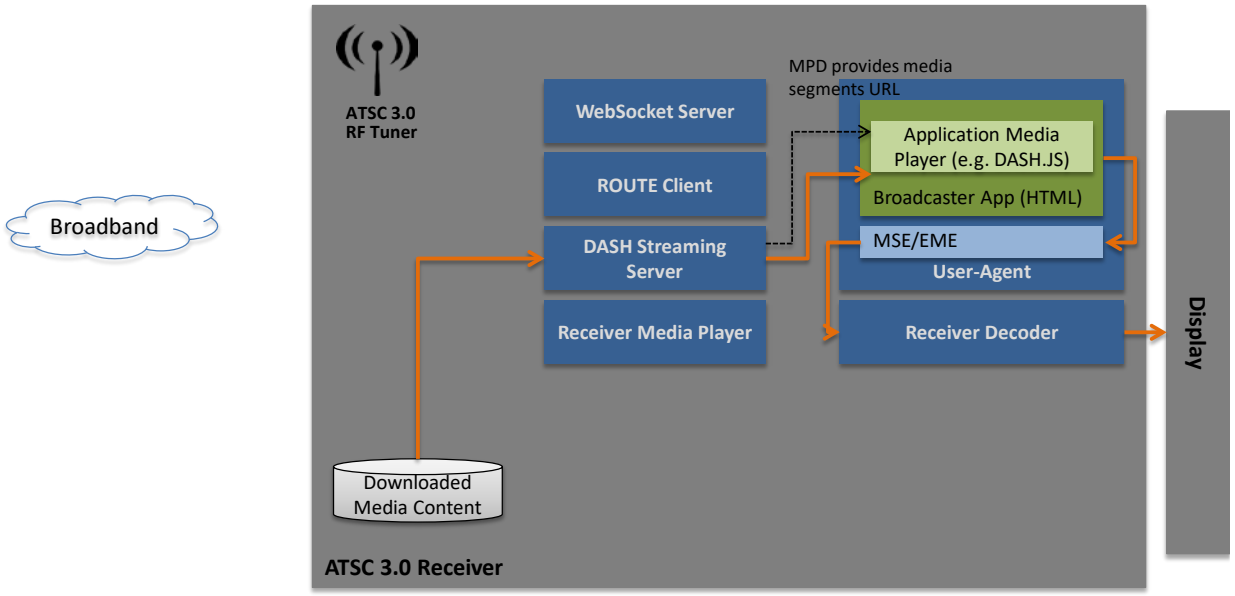


**Figure 7.7** Streaming downloaded media content using pull model.

In the push model, the Broadcaster Application provides a streaming media player along with its application (the AMP). The receiver provides the downloaded media segments via its WebSocket Server interface.

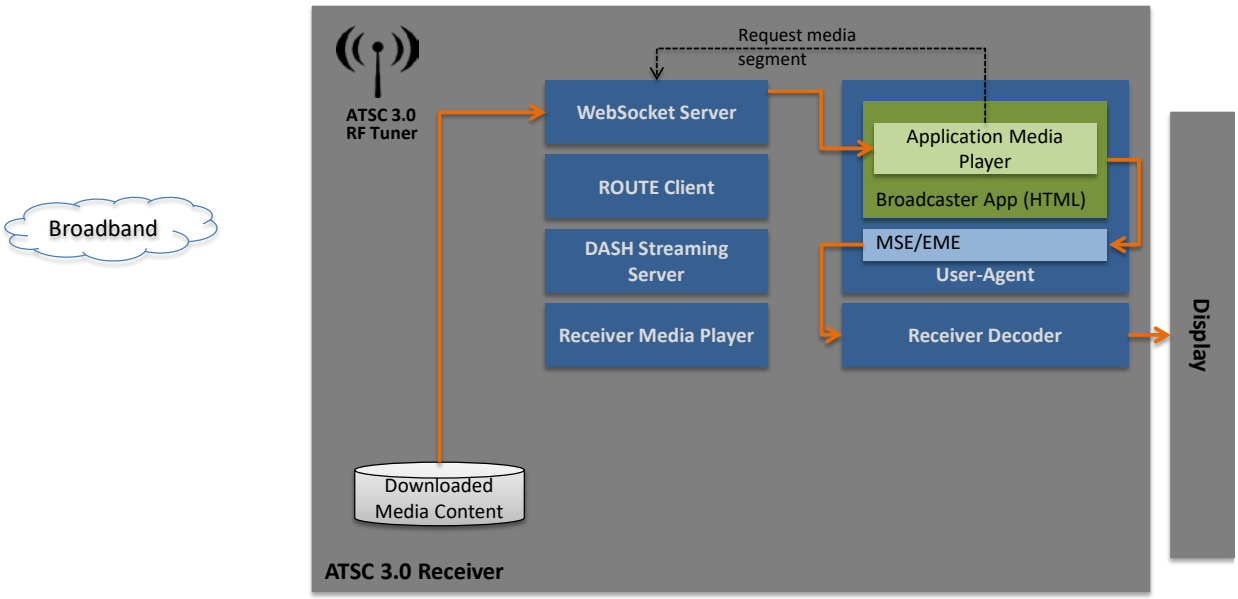


**Figure 7.8** Streaming downloaded media content using push model.

## 7.5 Downloading Media Content from Broadcast or Broadband

Media content may be downloaded from either broadcast or broadband and can be played utilizing either AMP or RMP as described previously.

> Note: The difference between downloaded content and streaming content is that, in the download case, the entire media content file is downloaded and made

available to the player of the choice, whereas, in the streaming case, play out begins as soon as possible before all of the content is available.

In order to download media content over broadcast, the Broadcaster Application may request the User Agent to download media content using the standardized W3C APIs. The use agent then downloads and caches the media content to be played back as described in the previous section.

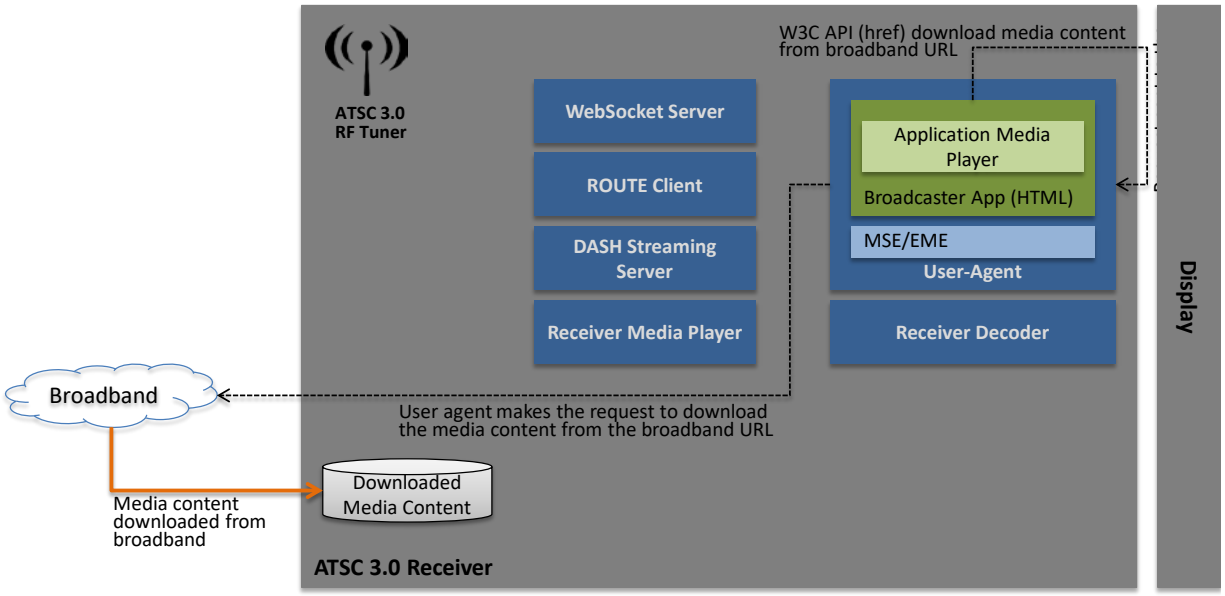


**Figure 7.9** Downloading media content from broadband.

Media content sent over broadcast can also be downloaded via ROUTE. A media content file, if it is signaled in ROUTE FDT associated with the Service, can be downloaded with no intervention from the application; in other words, if a media content file is present, the media content is downloaded regardless of any explicit request from the Broadcasters Application. Similar to the downloaded files from broadband, it is then made available for playback, as described in the previous section.

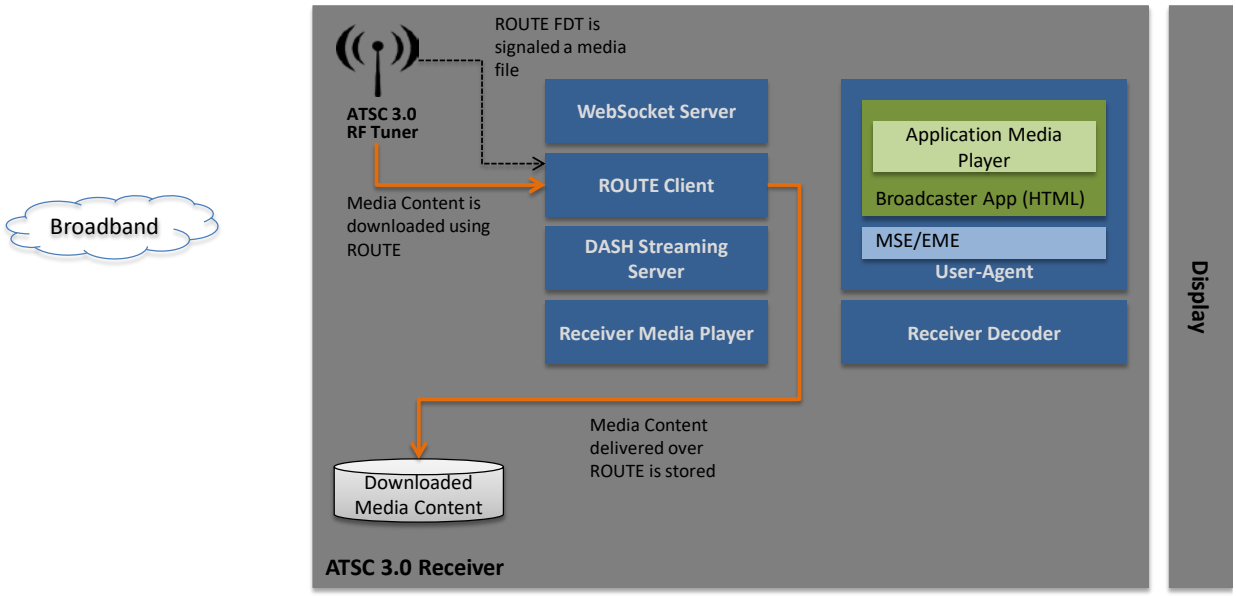


**Figure 7.10** Downloading media content from broadcast.

The receiver adheres to all caching hints as provided in the HTTP header whether the file is sent over broadband or broadcast.

# 8. ATSC 3.0 WEB INTERFACES

## 8.1 Introduction

A Broadcaster Application on the receiver may wish to exchange information with the receiver implementation to:

- Retrieve user settings
- Send an event from receiver to the application
  - Notification of change in a user setting
  - DASH-style Event Stream event (from broadcaster)
- Request receiver actions

In order to support these functions, the receiver includes a web server and exposes a set of WebSocket RPC calls. These RPC calls can be used to exchange information between an application running on the receiver and the receiver implementation. Figure 8.1 shows the interaction between these components.

In the case of a centralized receiver architecture, the web server typically can be accessed only from within the receiver by applications in the application execution environment (user agent).

In the case of many distributed receiver architectures, the web server can be accessed from outside the “gateway” component of the receiver, but the intent is to service requests only from the client component of the receiver.

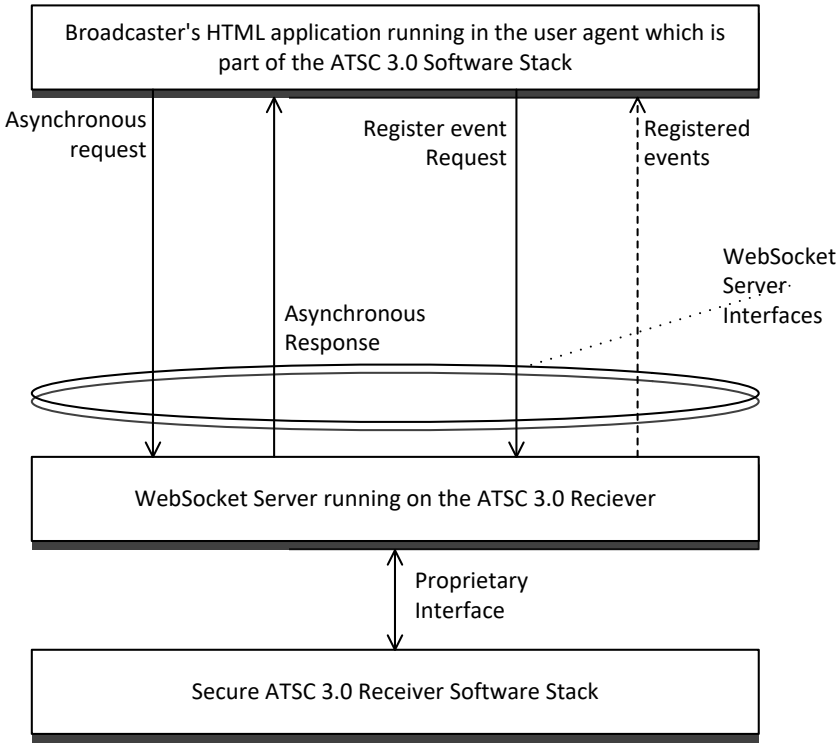


**Figure 8.1** Communication with ATSC 3.0 receiver.

One or more ATSC 3.0 WebSocket interfaces are exposed by the receiver. All receivers support a WebSocket interface used for command and control. Some receivers also support three additional WebSocket interfaces, one each for video, audio, and caption binary data. The Broadcaster Application or companion devices can connect to the command and control interface to retrieve state and settings from the receiver and perform actions, such as change channels.

## 8.2 Interface binding

Since the APIs described here utilize a WebSocket interface, the Broadcaster Application can rely on standard browser functionality to open the connection and no specific functionality needs to be present in the application. For a companion device, support is dependent on the operating system, but most modern mobile operating systems such as iOS and Android provide native support for WebSockets.

In order to communicate with the WebSocket server provided by the receiver, the broadcaster application needs to know the URL of the WebSocket server. The WebSocket server location may be different depending on the network topology (e.g., integrated vs. distributed architecture), or it may be different depending on the receiver implementation. In order to hide these differences from the broadcaster application, the Broadcaster Application is launched with a query term parameter providing information regarding the location of the WebSocket server in the receiver.

When an entry page of a Broadcast Application is loaded on the user agent, the URL shall include a query term providing the base URI of the ATSC 3.0 WebSocket Interface supported by receivers. Using the ABNF syntax, the query component shall be as defined below:

```
query =  "wsURL=" ws-url
```

The `ws-url` is the base WebSocket URI and shall be as defined in RFC 6455 [13].

The following shows an example of how such a query string can be used in the Broadcaster Application. In this example, if the launch page URL is

```
http:://localhost/xbc.org/x.y.z/home.html
```

the Broadcaster Application is launched as follows:

```
http:://localhost
/xbc.org/x.y.z/home.html?wsURL=wss://localhost:8000
```

The `wsURL` query parameter is added to load an entry page URL of a broadcast-delivered application. It is expected that a broadband web server would ignore a `wsURL` query parameter in the URL of an HTTP request if it were to appear.

The following shows sample JavaScript illustrating how the `wsURL` parameter can be extracted from the query string:

```
function getWSurl () {
    var params = window.location.search.substring(1);
    var result = 'ws://localhost:8080';// Default value if desired
    params.split("&").some(function (part) {
        var item = part.split("=");
        if (item[0] == 'wsURL') {
            result = decodeURIComponent(item[1]);
            return true;
        }
    })
    return result;                    // Returns 'wss://localhost:8000'
}
```

Once the URL of the WebSocket server is discovered in this way, it can be used to open a connection to a WebSocket.

### 8.2.1 WebSocket Servers

All receivers shall support access to a WebSocket interface used for communication of the APIs described in Section 9. Receivers which support push-mode delivery of binary media data (video, audio, and captions) also support three additional WebSocket interfaces, one for each type of media data. Table 8.2 describes the four interfaces. In the table, the term "*WSPath*" represents the value of the `wsURL` parameter discovered in the procedure above.

**Table 8.1** WebSocket Server Functions and URLs

| WebSocket Interface Function | URL | Receiver Support |
|---|---|---|
| Command and Control | *WSPath*/`atscCmd` | Required |
| Video | *WSPath*/`atscVid` | Optional |
| Audio | *WSPath*/`atscAud` | Optional |
| Captions | *WSPath*/`atscCap` | Optional |

The following shows sample code that implements a connection to the command and control WebSocket server using the value passed in the `wsURL` parameter and the `getWSurl()` function described above:

```
function setWebSocketConnection () {
    var wsURL = getWSurl()+'/atscCmd';
    myCmdSocket = new WebSocket(wsURL);
        // New WebSocket is created with URL = 'wss://localhost:8000/atscCmd'
    myCmdSocket.onopen ...
}
```

In the push model, each MPEG DASH Media Segment file delivered via a Video/Audio/Captions WebSocket interface is delivered in a binary frame of the WebSocket protocol. The command and control interface uses text frame delivery.

For more information on WebSocket within the browser please see the W3C specification: http://www.w3.org/TR/websockets/.

**Table 8.2** API Applicability

| WebSocket APIs | Reference | Applicability |
|---|---|---|
| Receiver Query APIs | Section 9.1 | Always |
| Asynchronous Notifications of Changes | Section 9.1.8 | Always |
| Event Stream APIs | Section 9.3 | RMP |
| Acquire Service API | Section 9.4.1 | Always |
| Video Scaling and Positioning API | Section 9.4.2 | RMP |
| Media Blob Event | Section 1.1.1 | AMP |
| XLink Resolution API | Section 9.4.3 | RMP |
| Subscribe MPD Changes API | Section 9.4.4 | AMP |
| Unsubscribe MPD Changes API | Section 9.4.5 | AMP |
| Set RMP URL API | Section 9.4.6 | RMP |
| Media Track Selection API | Section 9.4.7 | RMP |
| Media Segment Get API | Section 9.6 | AMP |
| Mark Unused API | Section 9.7 | Always |

## 8.3 Data Binding

Once the connection is established through receiver WebSocket command and control Server, messages can be sent and received. However, since the WebSocket Server is just a plain bidirectional socket with no structure other than message framing, a message format needs to be defined. This section defines the basic formatting of messages, and the following section defines the specific messages that are supported.

The data structure of the WebSocket interface for command and control is based on JSON-RPC version 2.0 [30]. JSON RPC provides RPC (remote procedure call) style messaging, including unidirectional notifications and well-defined error handling using the JavaScript Object Notation (JSON) data structure. For more information on JSON RPC, please see: http://www.jsonrpc.org/specification [30].

The data is always sent as UTF-8 stringified JSON object. The receiver shall parse the JSON object and route the method to the right handler for further processing. Several types of data messages are defined for the command and control WebSocket interface:

- Request message – used to request information or initiate an action
- Synchronous response – a definitive answer to a request provided immediately
- Asynchronous response – a definitive answer to the request provided asynchronously
- Error response – a definitive error to the request provided
- Notification – unidirectional notification, no synchronous or asynchronous response is expected

The other three WebSocket interfaces used for delivery of binary data from the receiver to the Broadcaster Application.

The notation used to describe the flow of data in this specification shall be as follows:

```
--> data sent to Receiver
<-- data sent to Broadcaster Application
```

Note: The interface is bidirectional, so requests, responses and notifications can be initiated by both the receiver and the Broadcaster Application.

Request/response example:

```
--> {"jsonrpc": "2.0", "method": "exampleMethod1", "params": 1, "id": 1}

<-- {"jsonrpc": "2.0", "result": 1, "id": 1}
```

Notification example:

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
```

Error example:

```
--> {"jsonrpc": "2.0", "method": "faultyMethod", "params": 1, "id": 6 }

<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not
found"}, "id": 6}
```

### 8.3.1 Error handling

JSON-RPC v2 defines a set of reserved error codes as shown in Table 8.3.

**Table 8.3** JSON RPC Reserved Error Codes

| Code | Message | Meaning |
|---|---|---|
| -32700 | Parse error | Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text. |
| -32600 | Invalid Request | The JSON sent is not a valid Request object. |
| -32601 | Method not found | The method does not exist / is not available. |
| -32602 | Invalid params | Invalid method parameter(s). |
| -32603 | Internal error | Internal JSON-RPC error. |
| -32000 to -32099 | Server error | Reserved for implementation-defined server-errors. |

ATSC-defined error codes from the receiver shall be as defined in Table 8.4.

**Table 8.4** JSON RPC ATSC Error Codes

| Code | Message | Meaning |
|---|---|---|
| -1 | Unauthorized | Request cannot be honored due to domain restrictions. |
| -2 | Not enough resources | No resources available to honor the request. |
| -3 | System in standby | System is in standby. Request cannot be honored. |
| -4 | Content not found | Requested content cannot be found. For example, invalid URL. |
| -5 | No broadband connection | No broadband connection available to honor the request |
| -6 | Service not found | The requested Service cannot be located. |
| -7 | Service not authorized | The requested Service was acquired but is not authorized for viewing due to conditional access restrictions. |
| -8 | Video scaling/position failed | The request to scale and/or position the video did not succeed. |
| -9 | XLink cannot be resolved | The request to resolve an XLink has failed. |
| -10 | Track cannot be selected | The media track identified in the Media Track Selection API cannot be found or selected. |
| -11 | The indicated MPD cannot be accessed | In response to the Set MPD URL API, the MPD referenced in the URL provided cannot be accessed. |
| -12 | The content cannot be played | In response to the Set MPD URL API, the requested content cannot be played. |
| -13 | The requested offset cannot be reached | In response to the Set MPD URL API, the offset indicated cannot be reached (e.g. beyond the end of the file). |

## 9. SUPPORTED METHODS

This chapter describes the methods that are supported on the command and control WebSocket Interface. These APIs are based on JSON-RPC 2.0 over WebSockets as described in Section 7. See above chapters for more information on the interface and data binding. All methods are in a reverse domain notation separated with a dot ".". All ATSC methods that are available over the interface are prefixed with "`org.atsc`", leaving room for other methods to be defined for new receiver APIs in the future.

## 9.1 Receiver Query APIs

The receiver software stack exposes a set of WebSocket APIs to the application to retrieve user settings and information, as described in the following sections.

If these settings are not available by the receiver, the Broadcaster Application may use default values based on its own business policy and logic. An application may choose to provide its own settings user interface and store the collected setting by cookies on the receiver. The application may use these settings to change the behavior of the running application.

The following settings and information may be retrieved by a Broadcaster Application:

- Content Advisory Rating setting
- State of Closed Caption display (enabled/disabled)
- Current Service ID
- Language preferences (Audio, User Interface, Captions, etc.)
- Closed Caption Display Preferences (font sizes, styles, colors, etc.)
- Audio Accessibility preferences
- A URL the Broadcaster Application can use to fetch the current broadcast MPD
- •Receiver Web Server URI

The following APIs are defined to allow Broadcaster Applications to retrieve these settings and information.

### 9.1.1 Query Content Advisory Rating API

Broadcaster Applications may wish to know the highest content advisory rating the viewer has unlocked on a receiver, in order to decide what applications, links or text within a page to make available to the user. For example, the application can use this rating to decide whether a description of a program should be presented to the viewer. If the rating is changed on the receiver, the receiver shall send an event to the application, indicating so, if the receiver supports this setting. If the receiver does not make this information available, the application may choose a default value based on its own business logic and policy.

The Query Content Advisory Level API shall be defined as follows:

method: "`org.atsc.query.ratingLevel`"

params: Omitted

Response:

result: a JSON object containing a `rating` key/value pair

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "rating": {"type": "string"}
   },
   "required": ["rating"]
}
```

`Rating` – This required string shall provide the content advisory rating in string format, as defined in A/331 [1], Section 7.3.

For example, consider the case that the rating setting is TV-PG-D-L for the US Rating Region 1. The application can make a request:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.ratingLevel",
    "id": 37
}
```

The Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"rating": "1,'TV-PG-D-L', {0 'TV PG'}{1 'D'}{2 'L'}"},
    "id": 37
}
```

### 9.1.2 Query Closed Captions Enabled/Disabled API

The Broadcaster Application may wish to know whether the user has turned on closed captions, in order to display its application graphics at a different location than where the closed caption is typically presented. The application requests the closed caption setting from the receiver via Receiver WebSocket Server interface.

The receiver sends an event when the user enables or disabled the closed captioning to the running application.

The Query Closed Captions Enabled/Disabled API shall be defined as follows:

method: "`org.atsc.query.cc`"

params: Omitted

Response:

result: a JSON object containing a `ccEnabled` key/value pair.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "ccEnabled": {"type": "boolean"}
   },
   "required": ["ccEnabled"]
}
```

`ccEnabled` – This required Boolean shall indicate true if closed captions are currently enabled by the user and false otherwise

For example, if closed captions are currently enabled:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.cc",
    "id": 49
}
```

The Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"ccEnabled": true},
    "id": 49
}
```

### 9.1.3 Query Service ID API

Since the same application may be used for multiple services within the same broadcast family, the application may wish know which exact service has initiated the application. This allows the application to adjust its user interface and provide additional features that might be available on one service vs. another.

The Query Service ID API shall be defined as follows:

method: "`org.atsc.query.service`"

params: Omitted

Response:

result: a JSON object containing a `currentSvc` key/value pair.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "properties": {"currentSvc": {
         "type": "string",
         "format": "uri"
      }},
      "required": ["currentSvc"]
   }
}
```

`currentSvc` – This required key shall indicate the globally unique Service ID associated with the currently selected service as given in the USBD in **`bundleDescription.userServiceDescription`**`@globalServiceID`. See A/331 [1] Sections 7.1.3 (ROUTE/DASH) and 7.2.1 (MMT).

For example, if the globally unique Service ID for the currently selected services is "`http://xbc.tv/wxbc-4.2`", and the application issues a request to the terminal:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.service",
    "id": 55
}
```

The Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"currentSvc": "http://xbc.tv/wxbc-4.2"},
    "id": 55
}
```

### 9.1.4 Query Language Preferences API

Broadcaster Application may wish to know the language settings in the Receiver, including the language selected for audio output, user interface displays, and subtitles/captions. The application may use the Query Language Preferences API to determine these settings.

The Query Language Preferences API shall be defined as follows:

method: "`org.atsc.query.languages`"

params: Omitted

Response:

result: a JSON object containing an object with three key/value pairs as defined below.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "preferredAudioLang": {"type": "string"},
      "preferredUiLang": {"type": "string"},
      "preferredCaptionSubtitleLang": {"type": "string"}
   },
   "required": ["msgType"]
}
```

`preferredAudioLang`, `preferredUiLang`, `preferredCaptionSubtitelLang` – Each of these strings indicate the currently set language preference of the respective item, coded according to RFC 5646 [5].

For example, the application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.languages",
    "id": 95
}
```

Moreover, if the user lives in the U.S. but has set his or her language preference for audio tracks and caption/subtitles to Spanish, the Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "preferredAudioLang": "es"},
        "preferredUiLang": "en"},
        "preferredCaptionSubtitleLang": "es"}
    },
    "id": 95
}
```

### 9.1.5 Query Caption Display Preferences API

The Broadcaster Application may wish to know the user's preferences for closed caption displays, including font selection, color, opacity and size, background color and opacity, and other characteristics. The application may use the Query Caption Display Preferences API to determine these settings.

The Query Caption Display Preferences API shall be defined as follows:

method: "`org.atsc.query.captionDisplay`"

params: Omitted

Response:

result: a JSON object containing an object with key/value pairs as defined below.

result JSON Schema:

```
{
    "type": "object",
    "properties": {
        "characterColor": {"type": "string"},
        "characterOpacity": {"type": "number"},
        "characterSize": {"type": "integer"},
        "fontStyle": {"enum": [
            "Default",
            "MonospacedSerifs",
            "PropoortionalSerifs",
            "MonospacedNoSerifs",
            "ProportionalNoSerifs",
            "Casual",
            "Cursive",
            "SmallCaps"
        ]},
        "backgroundColor": {"type": "string"},
        "backgroundOpacity": {"type": "number"},
        "characterEdge": {"enum": [
            "None",
            "Raised",
            "Depressed",
            "Uniform",
            "LeftDropShadow",
            "RightDropShadow"
        ]},
        "characterEdgeColor": {"type": "string"},
        "windowColor": {"type": "string"},
        "windowOpacity": {"type": "number"}
    },
    "required": ["msgType"]
}
```

For example, the application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.captionDisplay",
    "id": 932
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "msgType": "captionDisplayPrefs",
        "characterColor": "#F00",
        "characterOpacity": 0.5,
        "characterSize": 16,
        "fontStyle": "MonospacedNoSerifs",
        "backgroundColor": "#888888",
        "backgroundOpacity": 0,
        "characterEdge": "None",
        "characterEdgeColor": "black",
        "windowColor": 0,
        "windowOpacity": 0
    },
    "id": 932
}
```

### 9.1.6 Query Audio Accessibility Preferences API

The Broadcaster Application may wish to know the audio accessibility settings in the Receiver, including whether the automatic rendering of the following is enabled: video description service, audio/aural representation of emergency information and what are the corresponding language preferences. The application may use the Query Audio Accessibility Preferences API to determine these settings.

The Query Audio Accessibility Preferences API shall be defined as follows:

method: "`org.atsc.query.audioAccessibility`"

params: Omitted

Response:

result: a JSON object containing an object as defined below.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "videoDescriptionService": {
         "type": "object",
         "properties": {
            "enabled": {"type": "boolean"},
            "language": {"type": "string"}
         }
      },
      "audioEIService": {
         "type": "object",
         "properties": {
            "enabled": {"type": "boolean"},
            "language": {"type": "string"}
         }
      }
   }
}
```

`videoDescriptionService.enabled`, `audioEI.enabled` – Each of these Boolean values respectively indicate the currently state of automatic rendering preference of video description service (VDS), audio/aural representation of emergency information.

`videoDescriptionService.language` – A string indicating the preferred language of VDS rendering, coded according to RFC 5646 [5].

`audioEI.language` – A string indicating the preferred language of audio/aural representation of emergency information rendering, coded according to RFC 5646 [5].

When a terminal does not have the setting for `videoDescriptionService.enabled`, `videoDescriptionService.language`, `audioEI.enabled`, `audioEI.language` then it is expected that the response does not include the corresponding property.

For example, the application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.audioAccessibility",
    "id": 90
}
```

In addition, if the user has set his or her automatic rendering preference setting of video description service set to ON and the terminal does not have rest of the settings, then the Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "videoDescriptionService": {
            "enabled": true
        }
    },
    "id": 90
}
```

### 9.1.7 Query MPD URL API

The Broadcaster Application may wish to access the current broadcast DASH MPD. The Query MPD URL API returns a URL the Broadcaster Application can use to retrieve (for example, by XHR) the MPD.

The Request MPD URL API shall be defined as follows:

method: "`org.atsc.query.MPDUrl`"

params: Omitted

Response:

result: a JSON object containing an object as defined below.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "MPDUrl": {"type": "string"}
   },
   "required": ["MPDUrl"]
}
```

`MPDUrl` – A fully-qualified URL that can be used by the receiver, for example in an XHR request, to retrieve the current broadcast MPD.

For example, the application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.MPDUrl",
    "id": 913
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"MPDUrl": "http://127.0.0.1:8080/10.4/MPD.xml"},
    "id": 913
}
```

### 9.1.8 Query Receiver Web Server URI API

The Broadcaster Application may wish to access the location of the Application Context Identifier environment provided by the Receiver. This environment provides access to resources delivered under the auspices of the Application Context Identifier defined for the currently-loaded Broadcaster Application. These are made available through the Receiver Web Server using a URI (see Section 5.3). This API provides access to that URI.

The Receiver Web Server URI API shall be defined as follows:

method: "`org.atsc.query.rwsURI`"

params: Omitted

Response:

result: a JSON object containing an object as defined below.

result JSON Schema:

```
{
   "type": "object",
   "properties": {
      "rwsUri": {
         "type": "string",
         "format": "uri"
      },
   "required": ["rwsUri"]
}
```

`rwsURI` – This return parameter shall contain the URI where the resources associated with the Application Context Identifier environment may be accessed.

For example, the application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.rwsURI",
    "id": 90
}
```

The receiver will respond with the URI of the Receiver Web Server for the environment defined for the current Application Context Identifier:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "rwsUri": "http://localhost:8080/contextA"
    },
    "id": 90
}
```

The resulting URI can be prepended to relative references to resources to access those resources on the receiver.

## 9.2 Asynchronous Notifications of Changes

The types of notifications that the Receiver shall provide to the application through the APIs defined in this section are as specified in Table 9.1. All use a `method` of `org.atsc.notify` and include a parameter called "`msgType`" to indicate the type of notification.

**Table 9.1** Asynchronous Notifications

| msgType | Event Description | Reference |
|---|---|---|
| `ratingChange` | Parental Rating Level Change – a notification that is provided whenever the user changes the parental blocking level in the receiver. | Sec. 9.2.1 |
| `ratingBlock` | Rating Block Change – a notification that is provided whenever the user changes the parental blocking level in the receiver such that the currently decoding program goes from blocked to unblocked, or unblocked to blocked. | Sec. 9.2.2 |
| `serviceChange` | Service Change – a notification that is provided if a different service is acquired due to user action, and the new service signals the URL of the same application. | Sec. 9.2.3 |
| `captionState` | Caption State – a notification that is provided whenever the user changes the state of closed caption display (either off to on, or on to off). | Sec. 9.2.4 |
| `languagePref` | Language Preference – a notification that is provided whenever the user changes the preferred language. | Sec. 9.2.5 |
| `personalization` | Personalization – a notification that is provided whenever the user changes any parameter that can be discovered by calling the Personalization Data API (TBD). | Sec. 9.2.6 |
| `CCDisplayPref` | Closed Caption display properties preferences | Sec. 9.2.7 |
| `AudioAccessPref` | Audio Accessibilities preferences | Sec. 9.2.8 |
| `MPDChange` | Notification of a change to the broadcast MPD | Sec. 9.2.9 |

### 9.2.1 Rating Change Notification API

The Rating Change Notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user makes any change to the parental blocking level in the Receiver.

The Rating Change Notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`ratingChange`" and a key/value pair named "`rating`" representing the new rating value setting.

params JSON Schema:

```
{
    "type": "object",
    "properties": {
        "msgType": {"type": {"enum": ["ratingChange"]}},
        "rating": {"type": "string"}
    },
    "required": ["msgType","rating"]
}
```

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

The `rating` string shall conform to the encoding specified in A/331 [1], Section 7.3.

As an example, if the user changes the rating to "TV-PG-D" in the US system (Rating Region 1), then the Receiver would issue a notification to the application with the new rating level as follows:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "ratingChange",
        "rating": "{1,'TV-PG-D', {0 'TV PG'}{1 'D'}"
    },
}
```

### 9.2.2 Rating Block Change Notification API

The Rating Block Change Notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user makes a change to the parental blocking level in the Receiver that results in a change to the rating blocking of the currently displayed service, either from unblocked to blocked or vice versa.

The Rating Block Change Notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`ratingBlock`" and a key named "`blocked`" with a Boolean value representing the state of blocking after the user action.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": {"enum": ["ratingBlock"]}},
      "blocked": {"type": "boolean"}
   },
   "required": ["msgType","blocked"]
}
```

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

An example in which the state of program blocking has gone from unblocked to blocked:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "ratingBlock",
        "blocked": true
    },
}
```

### 9.2.3 Service Change Notification API

The Service Change Notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user changes to another service also associated with the same application. Note that if the user changes to another service not associated with the same application, or the new service has no signaled Broadcaster Application, no `serviceChange` event is fired.

The Service Change Notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`serviceChange`" and a key named "`service`" with a string indicating URI of the new service.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": {"enum": ["serviceChange"]}},
      "service": {"type": "string"}
   },
   "required": ["msgType","service"]
}
```

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

The `service` string shall consist of the globally unique Service ID associated with the newly selected service as given in the USBD in

**bundleDescription.userServiceDescription**@globalServiceID. See A/331 [1] Sections 7.1.3 (ROUTE/DASH) and 7.2.1 (MMT).

In the following example, the user has caused a service change to a service with a globally unique Service ID "http://xbc.tv/wxbc-4.2":

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "serviceChange",
        "service": "http://xbc.tv/wxbc-4.2"
    },
}
```

### 9.2.4 Caption State Change Notification API

The Caption State Change notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user turns captions on or off.

The Caption State Change notification API is defined as follows:

method: "org.atsc.notify"

params: A JSON object consisting of a key named msgType with value "captionState" and a key named "captionDisplay" with a Boolean value representing the new state of closed caption display.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": {"enum": ["captionState"]}},
      "captionDisplay": {"type": "boolean"}
   },
   "required": ["msgType","captionDisplay"]
}
```

No reply from the application is expected from this notification, hence the "id" term is omitted.

For example, the Receiver notifies the application that caption display has been turned on:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.event.notify",
    "params": {
        "msgType": "captionState",
        "captionDisplay": true
    }
}
```

### 9.2.5 Language Preference Change Notification API

The Language Preference Change notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user changes the preferred language applicable to either audio, user interfaces, or subtitles/captions.

The Language Preference Change notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`langPref`" and one or more key/value pairs described below.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": {"enum": ["langPref"]}},
      "preferredAudioLang": {"type": "string"},
      "preferredUiLang": {"type": "string"},
      "preferredCaptionSubtitleLang": {"type": "string"}
   },
   "required": ["msgType"]
}
```

`preferredAudioLang`, `preferredUiLang`, `preferredCaptionSubtitelLang` – Each of these strings indicate the preferred language of the respective item, coded according to RFC 5646 [5]. At least one key/value pair shall be present.

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

For example, if the user has changed the preferred language of the captions to French as spoken in Canada:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "langPref",
        "preferredCaptionSubtitleLang": "fr-CA"}
    }
}
```

### 9.2.6 Personalization Change Notification API

The Personalization Change notification shall be issued by the Receiver to the currently executing Broadcaster Application if …

<<<TBD>>>

### 9.2.7 Caption Display Preferences Change Notification API

The Caption Display Preferences Change notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the changes preferences for display of closed captioning.

The Caption Display Preferences Change notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`captionDisplayPrefs`" and a number of keys describing various aspects of the preferences, as defined below.

params JSON Schema:

```
{
    "type": "object",
    "properties": {
        "msgType": {"enum": ["captionDisplayPrefs"]},
        "displayPrefs": {
            "type": "object",
            "properties": {
                "characterColor": {"type": "string"},
                "characterOpacity": {"type": "number"},
                "characterSize": {"type": "integer"},
                "fontStyle": {"enum": [
                    "Default",
                    "MonospacedSerifs",
                    "PropoortionalSerifs",
                    "MonospacedNoSerifs",
                    "ProportionalNoSerifs",
                    "Casual",
                    "Cursive,"
                    "SmallCaps"
                ]},
                "backgroundColor": {"type": "string"},
                "backgroundOpacity": {"type": "number"},
                "characterEdge": {"enum": [
                    "None",
                    "Raised",
                    "Depressed",
                    "Uniform",
                    "LeftDropShadow",
                    "RightDropShadow"
                ]},
                "characterEdgeColor": {"type": "string"},
                "windowColor": {"type": "string"},
                "windowOpacity": {"type": "number"}
            }
        }
    },
    "required": [
        "msgType",
        "displayPrefs"
    ]
}
```

`displayPrefs` – This required object shall provide one or more of the closed caption display preferences to follow.

`characterColor` – This parameter is a string that shall represent the color of the characters. The color value shall conform to the encoding for color as specified in the W3C recommendation for CSS3 color [18]. For example, red may be represented as `"#FF0000"`, `"#F00"`, `"rgb(100%,0%, 0%)"`, `"rgb(255,0, 0)"`, or simply `"red"`.

`characterOpacity` – This parameter is an integer or fixed-point number in the range 0 to 1 inclusive that shall represent the opacity of the characters. For example, a value of .33 shall mean 33% opaque; a value of 0 shall mean completely transparent.

`characterSize` – This parameter is a number that shall represent the font size in points of the characters, in the range 8 to 26.

`fontStyle` – This string shall indicate the style of the preferred caption font. The eight possible choices are as specified in CTA-708 [31] Section 8.5.3:

- "Default" (undefined)
- "MonospacedSerifs" – Monospaced with serifs (similar to Courier)
- "ProportionalSerifs" – Proportionally spaced with serifs (similar to Times New Roman)
- "MonospacedNoSerifs" – Monospaced without serifs (similar to Helvetica Monospaced)
- "ProportionalNoSerifs" – Proportionally spaced without serifs (similar to Arial and Swiss)
- "Casual" – Casual font type (similar to Dom and Impress)
- "Cursive" – Cursive font type (similar to Coronet and Marigold)
- "SmallCaps" – Small capitals (similar to Engravers Gothic)

`backgroundColor` – This parameter represents the color of the character background, given in the same CSS-compatible format as `characterColor`.

`backgroundOpacity` – This parameter is an integer or fixed-point number in the range 0 to 1 that shall represent the opacity of the character background. A value of 1 shall mean 100% opaque; a value of 0 shall mean completely transparent.

`characterEdge` – This parameter shall indicate the preferred display format for character edges. The preferred color of the edges (or outlines) of the characters are as given in `characterEdgeColor`. Edge opacities have the same attribute as the character foreground opacities. The choices and their effects on the character display are as specified in CEA-708-E [31] Section 8.5.8: "None", "Raised", "Depressed", "Uniform", "LeftDropShadow", and "RightDropShadow".

`characterEdgeColor` – This parameter represents the color of the character edges, if applicable, given in the same CSS-compatible format as `characterColor`.

`windowColor` – This parameter represents the color of the caption window background, given in the same CSS-compatible format as `characterColor`.

`windowOpacity` – This parameter is a number in the range 0 to 1 that shall represent the opacity of the caption window. A value of 1 shall mean 100% opaque; a value of 0 shall mean completely transparent.

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

For example, the Receiver notifies the application that the user has changed their caption display preferences to red text on gray background. All the available parameters are provided:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.event.notify",
    "params": {
        "msgType": "captionDisplayPrefs",
        "displayPrefs": {
        "characterColor": "red",
        "characterOpacity": 0.5,
        "characterSize": 18,
        "fontStyle": "MonospacedSerifs",
        "backgroundColor": "#888",
        "backgroundOpacity": 0.25,
        "characterEdge": "Raised",
        "characterEdgeColor": "black",
        "windowColor": 0,
        "windowOpacity": 0
    }
}
```

### 9.2.8 Audio Accessibility Preference Change Notification API

The Audio Accessibility Preference Change notification API shall be issued by the Receiver to the currently executing Broadcaster Application if the user changes accessibility settings for either video description service and/or audio/aural representation of emergency information (EI).

The Accessibility Preference Change notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`audioAccessibilityPref`" as described below.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": "string", "enum": ["audioAccessibilityPref"]},
      "videoDescriptionService": {
         "type": "object",
         "properties": {
            "enabled": {"type": "boolean"},
            "language": {"type": "string"}
         },
         "required": ["enabled"]
      },
      "audioEIService": {
         "type": "object",
         "properties": {
            "enabled": {"type": "boolean"},
            "language": {"type": "string"}
         },
         "required": ["enabled"]
      }
   },
   "required": ["msgType"], "minProperties": 2
}
```

`videoDescriptionService.enabled` – A Boolean value representing the new state of video description service (VDS) rendering.

`videoDescriptionService.language` – A string indicating the preferred language of VDS rendering, coded according to RFC 5646 [5]. This property shall be present in the notification when `videoDescriptionService.enabled` is equal to `true` and the preferred language of VDS rendering is available at the terminal.

`audioEIService.enabled` – A Boolean value representing the new state of audio/aural representation of emergency information rendering.

`audioEIService.language` – A string indicating the preferred language of audio/aural representation of emergency information rendering, coded according to RFC 5646 [5]. This property shall be present in the notification when `audioEIService.enabled` is equal to `true` and the preferred language of audio/aural representation of emergency information rendering is available at the terminal.

No reply from the application is expected for this notification, hence the "`id`" term is omitted.

For example, if the user has changed the video description service's accessibility preference to ON, the terminal notifies the application the current state of video description service and the VDS language preference (when present) as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "audioAccessibilityPref",
        "videoDescriptionService": {
            "enabled”: true,
            "language”: "en"
        }
    }
}
```

### 9.2.9 MPD Change Notification API

The MPD Change notification API shall be issued by the Receiver to the currently executing Broadcaster Application if there is a change to the version of the broadcast MPD and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.4.4. The Broadcaster Application may respond to the notification of a change to the MPD by using the MPD URL discovered using the Query MPD URL API specified in Section 9.1.7 to fetch a new copy.

The MPD Change notification API is defined as follows:

method: "`org.atsc.notify`"

params: A JSON object consisting of a key named `msgType` with value "`MPDChange`".

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "msgType": {"type": {"enum": ["MPDChange"]}},
   },
   "required": ["msgType"]
}
```

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

For example, the Receiver may indicate that a new MPD is available by issuing this JSON RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "MPDChange",
    }
}
```

## 9.3 Event Stream APIs

DASH-style Events intended for broadcast applications can be encountered in broadcast media, either as Event Message (`'emsg'`) Boxes in band with the media in DASH Segments, or as **`EventStream`** elements at the Period level in a DASH MPD. These Events can initiate interactive

actions on the part of an application, or they can indicate that new versions of files are being delivered, or various other things.

Three APIs are needed to support this function:

- Subscribe to an Event Stream
- Unsubscribe from an Event Stream
- Receive an Event from a subscribed Event Stream

### 9.3.1 Event Stream Subscribe API

Broadcaster Applications can be notified when certain DASH Event Stream events are encountered in the MPD or the Media Segments. For MPEG DASH, the Event Message Box (`'emsg'`) box contains in-band events, and the MPD can include static events in an **`EventStream`** element at the **`Period`** level. A Broadcaster Application that wishes to be notified when a particular type of event occurs may register for that type of event using a `schemeIdUri` and optionally an accompanying `value` parameter.

The Event Stream Subscribe API (sent from the application to Receiver) shall be defined as follows:

method: "`org.atsc.eventStream.subscribe`"

params: A JSON object containing a `schemeIdUri` and optionally an accompanying `@value`.

params JSON Schema:

```
{
    "type": "object",
    "properties": {
        "schemeIdUri": {
            "type": "string",
            "format": "uri"
        },
        "value": {"type": "string"}
    },
    "required": ["schemeIdUri"]
}
```

`schemeIdUri` – The `schemeIdUri` URI string associated with the Event Stream event of interest to the application.

`value` – An optional string used to identify a particular Event Stream event.

For example, if the application wishes to register for Event Stream events associated with `schemeIdUri` "`urn:uuid:9a04f079-9840-4286`", it could subscribe as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.subscribe",
    "params": {"schemeIdUri": "urn:uuid:9a04f079-9840-4286"},
    "id": 22
}
```

The Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 22
}
```

The Receiver would then be set to communicate any Event Stream events tagged with `schemeIdUri` `"urn:uuid:9a04f079-9840-4286"` to the application using the Event Stream Event API defined in Section 9.3.3 below.

If the application were only interested in Event Stream events associated with this `schemeIdUri` when the accompanying `value` = "17", it could subscribe while including the `value` parameter:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.subscribe",
    "params": {
        "schemeIdUri": "urn:uuid:9a04f079-9840-4286",
        "value": "17"}
    },
    "id": 23
}
```

The Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 23
}
```

The Receiver would then be set to communicate any Event Stream event tagged with `schemeIdUri` `"urn:uuid:9a04f079-9840-4286"` and `value` = "17" to the application using the notification API defined in Section 9.3.3 below. The application would not be notified of Event Stream events tagged with unsubscribed values of `schemeIdUri` or those with a subscribed `schemeIdUri` but not matching any specified `value`.

The application may subscribe to multiple different Event Stream events (with different `schemeIdUri` values, or different `schemeIdUri`/`value` combinations).

Once subscribed, the application may unsubscribe using the API described in Section 9.3.2.

### 9.3.2 Event Stream Unsubscribe API

If a Broadcaster Application has subscribed to an Event Stream using the Event Stream Subscribe API defined in Section 9.1, it can use the Event Stream Unsubscribe API defined here to request that the Receiver discontinue notifications pertaining to the identified event.

method: "`org.atsc.eventStream.unsubscribe`"

params: A JSON object containing a `schemeIdUri` and optionally an accompanying `@value`.

params JSON Schema:

```
{
    "type": "object",
    "properties": {
        "schemeIdUri": {
            "type": "string",
            "format": "uri"
        },
        "value": {"type": "string"}
    },
    "required": ["schemeIdUri"]
}
```

`schemeIdUri` – The `schemeIdUri` URI string associated with the Event Stream event for which the application would like to remove the subscription.

`value` – An optional string used to identify a particular Event Stream event.

For example, if the application wishes to unsubscribe to all Event Stream events associated with `schemeIdUri` "`urn:uuid:9a04f079-9840-4286`", regardless of the value of the `value` parameter, it could use the following API:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.unsubscribe",
    "params": {"schemeIdUri": "urn:uuid:9a04f079-9840-4286"},
    "id": 26
}
```

If the operation was successful, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 26
}
```

If the application had subscribed to this same `schemeIdUri` using `value`="47" and `value`="48", and now wished to unsubscribe to the latter, it could use the following API:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.unsubscribe",
    "params": {
        "schemeIdUri": "urn:uuid:9a04f079-9840-4286",
        "value": "48"
    },
    "id": 29
}
```

If the operation were successful, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 29
}
```

### 9.3.3 Event Stream Event API

The Event Stream Event API shall be issued by the Receiver to the currently executing Broadcaster Application if an event is encountered in the content of the currently selected Service or currently playing content that matches the value of `schemeIdUri` (and accompanying value, if it was provided in the subscription) provided in a prior Event Stream Subscription.

The Event Stream Event API shall be as defined below:

method: "`org.atsc.eventStream.event`"

params: A JSON object conforming to the JSON Schema defined below.

params JSON Schema:

```
{
    "type": "object",
    "properties": {
        "schemeIdUri": {
            "type": "string",
            "format": "uri"
        },
        "value": {"type": "string"},
        "timescale": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4294967295
        },
        "presentationTime": {
            "type": "integer",
            "minimum": 0,
            "maximum": 4294967295
        },
        "duration": {
            "type": "integer",
            "minimum": 0,
            "maximum": 4294967295
        },
        "id": {
            "type": "integer",
            "minimum": 0,
            "maximum": 4294967295
        },
        "data": {"oneOf": [
            {"type": "string"},
            {"type": "object"}
        ]}
    },
    "required": ["schemeIdUri"]
}
```

No reply from the application is expected from this notification, hence the "`id`" term is omitted.

An example Event Stream notification message that might occur if the application had registered for Event Stream events using a `schemeIdUri` of `tag:xyz.org:evt:xyz.aaa.9`:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.event",
    "params": {
        "schemeIdUri": "tag:xyz.org:evt:xyz.aaa.9",
        "value": "ev47",
        "timeScale": 10,
        "presentationTime": 5506,
        "id": 60,
        "data": "d8a0c98fs08-d9df0809s"
    }
}
```

Note in this example that if the application had included a value parameter in the subscription, and that parameter had not been "ev47", this particular event would not be forwarded to the application.

## 9.4 Request Receiver Actions

### 9.4.1 Acquire Service API

The current service may be changed by two entities, the application via request to the receiver, or the user via the receiver directly. Depending on the information sent in the application signaling, the receiver will do one of the following when a new service is successfully selected:

- If the application signaling indicates that the same application should be launched for the new service, then allow the application to continue to run, and send the application a service change notification. (This is also what would happen if the service were changed by the user directly, and the application signaling indicated that the same application should be launched for the new service.)
- If the application signaling indicates that no application or a different application should be launched for the new service, then terminate the current application.

The reason why a Broadcaster Application might request the receiver to change the service selection might be to jump to another service of the same broadcaster for content that might be of interest to the user. This request is an asynchronous request. The receiver processes the request and if it can, it changes the service selection.

The Acquire Service API shall be defined as follows:

method: "`org.atsc.acquire.service`"

params: the globally unique Service ID of the service to be acquired.

params JSON Schema:

```
{
  "type": "object",
  "properties": {
    "svcToAcquire": {"type": "string"},
    "required": ["svcToAcquire"]
  }
}
```

`svcToAcquire` – This required string shall correspond to the globally unique Service ID (as defined in **`bundleDescription.userServiceDescription`**`@globalServiceID`; see A/331 [1]) of the service to acquire.

Response:

If the acquisition is successful, the Receiver shall respond with a JSON RPC response object with a null `result` object. If acquisition is not successful, the Receiver shall respond with a JSON RPC response object including one of the following `error` objects (See Table 8.4):

```
"error": {"code": -6, "message": "Service not found"}
"error": {"code": -7, "message": "Service not authorized"}
```

For example, if the application requests access to a service represented by globally unique Service ID "`http://xbc.tv/wxbc-4.3`", it can issue this request to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.acquire.service",
    "params": {"svcToAcquire": "http://xbc.tv/wxbc-4.3"},
    "id": 59
}
```

The Receiver would respond, if acquisition were successful with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 59
}
```

If globally unique Service ID "`http://xbc.tv/wxbc-4.3`" is unknown to the receiver, the response would be:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -6, "message": "Service not found"},
    "id": 59
}
```

### 9.4.2 Video Scaling and Positioning API

A Broadcaster Application in an application-enhanced Service (e.g. playing within the video plane that is positioned on top of the video produced by the Receiver Media Player) can use the video scaling and positioning JSON RPC method to request that the RMP render its video at less than full-scale (full screen), and to position it at a specified location within the display window.

The Video Scaling and Positioning API shall be defined as follows:

method: "`org.atsc.scale-position`"

params: the scaling factor and the X/Y coordinates of the upper left corner of the scaled window.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "scaleFactor": {
         "type": "integer",
         "minimum": 10,
         "maximum": 100
      },
      "xPos": {
         "type": "integer",
         "minimum": 0,
         "maximum": 100
      },
      "yPos": {
         "type": "integer",
         "minimum": 0,
         "maximum": 100
      },
      "required": ["scaleFactor","xPos","yPos"]
   }
}
```

`scaleFactor` – This required integer in the range 0 to 100 shall represent the video scaling parameter, where 100 represents full-screen (no scaling);

`xPos` – This required integer in the range 0 to 100 shall represent the X-axis location of the left side of the RMP's video window, represented as a percentage of the full width of the screen. A value of 0 indicates the left side of the video window is aligned with the left side of the display window. A value of 50 indicates the left side of the video window is aligned with the vertical centerline of the display window, etc.

`yPos` – This required integer in the range 0 to 100 shall represent the Y-axis location of the top of the RMP's video window, represented as a percentage of the full height of the screen. A value of 0 indicates the top of the video window is aligned with the top of the display window. A value of 50 indicates the top of the video window is aligned with the horizontal centerline of the display window, etc.

The zero axis of the coordinate system shall be the upper left corner, as with CSS.

The parameter values shall be set such that no portion of the video window would be rendered outside the display window.

For example, if the application wished to scale the displayed video to 25% of full screen, and position the left edge of the display horizontally 10% of the screen width and the top edge of the display vertically 15% of the screen height, it would issue this JSON RPC API to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.scale-position",
    "params": {
        "scaleFactor": 25,
        "xPos": 10,
        "yPos": 15
    },
    "id": 589
}
```

If scaling/positioning were successful, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 589
}
```

If scaling/positioning were not successful, the Receiver would respond with a JSON object including an "`error`" object:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -8, "message": "Video scaling/position failed"},
    "id": 589
}
```

### 9.4.3 XLink Resolution API

An XLink Resolution request shall be issued by the Receiver to the currently executing Broadcaster Application if the Receiver Media Player encounters an `xlink` attribute in an MPD as an attribute of a **`Period`** element. The Receiver shall request that the Broadcaster Application resolve the XLink, e.g. to return one or more **`Period`** elements it will use to replace the **`Period`** element in which the XLink appeared.

The XLink Resolution API shall be defined as follows:

method: "`org.atsc.xlinkResolution`"

params: A JSON object consisting of a key named `xlink` and a string representing the contents of the "`xlink:href`" element.

params JSON Schema:

```
{
    "type": "object",
    "properties": {"xlink": {
        "type": "string",
    }},
    "required": ["xlink"]
}
```

`xlink` – This required string shall be the XLink value from the `xlink:href` attribute in the MPD **`Period`** element.

Response:

result: A JSON object containing a "resolution" key whose string value represents one or more **`Period`** elements.

result JSON Schema:

```
{
   "type": "object",
   "properties": {"resolution": {
      "type": "string",
   }},
   "required": ["resolution"]
}
```

For example, the Receiver notifies the application of an XLink:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.xlinkResolution",
    "params": {"xlink": "urn:xbc 4399FB77-3939EA47"},
    "id": 5
}
```

Upon success, the application might respond:

```
--> {
    "jsonrpc": "2.0",
    "result": {"resolution":"<Period start='PT9H'> <AdaptationSet
      mimeType='video/mp4' /> <SegmentTemplate timescale='90000'
      media='xbc-$Number$.mp4v' duration='90000' startNumber='32401' />
      <Representation id='v2' width='1920' height='1080' />"},
    "id": 5
}
```

Note the use of single quotes for all attributes in the Period (required for proper JSON value syntax).

If the application is unable to resolve the XLink, it can respond with an error code -9:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -9, "message": "XLink cannot be resolved"},
    "id": 59
}
```

### 9.4.4 Subscribe MPD Changes API

The Subscribe MPD Changes API can be used by a Broadcaster Application to be notified whenever the version of the broadcast MPD currently in use by the RMP changes. Once subscribed, the Receiver notifies the Broadcaster Application when any version change occurs by issuing the MPD Change Notification API specified in Section 9.2.9. Notifications continue until an Unsubscribe MPD Changes API (Section 9.4.5) is issues, or until the Service is changed.

The Subscribe MPD Changes API shall be defined as follows:

method: "`org.atsc.subscribeMPDChange`"

params: none.

For example, the Broadcaster Application can subscribe to MPD changes by issuing:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.subscribeMPDChange",
    "id": 55
}
```

Upon success, the application would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 55
}
```

### 9.4.5 Unsubscribe MPD Changes API

The Unsubscribe MPD Changes API can be issued by a Broadcaster Application to stop receiving notifications of MPD changes.

The Unsubscribe MPD Changes API shall be defined as follows:

method: "`org.atsc.unsubscribeMPDChange`"

params: none.

For example, the Broadcaster Application can subscribe to MPD changes by issuing:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.unsubscribeMPDChange",
    "id": 56
}
```

Upon success, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 56
}
```

### 9.4.6 Set RMP URL API

The Broadcaster Application may choose to use the Receiver Media Player to play video content originated from an alternate source (e.g. broadband or locally cached content) instead of the broadcast-delivered content. In this way, the Broadcaster Application can take advantage of an optimized media player provided by the receiver. The Broadcaster Application may use the SET RMP URL API to request the receiver to use its RMP to play content originated from a URL provided by the application. Once the receiver is notified to play content from the application-provided URL, the RMP stops rendering the broadcast content (or the content being rendered at the time of the request) and begins rendering the content referenced by the new URL.

The Set RMP URL API shall not change the information on the selected service. It only changes the location from which the RMP fetches the media content it will play. The effects of changing this URL are temporary and if the Service is re-selected, the RMP defaults back to using the MPD defined in the service-level signaling.

The Broadcaster Application specifies the content to be played by the RMP by providing the URL of an MPD. The MPD shall be constructed in accordance with the "Guidelines for Implementation: DASH-IF Interoperability Point for ATSC 3.0" [3].

An optional "offset" parameter may be included. For locally cached content or broadband content, the offset indicates the time offset of a point in the stream at which the video should be started. If not provided, the content shall be played from the beginning. This allows flexibility for many use cases including bookmarking.

The SET RMP URL API shall be defined as follows:

method: "`org.atsc.setRMPURL`"

params: A JSON object consisting of a key named `rmpurl` and an optional additional key named `offset`.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "rmpurl": {"type": "string"},
      "offset": {"type": "number"}
   },
   "required": ["rmpurl"]
}
```

`rmpurl` – This required string shall be a URL referencing an MPD to be played by the RMP. The URL shall be accessible to the Receiver.

`offset` – This optional numeric value shall be defined as follows: TBD.

Response:

result: A null object upon success.

error: The following error codes may be returned:

- -11: The indicated MPD cannot be accessed
- -12: The content cannot be played
- -13: The requested offset cannot be reached

For example, if the Broadcaster Application requests the RMP to play content from a broadband source at a DASH server located at `http://stream.wxyz.com/33/program.xml`, it can issue a command to the receiver as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "properties": {"rmpurl": "http://stream.wxyz.com/33/program.xml"},
    "id": 104
}
```

Upon success, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 104
}
```

If the receiver's RMP cannot play the content, the Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "error": { "code": -12, "message": "The content cannot be played"},
    "id": 104
}
```

### 9.4.7 Audio Volume API

This API is under discussion and thus subject to change. Use cases for this API will be considered, as well as the potential impact of immersive audio modes.

By default, the audio output of the Receiver Media Player and that of the user agent are mixed. The Broadcaster Application may set and get the volume of the HTML media element using the `.volume` property. It may wish to set and get the audio volume of the Receiver Media Player. For example, the Broadcaster Application might mute the audio output of broadcast service when the user chooses to watch broadband content rendered with an HTML media element. The Audio Volume API may be used for such a case.

Figure 9.1 illustrates audio processing in an example receiver in which the audio output of the User Agent is mixed with the audio output of the Receiver Media Player for presentation to the user. The Broadcaster Application controls the volume of its output using the `.volume` property of the `HTMLMediaElement`. Analogously, the Audio Volume API defined here may be used to set the volume of the Receiver Media Player, shown as "V1" in the figure.

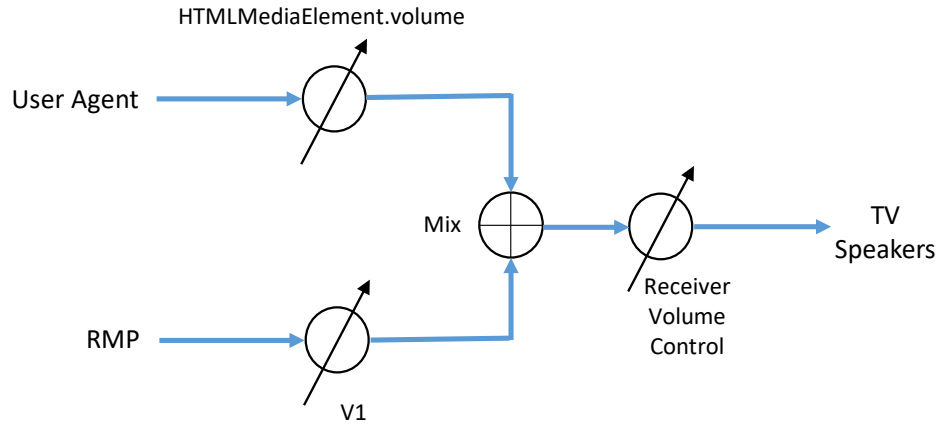


**Figure 9.1** RMP audio volume.

This request is an asynchronous request. If a volume element is provided in the request, the receiver processes the request to set the RMP volume. The receiver's response provides the current volume in either case.

The Audio Volume API shall be defined as follows:

method: "`org.atsc.audioVolume`"

params JSON Schema:

```
{
  "type": "object",
  "properties": {
    "audioVolume": {"type": "number"},
  }
}
```

`audioVolume` – This optional floating-point number in the range 0 to 1, when present, shall correspond to a value of audio volume to be set in the Receiver Media Player. The value of shall be from 0.0 (minimum or muted) to 1.0 (full volume). The encoding is the same as the `.volume` property of the HTML media element. If volume is not specified in the request, the volume is not changed by this request. This can be used to determine the current volume setting.

Response:

result: A JSON object containing an audioVolume key whose value represents one or more **`Period`** elements.

result JSON Schema:

```
{
  "type": "object",
  "properties": {"audioVolume": {
    "type": "number",
  }},
  "required": ["audioVolume"]
}
```

`audioVolume` – This floating-point number in the range 0 to 1 shall indicate the current audio volume of the Receiver Media Player, where 0 indicates minimum volume or muted, and 1.0 indicates full volume.

For example, if the application wishes for the Receiver Media Player set the audio volume to half volume (50%):

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.audioVolume ",
    "params": {"audioVolume ": 0.5},
    "id": 239
}
```

If the request is processed successfully, the Terminal might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"audioVolume ": 0.5},
    "id": 239
}
```

### 9.5 Media Track Selection API

The Broadcaster Application may request the receiver's Receiver Media Player to select a particular video stream available in the Service, for example an alternate camera angle. Alternatively, it might request the Receiver Media Player to select an audio presentation other than the one it would have chosen based on the user's preferences. The Media Track Selection API may be used for these cases.

This request is an asynchronous request. The receiver processes the request and if it can, it changes the selection.

The Media Track Selection API shall be defined as follows:

method: "`org.atsc.track.selection`"

params: an id value associated with the DASH **`Period.AdaptationSet`** element to be selected, or alternatively, for complex audio presentations involving pre-selection, the DASH **`Period. Preselection@id`** value. For unambiguous selection of one track or audio presentation, all id values within the Period should be unique.

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "selectionId": {"type": "integer"},
      "required": ["selectionId"]
   }
}
```

`selectedId` – This required integer shall correspond to a value of `@id` attribute in either an **`AdaptationSet`** in the current Period, or an `@id` attribute in a **`Preselection`** element in the current Period.

For example, if the application wishes for the Receiver Media Player to find and select a video **`AdaptationSet`** with an id value of 5506, it could send the following WebSocket message:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.track.selection",
    "params": {"selectionId": 5506},
    "id": 29
}
```

If the requested **`AdaptationSet`** were successfully selected, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 329
}
```

If the requested track cannot be selected, the Receiver shall respond with error code -10:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -10, "message": "Track cannot be selected"},
    "id": 329
}
```

### 9.6 Media Segment Get API

Broadcaster Application may request media segments based on interpretation of the MPD. Since JSON is not a suitable mechanism for delivery of binary data, the JSON messaging is meant to indicate the beginning and end of the media segment/sub-segment being delivered to the application.

The Media Segment Get API is defined as follows:

method: "`org.atsc.media.segment.get`"

params: URL

Response:

result:

- Media Segment response – JSON-RPC delimited binary segment

For example:

```
--> {"jsonrpc": "2.0", "method": "org.atsc.media.segment.get", "URL":  URL}
<-- {"jsonrpc": "2.0", "response": "org.atsc.media.segment.start"}
<-- Binary data through WebSocket
<-- {"jsonrpc": "2.0", "response": "org.atsc.media.segment.end"}
```

### 9.7 Mark Unused API

The Mark Unused API shall be used by the currently-executing Broadcaster Application to indicate to the Local Receiver Cache system that an element within the cache is unused. The

Receiver may then perform the appropriate actions to reclaim the resources used by the unused element.

The Mark Unused API is defined as follows:

method: "`org.atsc.cache.markUnused`"

params: A JSON object consisting of a key named `elementUri` with the URI of the element that should be marked unused as follows:

params JSON Schema:

```
{
   "type": "object",
   "properties": {
      "elementUri": {"type": "string"}
      "format": "uri"
   },
   "required": ["elementUri"]
}
```

For example, the Broadcaster Application may wish to indicate that a particular replacement ad was not needed anymore after it had been used. The Broadcaster Application would mark the MPD file as unused indicating the underlying resources could be reclaimed, perhaps for another replacement ad. Similarly, the Broadcaster Application would also mark all segments referenced by the MPD as unused as well. Alternatively, it could mark the entire directory, "`adContent`", unused if the directory only contained the replacement ad MPD and its associated segments.

The RPC request would be formatted as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.cache.markUnused",
    "params": {"elementUri": " http://192.168.0.42:4488/2/adContent
      /replacement.ad.mpd"},
    "id": 42
}
```

The Receiver responds with the following on success:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 42
}
```

Standard HTTP failure codes shall be used to indicate issues with the formation of the URI and that the file or directory referenced could not be marked as unused. If an element is successfully marked as unused, future attempts to access that element will have indeterminate results in that some receivers may not have made the element unavailable and respond positively to the request while others may immediately respond with an error status.

# 10. DASH AD INSERTION

The ATSC 3.0 runtime environment supports personalized client-side ad insertion using MPEG DASH tools. The mechanism depends on the type of service: for regular "watch TV" types of services that include application-based features, the application can provide a service to the Receiver Media Player to resolve an XLink, while for application-based services, the application itself can determine the appropriate content to play for any particular user. The XLink-based method is described and specified here.

## 10.1 Dynamic Ad Insertion Principles

The basic principles of dynamic ad insertion include:

1) Signaling an "ad avail" to indicate an available slot for a replacement ad or ad pod;
2) Controlling the pre-caching of ad replacement content;
3) Choosing the appropriate ad for download based on user preferences, cookies (past viewing, or past answers to questions), custom logic, etc.; and
4) Executing the seamless splice of replacement ad into broadcast stream

The method specified herein involves:

1) Signaling an "ad avail" as a DASH Period in an MPD;
2) Tagging each replaceable Period with an XLink;
3) Having the Receiver Media Player call on the Broadcaster Application to resolve each XLink;
4) Managing the pre-caching of ad content, if necessary, within the Broadcaster Application;
5) Employing the Broadcaster Application to choose the appropriate ad, personalized to this viewer; and
6) Having the Receiver Media Player execute the seamless splice;

Figure 10.1 describes three DASH Periods, designated as P1, P2, and P3. Here, P2 is an "ad avail," meaning that a possible replacement ad may be substituted for the content that would otherwise play during that interval. If no personalization is done, the default content (called "P2*") will play. The asterisk indicates that the MPD element describing P2 includes an XLink. Based on some personalization criteria (described later), P2 could be replaced with an alternative content, shown as P2a, P2b, or P2c in the figure.

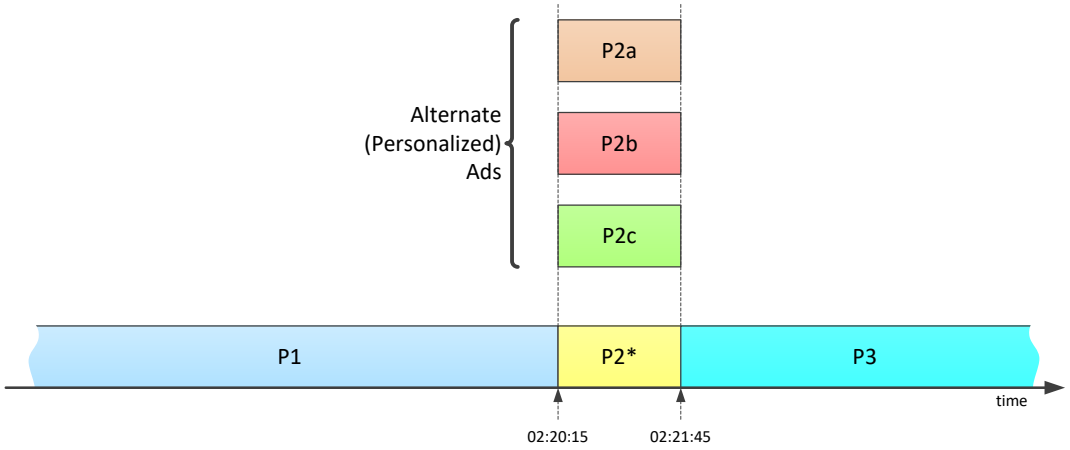


**Figure 10.1** Personalized ad periods.

In this example, the broadcast MPD includes an XLink attribute within the Period element describing P2.

## 10.2 Overview of XLinks

The XLink concept is specified by W3C in [30]. While XLinks have many applications, for the purposes of the ATSC 3.0 runtime environment we are only using one feature: an XLink appearing as an attribute of an XML element can be "resolved" to replace that entire element with the result of the resolution process. This feature is signaled by setting the XLink "show" attribute to the value "embed," as shown in the simplified example `MPD.Period` given in Figure 10.2.

```
<Period start="PT9H" xlink:show="embed" xlink:href="private-data-known-to-app">
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

**Figure 10.2** Example period with XLink.

When the Receiver Media Player receives an updated MPD including XLinks in one or more `Period` elements, if there is a Broadcaster Application currently running, it uses the XLink Resolution API specified in Section 9.4.3 to pass the contents of the `xlink:href` attribute as a parameter to the application to attempt to get it resolved. If successful, the application will respond with a replacement `Period` element. For the example above, the replacement `Period` might look like the one given in Figure 10.3.

```
<Period start="PT9H">
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="ad6-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

**Figure 10.3** Example remote period.

In the example, the original content (referenced by the relative URL `"xbc-$Number$.mp4v"`) was replaced by a personalized ad segment referenced by `"ad6-$Number$.mp4v"`.

# *Annex A*: Obscuring the Location of Ad Avails

## A.1 OBSCURING THE LOCATION OF AD AVAILS

The XLink-based technique described in Section 10.2 may be used to obscure the location of ad avails, thanks to the possibility that the XLink resolution can return not one, but multiple `Period` elements.

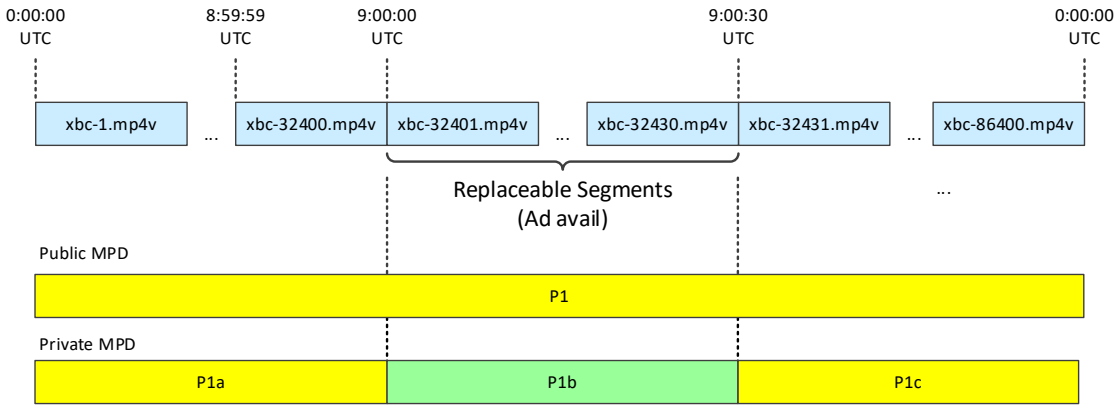


**Figure A.1.1** Public and private MPDs.

Figure A.1.1 describes a "public" MPD and a "private" MPD. The public MPD is delivered in the broadcast emission stream, and signals one Period (P1) starting at midnight UTC on 1 July, 2016, and lasting for 24 hours. Figure A.1.2 shows, in simplified form, this MPD instance.

```
<MPD type="dynamic" … availabilityStartTime="2016-07-01T00:00:00Z">
…
<Period start="PT0S" duration="PT1D" xlink:show="embed"
   xlink:href="urn:xbc?d=2016-07-01&s=4A92D344092A9A09eC35" > <!-- P1 -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
</MPD>
```

**Figure A.1.2** Example public MPD.

The MPD includes an XLink within the one Period. The resolution of this XLink yields three Periods, replacing the one. This "private" MPD is shown in Figure A.1.1 at the bottom. Now, the Period P1 is broken into three sub-Periods, P1a, P1b, and P1c. The middle Period, P1b, is actually an ad avail occurring at 9:00 am UTC and lasting for 30 seconds.

Note that the public MPD indicated Media Segments of 1-second duration running all day long. Thus, the first Media Segment of the day would be `xbc-1.mp4v`, running through to the end of the day with `xbc-864000.mp4v`.

Figure A.1.3 illustrates an MPD that would result in output of exactly the same media content as the MPD given in Figure A.1.2. This example is given only to illustrate the concept of the `Period.SegmentTemplate.startNumber`.

```
<MPD type="dynamic" … availabilityStartTime="2016-07-01T00:00:00Z">
…
<Period start="PT0S" >   <!-- P1a -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

```
<Period start="PT9H" >   <!-- P1b -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000"
        startNumber="32401" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

```
<Period start="PT9H0M30S" >      <!-- P1c -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000"
        startNumber="32431" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
</MPD>
```

**Figure A.1.3** Example MPD equivalent.

Although there are two additional Periods, the media content referenced by the middle one (the Period starting at 9 a.m.), is exactly the same as before, because the starting index number is set to 32,401. This is the same number, and hence the same Media Segment, that would have played at 9 a.m. according to the MPD of Figure A.1.2. Likewise, at 30 seconds after 9:00, the start number of 32,431 references the same content as before.

Therefore, if XLink resolution would return these three Periods, nothing would change. A seamless ad replacement can occur, however, if the application replaces the middle period with different content. Figure A.1.4 illustrates the case that the content referenced in the middle period is now to a personalized ad of 30 seconds duration, by reference to `media="ad7-$Number$.mp4v"`.

```
<MPD type="dynamic" … availabilityStartTime="2016-07-01T00:00:00Z">
…
<Period start="PT0S" >   <!-- P1a -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

```
<Period start="PT9H" >  <AdaptationSet mimeType="video/mp4" … >   <!-- P1b* -->
    <SegmentTemplate timescale="90000" … media="ad7-$Number$.mp4v" duration="90000" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
```

```
<Period start="PT9H0M30S" >   <!-- P1c -->
  <AdaptationSet mimeType="video/mp4" … >
    <SegmentTemplate timescale="90000" … media="xbc-$Number$.mp4v" duration="90000"
       startNumber="32431" />
    <Representation id="v2" width="1920" height="1080" … />
  </AdaptationSet>
</Period>
</MPD>
```

**Figure A.1.4** Example ad replacement.

— End of Document —