ATSC Standard:
A/344:2025-06: “ATSC 3.0 Interactive Content”

A/344:2025-06
11 June 2025

Advanced Television Systems Committee
1300 I Street, N.W., Suite 400E
Washington, D.C. 20005
202-872-9160

The Advanced Television Systems Committee, Inc. is an international, non-profit organization developing voluntary standards and recommended practices for broadcast television and multimedia data distribution. ATSC member organizations represent the broadcast, professional equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries. ATSC also develops implementation strategies and supports educational activities on ATSC standards. ATSC was formed in 1983 by the member organizations of the Joint Committee on Inter-society Coordination (JCIC): the Consumer Technology Association (CTA), the Institute of Electrical and Electronics Engineers (IEEE), the National Association of Broadcasters (NAB), the Internet & Television Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE). For more information visit www.atsc.org.

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.

Implementers with feedback, comments, or potential bug reports relating to this document may contact ATSC at https://www.atsc.org/feedback/.

Revision History

Version	Date
Candidate Standard approved	29 December 2016
A/344:2017 Standard approved	18 December 2018
Candidate Standard Revision approved See Section 9.1 for details of WebSocket API changes	27 April 2018
CS update approved	24 January 2019
A/344:2019 Standard approved	2 May 2019
Amendment No. 1, Persistent IDs," approved	29 July 2019
Amendment No. 2, "JSON RPC Cancel Request API Addition," approved	16 December 2019
Amendment No. 3, "Redistribution Use Case," approved	18 December 2019
Amendment No. 4, "RMP State," approved	19 December 2019
Amendment No. 5, "Capabilities," approved	23 December 2019
Amendment No. 6, "Remove JavaScript Sample Code," approved	24 December 2019
Amendment No. 7, "CacheRequest API", approved	25 December 2019
Amendment No. 8, "Signaling Data", approved	7 February 2020
Amendment No. 9, "Service Guide", approved	31 January 2020
Amendment No. 10, "DRM Error Codes," approved	3 February 2020
A/344:2020 published (a roll-up of Amendments No. 1 through 10 to A/344:2019 plus editorial cleanup)	7 February 2020
Corrigendum No. 1 approved	29 April 2020
Amendment No. 1 approved	12 October 2020
A/344:2021 Standard approved	23 March 2021
A/344:2021 Candidate Standard Revision	14 June 2021
Updated CS approved by TG3/S38	13 September 2021

A/344:2021 Revision approved	18 March 2022
A/344:2022-03 Published (references to ATSC documents updated)	31 March 2022
A/344:2023-02 Revision approved	17 February 2023
A/344:2023-03 Published (references to ATSC documents updated)	28 March 2023
A/344:2023-03 Candidate Standard Revision approved	25 April 2023
A/344:2024-02 Revision approved	13 February 2024
A/344:2024-04 Published (references to ATSC documents updated)	3 April 2024
A/344:2024-04 Candidate Standard Revision approved	22 August 2024
A/344:2024-04 Revision approved Corrigendum No. 1 approved	26 February 2025 11 June 2025
A/344:2025-06 Published (a rollup of A/344:2025-02 Corrigendum No. 1)	11 June 2025

Table of Contents

Index of Figures and Tables

ATSC Standard:
"ATSC 3.0 Interactive Content"

1. SCOPE

This document describes the interactive content environment provided by an ATSC 3.0 Receiver. This environment is comprised of a standard W3C User Agent with known characteristics, a WebSocket interface for obtaining information from the Receiver and controlling various Receiver functionality, and an HTTP interface for accessing files delivered over broadcast. This document also specifies the life cycle of the interactive content when delivered over broadband or broadcast or both.

ATSC 3.0 is a defining emission standard, while the W3C User Agent defines a standard environment for executing interactive content. In order to create the appropriate user experience that aligns with the ATSC 3.0 delivery mechanisms, it is considered useful to specify a reference architecture of an ATSC 3.0 Receiver device, referred to in this document as the Reference Receiver Model (RRM) or simply the Receiver, to define and/or verify the proper emission specifications and to allow interactive content developers to target a known environment (see Section 5).

A decomposition of the functions and interfaces in the Receiver enables the definition of proper emission formats in order to verify that the distribution formats result in expected functionality to fulfill the ATSC 3.0 system requirements.

By no means would such a reference Receiver imply a normative implementation, as it would only provide an example implementation to verify the adequacy of the delivery specification. The RRM is expected to decompose the ATSC 3.0 Receiver device into the relevant network interfaces, device internal functions, interfaces to the Broadcaster Application and interfaces to the media playout pipeline.

It should be noted that the phrase "expected to" is used to describe how an interface or method is expected to work. It is anticipated that if the Receiver implements the interface or method, that the resultant behavior is consistent with this specification. This allows interactive content developers to implement to a well-defined application programming interface. Recommendations for Receiver implementations can be found in CTA CEB32.8 [40].

1.1 Introduction and Background

This document describes the environment and interfaces that can be used by interactive content to provide an enhanced viewer experience on a supporting ATSC 3.0 Receiver.

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 interactive content environment from the system level
- Section 5 – Specification of the Reference Receiver Model
- Section 6 – Describes how the Broadcaster Application is managed
- Section 7 – Details of the various Media Players supported by this standard

- Section 8 – Overview of the WebSocket interface supported by the Receiver
- Section 9 – Supported methods of the WebSocket interface
- Annex A – An informative Application Lifecycle Sequence Diagram
- Annex B – A complete copy of the JSON-RPC 2.0 specification used by this standard

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 Standard: System Discovery and Signaling," Doc. A/321:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[2] ATSC: "ATSC Standard: Link Layer Protocol," Doc. A/330:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[3] ATSC: "ATSC Standard: Signaling, Delivery, Synchronization, and Error Protection," Doc. A/331:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[4] ATSC: "ATSC Standard: Service Announcement," Doc. A/332:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[5] ATSC: "ATSC Standard: Content Recovery in Redistribution Scenarios," Doc. A/336:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[6] ATSC: "ATSC Standard: Application Event Delivery," Doc. A/337:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[7] ATSC: "ATSC Standard: Captions and Subtitles," Doc A/343:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

[8] ATSC: "ATSC Standard: ATSC 3.0 Security and Service Protection," Doc. A/360:2024-04, Advanced Television Systems Committee, Washington, DC, 3 April 2024.

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

[10] CTA: "CTA Bulletin: Recommendations for User Overrides for Closed Caption Decoders", Doc. CTA-CEB35, December 2019.

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

[12] IETF: "Augmented BNF for Syntax Specifications: ABNF," Doc. RFC 5234, Internet Engineering Task Force, January 2008. https://tools.ietf.org/html/rfc5234

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

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

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

[16] 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

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

[18] 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

[19] IETF Internet-Draft: "JSON Schema: A Media Type for Describing JSON Documents", September 16, 2019. Work in Progress.
https://tools.ietf.org/html/draft-handrews-json-schema-02

[20] IETF: "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, Internet Engineering Task Force, November 1996.
https://tools.ietf.org/html/rfc2045

[21] IETF: BCP 47, "Tags for Identifying Languages," Internet Engineering Task Force, Reston, VA, September 2009.
https://tools.ietf.org/html/bcp47

[22] IETF: "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 7159, Internet Engineering Task Force, March 2014.
https://tools.ietf.org/html/rfc7159

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

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

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

[26] IETF: "A Universally Unique IDentifier (UUID) URN Namespace," Doc. RFC 4122, Internet Engineering Task Force, July 2005.
https://tools.ietf.org/html/rfc4122

[27] IETF: "The Base16, Base32, and Base64 Data Encodings," RFC 4648, Internet Engineering Task Force, October 2006.
https://tools.ietf.org/html/rfc4648

[28] IETF: "The "data" URL scheme," RFC 2397, Internet Engineering Task Force, August 1998.
https://tools.ietf.org/html/rfc2397

[29] 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, 15 May 2014.

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

[31] W3C: "Encrypted Media Extensions," W3C Recommendation, World Wide Web Consortium, 18 September 2017. http://www.w3.org/TR/encrypted-media/

[32] W3C: "UI Events KeyboardEvent key Values," Section 3.18, Media Controller Keys, W3C Candidate Recommendation, 1 June 2017, World Wide Web Consortium. https://www.w3.org/TR/DOM-Level-3-Events-key/#keys-media-controller

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

[34] W3C: "Mixed Content," W3C Candidate Recommendation, Worldwide Web Consortium, 2 August 2016. (*work in process*). http://www.w3.org/TR/mixed-content/

[35] W3C: "XML Schema Part 2: Datatypes Second Edition," W3C Recommendation, Worldwide Web Consortium, 28 October 2004. https://www.w3.org/TR/xmlschema-2/

[36] WHATWG: "Living Standard", "Fetch Commit Snapshot", 30 November 2020: https://fetch.spec.whatwg.org/commit-snapshots/eda41525e3b462ce2035dd3cfc4a6ec1fc093c1d/

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

2.2 Informative References

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

[37] ATSC: "ATSC Standard: Companion Device, " Doc. A/338:2024-04, Advanced Television Systems Committee, 3 April 2024.

[38] ATSC: "ATSC Recommended Practice: Techniques for Signaling, Delivery and Synchronization," Doc. A/351:2024-04, Advanced Television Systems Committee, 3 April 2024.

[39] ATSC: "ATSC Recommended Practice: Digital Rights Management (DRM)," Doc. A/362:2024-04, Advanced Television Systems Committee, 3 April 2024.

[40] CTA: "Recommended Practice for ATSC Television Sets, Application Runtime Environment (CTA-CEB32.8-C)," December 2024, https://shop.cta.tech/collections/standards/products/recommended-practice-for-atsc-3-0-television-sets-application-runtime-environment-cta-ceb32-8-c

[41] DASH-IF: "Guidelines for Implementation: DASH-IF Interoperability Point for ATSC 3.0," Version 1.1, DASH Industry Forum, 12 June 2018. https://dashif.org/docs/DASH-IF-IOP-for-ATSC3-0-v1.1.pdf

[42] DASH-IF: "Implementation Guidelines: DASH events and timed metadata tracks timing and processing model and client reference model," DASH Industry Forum. https://dashif.org/docs/EventTimedMetadataProcessing-v1.0.2.pdf

[43] DASH-IF: "Protection System-Specific Identifiers," DASH Industry Forum. https://dashif.org/identifiers/content_protection/
[44] IANA Registry: Uniform Resource Names (URN) Namespaces. https://www.iana.org/assignments/urn-namespaces/urn-namespaces.xml
[45] IEEE: IEEE Registration Authority. https://regauth.standards.ieee.org/standards-ra-web/pub/view.html
[46] JSON-RPC: "JSON-RPC 2.0 Specification," JSON-RPC Working Group. http://www.jsonrpc.org/specification
[47] JSON Schema: "JSON Schema: A Media Type for Describing JSON Documents," Internet Engineering Task Force, JSON-Schema Working Group, 17 September 2019. http://json-schema.org/latest/json-schema-core.html *(work in progress)*
[48] W3C: "TTML Profiles for Internet Media Subtitles and Captions 1.0.1 (IMSC1)," W3C Recommendation, Worldwide Web Consortium. http://www.w3.org/TR/ttml-imsc1.0.1
[49] W3C: "XML Linking Language (XLink)," Recommendation Version 1.1, Worldwide Web Consortium, 6 May 2010. http://www.w3.org/TR/xlink11/
[50] WHATWG: "HTML Living Standard," Section 9.3 "Web sockets," Web Hypertext Application Technology Working Group. https://html.spec.whatwg.org/multipage/web-sockets.html
[51] J. Keiser, D. Lemire, "Validating UTF-8 In Less Than One Instruction Per Byte," Software: Practice and Experience, Vol. 51, No. 5, October 2020. https://arxiv.org/abs/2010.03090
[52] Android Media TV onSignalStrengthUpdated https://developer.android.com/reference/android/media/tv/TvView.TvInputCallback#onSignalStrengthUpdated(java.lang.String,%20int)

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 [11] shall be used. Where an abbreviation is not covered by IEEE practice or industry practice differs from IEEE practice, the abbreviation in question is 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.

3.1.1 A/344-specific Terms

The phrase "**expected to**" or the word "**expected**" are used to specify that the Receiver Reference Model and Broadcaster Application are expected to behave in a particular manner. Similarly, "not

expected to" is used to specify that the Receiver or Broadcaster Application are not expected to behave in the described manner. Many of these requirements are described as recommendations in CTA CEB32.8-C [40]. Note that neither A/344 nor CEB32.8 specify normative requirements for Receiver implementations. Furthermore, Broadcaster Application implementations are out-of-scope for ATSC standards, so conformance language is inappropriate in those cases.

3.2 Treatment of Syntactic Elements

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

3.2.1 Reserved Elements

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

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

3.3 Acronyms and Abbreviations

The following acronyms and abbreviations are used within this document.

ABNF Augmented Backus-Naur Form
AEA Advanced Emergency Alert
AEAT Advanced Emergency Alert Table [3]
AMP Application Media Player
API Application Programming Interface
ATSC Advanced Television Systems Committee
A/V Audio / Video
BA Broadcaster Application
BCP Best Current Practice
CD Companion Device [37]
CDM Content Decryption Module
CORS Cross-Origin Resource Sharing
CSS Cascading Style Sheets
CTA Consumer Technology Association
DASH Dynamic Adaptive Streaming over HTTP [29]
dB Decibels
DOM Document Object Model
DRM Digital Rights Management

DWD Distribution Window Description [3]
EFDT Extended File Delivery Table
EME W3C Encrypted Media Extensions [31]
ESG Electronic Service Guide [4]
FDT File Delivery Table
FLUTE File Delivery over Unidirectional Transport
GIF Graphics Interchange Format
HDMI High Definition Multimedia Interface
HELD HTML Entry pages Location Description [3]
HTML5 HyperText Markup Language, Fifth Version
HTTP HyperText Transfer Protocol
HTTPS HyperText Transfer Protocol Secure
IANA Internet Assigned Numbers Authority
ID Identifier
IMSC1 Internet Media Subtitles and Captions 1.0 [48]
IP Internet Protocol
JPEG Joint Photographic Experts Group
JSON JavaScript Object Notation [22]
JSON-RPC JSON Remote Procedure Call (Annex A)
LCT Layered Coding Transport
LLS Low-Level Signaling [3]
MIME Multipurpose Internet Mail Extensions
MMT MPEG Media Transport
MPD Media Presentation Description [41]
MPEG Moving Pictures Experts Group
MPU Media Processing Unit
MSE W3C Media Source Extensions [33]
msec Milliseconds
NRT Non-Real Time
OSN On Screen message Notification [3]
PD Primary Device [37]
PNG Portable Network Graphics
PTP Precision Time Protocol
RDT Recovery Data Table [5]
RFC Request For Comment
RMP Receiver Media Player
ROUTE Real-Time Object Delivery over Unidirectional Transport [3]
RPC Remote Procedure Call
RRM Reference Receiver Model
SHA1 Secure Hash Algorithm 1
SLS Service-Level Signaling [3]
SLT Service List Table **[3]**

S/MIME Secure/Multipurpose Internet Mail Extensions [8]
sRGB Standard Red Green Blue
STB Set-Top Box
TV Television
UI User Interface
URI Uniform Resource Identifier
URL Uniform Resource Locator
URN Universal Resource Name
UTC Universal Time Coordinated
UTF-8 Unicode Transformation Format – 8-bit
UUID Universally Unique Identifier
VDS Video Description Service
W3C Worldwide Web Consortium
WHATWG Web Hypertext Application Technology Working Group
WS WebSocket [50]
XHR XMLHttpRequest
XLink XML Linking Language
XML eXtensible Markup Language

3.4 Terms

The following terms are used within this document.

Application Context Cache – The Application Context 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 Application Context Cache as if it were implemented as actual storage though this is for convenience only. An Application Context Cache corresponds to the Application Context Identifier associated with each Broadcaster Application. Files delivered over ROUTE contain attributes that determine the Application Context Cache with which they are associated.

Application Context Identifier – An Application Context Identifier is a unique URI that determines which resources are provided to an associated Broadcaster Application by the Receiver. Resources may be associated with multiple Application Context Identifiers, but a Broadcaster Application is only associated with a single Application Context Identifier. Details of the Application Context Identifier syntax are specified in the HELD [3].

Base URI – As defined in RFC 3986 [25], the Base URI specifies the initial portion of a URL used by the Broadcaster Application to access files within the Application Context Cache. The Base URI is prepended to the relative URI path of a file to obtain the full URL of the file within the Application Context Cache. The Base URI is uniquely generated by the Receiver based on the Application Context Identifier defined for the Broadcaster Application.

Broadcaster Application – A Broadcaster Application is used herein to refer to the functionality embodied in a collection of files comprised of an HTML5 document, known as the Entry Page and other HTML5, CSS, JavaScript, image and multimedia resources referenced directly or indirectly by that document, all 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 can be delivered over broadcast as packages via the ROUTE protocol.

Entry Package – The Entry Package contains one or more files that comprise the functionality of the Broadcaster Application. The Entry Package includes the Entry Page and perhaps additional supporting files including JavaScript, CSS, image files and other content.

Entry Page – The Entry Page is the initial HTML5 document referenced by application signaling that should be loaded first into the User Agent. The Entry Page is one of the files in the Entry Package.

Event Stream – An Event Stream is a series of messages, either static, in DASH signaling, in MMT signaling, or dynamic, contained in defined messages within media segments. The events contained within the Event Stream can initiate interactive actions on the part of a Broadcaster Application.

Filter Code – See the definition in A/331 [3]. The Filter Code defined on a file is compared with Filter Code Instances set on an Application Context Cache to determine whether the file can be stored in the cache or not.

Filter Code Instance – A Filter Code Instance provides a way for the Broadcaster Application to control which NRT data is stored by the Receiver in its associated Application Context Cache. A Filter Code Instance is a data structure associated with an Application Context Cache with a Filter Code value and an associated expiration time. A Filter Code Instance persists along with an Application Context Cache until its expiration time has been met. The Filter Code Instance expiration can be explicitly set or defaults to the life span of the Broadcaster Application, if not.

MMT Asset – An MMT Asset or Asset is a collection of one or more MPUs with the same Asset ID which is provided in the 'mmpu' box [30].

MMT-Asset File – An MMT Asset file consists of a sequence of MMTP packets, each containing headers and payloads that make up MPUs for different media components. Each component (video, audio, closed captions, etc.) is encoded and packetized into binary data structures that are then multiplexed together within the MMTP packets [30].

Receiver – The Receiver described in this document refers to an entity that implements the functions of the Reference Receiver Model.

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

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

Redistribution – A use case wherein an ATSC 3.0 service is delivered to a Receiver via a protocol other than ATSC 3.0; e.g., HDMI.

Reference Receiver Model – A conceptual receiver device that is capable of executing the APIs and behavior specified in this document. This document specifies normative attributes of the model, which are intended to inform actual receiver implementations.

reserved – Set aside for future use by a Standard.

User Agent – Software provided by the Receiver that retrieves and renders Web content. The User Agent interprets HTML5, CSS, and JavaScript, renders media, text, and graphics, and can create user interaction dialogs.

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 a Broadcaster Application to run. In the broadcast environment, the files associated with a Broadcaster Application are delivered in ROUTE packages that are unpacked into a conceptual cache area. The pages and resources of a Broadcaster Application are then made available to the User Agent associated with the Receiver. In the broadband environment, launching a Broadcaster Application behaves in the same way as in a normal web environment with no specialized behavior or intervention from the 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 Broadcaster Application requires the Receiver to perform a specific action that is not defined by standard W3C User Agent APIs that are widely implemented by browsers, then the Broadcaster Application sends a request to the Receiver WebSocket Server utilizing the set of JSON-RPC messages 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 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 mechanisms
  - 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 media 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 broadcast signaling. In this case, it is the responsibility of the Receiver using its User Agent to launch or terminate the Broadcaster 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
Querying 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 9.2
Requesting to stream downloaded media file from broadcast	Via push or pull model, described in this specification in Sections 9.2 and 9.6.2
Requesting to stream downloaded media file from broadband	Via push or pull model, described in this specification in Sections 9.2 and 9.6.2
Requesting the Receiver Media Player to play a broadband-delivered media stream	Receiver WebSocket Server APIs, described in this specification in Section 9.7.3
Subscribing (or un-subscribing) to stream event notifications that are sent over broadcast	Receiver WebSocket Server APIs, described in this specification in Sections 9.6.1 and 9.6.2
Receiving stream event notifications that are sent over broadcast	Receiver WebSocket Server APIs, described in this specification in Section 9.6.3
Querying the Receiver to learn the identity of the currently selected broadcast service	Receiver WebSocket Server APIs, described in this specification in Section 9.2.3
Receiving notice of changes to user display and presentation preferences	Receiver WebSocket Server APIs, described in this specification in Section 9.3.6
Requesting the Receiver to select a new broadcast service	Receiver WebSocket Server APIs, described in this specification in Section 9.7.1

4.2 Receiver Media Player Display

4.2.1 Rendering Model

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.

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 Broadcaster 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 9.7.2 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 Broadcaster Application is likely 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.

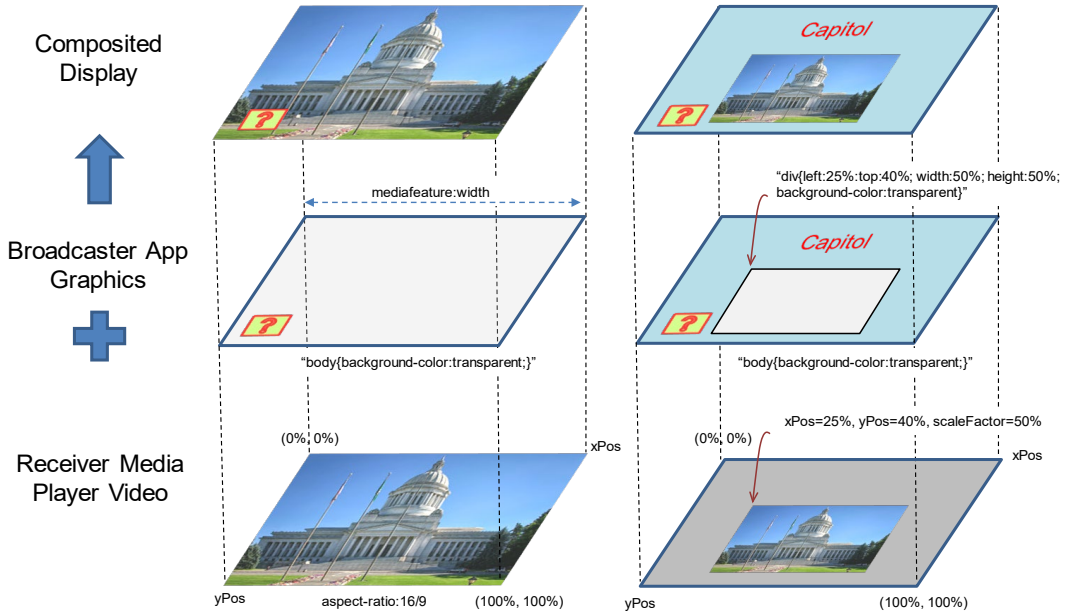


Figure 4.1 Rendering model for application enhancements using RMP.

A Broadcaster Application can expect that the User Agent graphics window, [0,0] to a full 100% in both axes, maps directly to the RMP logical video display window at its full dimensions. Since most Receiver user interfaces may not conveniently enable scroll bar manipulation, the Broadcaster Application should consider disabling scroll bars using standard W3C mechanisms in most situations.

A Receiver may choose to render its own native application on top of the Broadcaster Application due to some user interaction or other similar events. For example, this may happen when the viewer chooses to configure the Receiver settings while a Broadcaster Application is active.

When the Receiver presents its own native application, the Receiver, through standard W3C notification methods, is expected to notify the Broadcaster Application that it no longer has the focus. The Broadcaster Application may choose to either hide itself or maintain its current display. This behavior is left up to design of each Broadcaster Application.

Additionally, the Receiver may choose to hide the launched Broadcaster Application to avoid issues with scaling video and a full-scale Broadcaster Application. The behavior of whether the Broadcaster Application is hidden or not is left up to the Receiver, but the Receiver is not expected

to terminate the Broadcaster Application, as long as the associated service remains selected and application signaling has not selected another Broadcaster Application.

Regardless of whether the Broadcaster Application is hidden or behind a Receiver native application, the Broadcaster Application is notified that it has lost its focus via standard W3C notification methods.

4.2.2 Closed Captioning

Closed captioning is expected to be rendered on top of all video and Broadcaster Application content.

In a worst-case scenario, the captions presented could be opaque, and they could cover a crucial element of the Broadcaster Application, such as an exit button. Three APIs are provided to enable the Broadcaster Application to mitigate being obscured by captions. These are:

- Query Display Components API (Section 9.2.12)
- Video Scaling and Positioning API (Section 9.7.2)
- Graphics Display Regions API (Section 9.7.8)

The Query Display Components API serves two functions. First, it can be used by the Receiver to inform the Broadcaster Application of the capabilities of the Receiver regarding video and caption scaling. For example, consider an L-bar layout scenario. If the Receiver supports video and caption scaling, the captions are expected to exist only on top of the video, and the L-bar would never be obscured. On the other hand, if video and caption scaling are not supported, the Broadcaster Application might need to take measures to avoid being obscured by captions, when present.

The second function of the Query Display Components API is that it can be used by the Receiver to inform the Broadcaster Application of the current area that is being used for captions. The Broadcaster Application can use this information to avoid the area of the display that is occupied by captions. This function is intended for use just prior to the presentation of a call-to-action prompt, when closed captions are enabled. (A call-to-action prompt can be presented by a Broadcaster Application for a limited amount of time to inform the user of the presence of the Broadcaster Application, without any action being taken by the user.) It is not intended for use during normal, longer-term presentation of the Broadcaster Application, since the size and position of the caption region could potentially cause display area conflicts or lead to disruptive changes to the layout of the Broadcaster Application graphics.

The Video Scaling and Positioning API (Section 9.7.2) includes optional information in the response about the minimum scale factor supported by the Receiver. A Receiver might set a relatively large minimum scale factor when captions are present in order to ensure that the captions are not scaled too small to be reasonably legible. If the API returns an error, the Broadcaster Application can confirm that the scale factor and x and y positions are within range and attempt the scaling again.

The Graphics Display Regions API can be used by the Broadcaster Application to inform the Receiver of the areas of the display that include graphical content from the Broadcaster Application. The Receiver can potentially make use of this information to reposition the captions, so as not to obscure the Broadcaster Application graphics.

5. ATSC REFERENCE RECEIVER MODEL

5.1 Introduction

An ATSC 3.0 Reference Receiver Model may be composed of several logical components, which are described in this section. 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 Reference Receiver Model. 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.

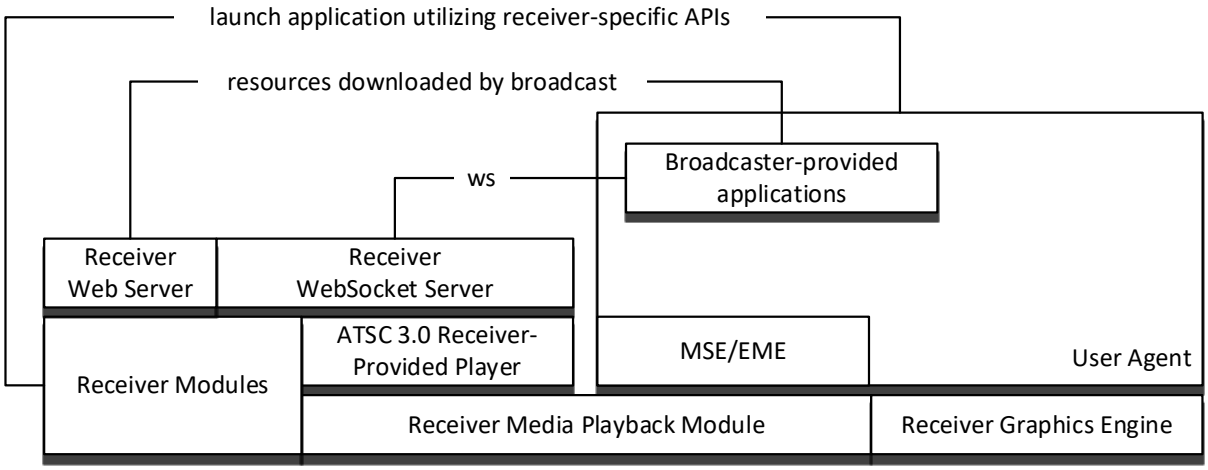


Figure 5.1 ATSC 3.0 Reference Receiver Model Logical Components.

5.2 User Agent Definition

Receivers are expected to implement an HTML5 User Agent that complies with all normative requirements specified in the CTA Web Media API Snapshot (CTA-5000-G) [9]. In addition, the features described in the following sections are expected to be supported.

5.2.1 HTTP Protocols

The User Agent is expected to implement the HTTP protocols specified in RFC 7230 through RFC 7235, references [13], [14], [15], [16], [17] and [18]. User Agents are expected to implement the Web Origin Concept specification [23] and the HTTP State Management Mechanism specification (Cookies) ([9] Section 4.2) as well.

5.2.2 XMLHttpRequest (XHR)

The User Agent is expected to support the `XMLHttpRequest` and related interfaces of the [XHR] reference in [9]. In the case of an XHR request where the request URL identifies a broadcast resource, the request is delivered to the Receiver Web Server, rather than to an Internet web server.

5.2.3 Cross-Origin Resource Sharing (CORS)

The User Agent is expected to support Cross-Origin Resource Sharing as defined in WHATWG Fetch [36].

5.2.4 Mixed Content

The User Agent is expected to 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 [34] though Broadcaster Applications are encouraged to only reference trusted content. References to files within the Application Context Cache (see Section 5.3 below) are considered to be "a priori authenticated" in the terminology of W3C Mixed Content. Any resource accessed from the Application Context Cache is considered to have been accessed within a secure context.

5.2.5 Transparency

The background of the User Agent's drawing window might be transparent by default. Nevertheless, it is recommended that Broadcaster Applications explicitly specify the areas desired to be opaque or transparent to maintain consistency across Receivers. Thus, for example, if any element in the web page (such as a table cell) includes a CSS style attribute "`background-color: transparent`", and that area is not covered by another layer with an opaque element, then video content presented by the Receiver Media Player (see Section 4.2) might be visible in that area. Note that certain areas can be specified as transparent while others are opaque.

5.2.6 Full Screen

As stated in Section 4.2, the Receiver is expected to map the User Agent graphics window, [0,0] to a full 100% in both axes directly to the RMP logical video display window at its full dimensions. The "width" media feature of CSS MediaQueries [9] is expected to align with the width of the RMP logical video display window. In most viewing conditions, the RMP logical video display window is expected to fill the entire screen.

5.2.7 Visibility and Focus

The Receiver is expected to use the W3C Page Visibility Level 2 API as required by CTA-5000-G [9] to inform the Broadcaster Application whether its display output is visible or not. The Receiver may choose to obscure or mute the Broadcaster Application display output for a variety of reasons including but not limited to display of Receiver preference dialogs, content blocking, or other Receiver information. Similarly, the Receiver is expected to provide the W3C Focus Events as required by CTA-5000-G [UIEvents] [9] as well as the standard DOM `activeElement` property to allow the Broadcaster Application to determine if it can receive user input or not.

5.3 Application Context Identifier, Base URI and Cache Path

5.3.1 Application Context Identifier

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 Base URI that converts the relative URL reference to one or more absolute URLs, taking its Application Context Identifier(s) into account.

An Application Context Identifier (`appContextId`) is a unique URI that determines which resources are provided to an associated executing Broadcaster Application by the Receiver. The Application Context Identifier to be bound to the Broadcaster Application is signaled in the HELD [6]. An Application Context Identifier may be associated with many Broadcaster Applications, and the same Broadcaster Application may be associated with many Application Context Identifiers. However, each executing Broadcaster Application is expected to 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. This unique conceptual environment is referred to herein as the Application Context Cache. Thus, each Application Context Identifier uniquely identifies an Application Context Cache.

The Broadcaster Application is expected 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 of the associated resources are expected to continue to be available within the Application Context Cache through the Receiver Web Server. Entry Page changes are signaled by application signaling as described in Section 6.3.

If the Application Context Identifier changes, the Receiver may reuse the Application Context Cache previously created for that Application Context Identifier or a new cache may be created. The Receiver may elect to maintain any previous Application Context Cache, albeit unknown to Broadcaster Applications with differing Application Context Identifiers, on the presumption that these previous Application Context Caches may be needed soon. Alternatively, the Receiver can free the resources associated with the previous Application Context Cache. 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. File caching decisions are left entirely to the Receiver implementation; however, attributes associated with Application Context Cache files are intended to provide prioritization information to the Receiver caching mechanisms (see Section 6.2).

Note that an `appContextId` does not have to be resolvable on the Internet. The domain name root portion of `authority` shall be registered and under the control of one of the signers of the Broadcast Application (author or broadcaster/distributor).

Although the construction of an `appContextId` is required to be globally unique, Receivers should treat it as an opaque string and tolerate any string syntax.

Examples of `appContextId` URIs include:

```
urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
http://kids.pbs.org/app1
urn:tv:nbc.com
```

5.3.2 Origin Considerations

The origin of a web resource is defined in RFC 6454 [23]. While the technical description in RFC 6454 is convoluted to cover the multitude of edge cases, the resultant URIs should be familiar with a `"scheme"` portion (e.g., `"http"`) and an `"authority"`, typically an IP address or hostname, and perhaps a port number (e.g., 10.2.12.45:8080). The algorithm used by an ATSC 3.0 Receiver to generate the portion of a URI that determines the origin of a broadcast file shallis expected to conform to the restrictions specified below. Note that a resource from a broadband source has an origin defined by the web server hosting the resource.

5.3.3 Base URI

The Base URI for the Application Context Cache is for broadcast resources, Broadcaster Application resources, and broadband resources accessed through the Cache Request APIs of Section 9.3.14 only. Broadband resource caching from HTML5 fetch() [36] is separate and subject to the Receiver's Web Server User Agent caching policy.

The Receiver is expected to implement an obfuscation function (e.g., SHA1) in the creation of the Base URI in a manner that is statistically unique globally. The obfuscation function is expected to combine `appContextId` with some device-specific information such that knowledge of the obfuscation function alone would not be sufficient to allow a Broadcaster Application or external entity to do any of the following:

- Recovering the `appContextId` from the Base URI
- Creating the same Base URI from an `appContextId`
- Using a Base URI from one Receiver to provide access to the same resources on another Receiver

Broadcast resources can be shared in multiple caches using the ROUTE EFDT signaling. The above construction allows either a broadcast- or broadband-delivered Broadcaster Application to access the same local storage information over multiple services.

For example, in a multipart package:

Service 1 contains:

```
EFDT.FDT-Instance.File@Content-Location="package1"
EFDT.FDT-Instance@afdt:appContextIdList="http://kids.pbs.org/app1"
```

Where there is a resource with a multipart boundary, `"Content-Location: folder1/file1.txt"`.

Service 2 contains:

```
EFDT.FDT-Instance.File@Content-Location="package2"
EFDT.FDT-Instance@afdt:appContextIdList="http://kids.pbs.org/app2"
```

Where there is a resource with a multipart boundary, `"Content-Location: folder1/file2.txt"`.

Service 3 contains:

```
EFDT.FDT-Instance.File@Content-Location="package3"
EFDT.FDT-Instance@afdt:appContextIdList="http://kids.pbs.org/app1
   http://kids.pbs.org/app2"
```

Where there is a resource with a multipart boundary, `"Content-Location: folder2/file3.txt"`.

Then the cache for `app1` contains both `folder1/file1.txt` and `folder2/file3.txt`, and the cache for `app2` contains both `folder1/file2.txt` and `folder2/file3.txt`.

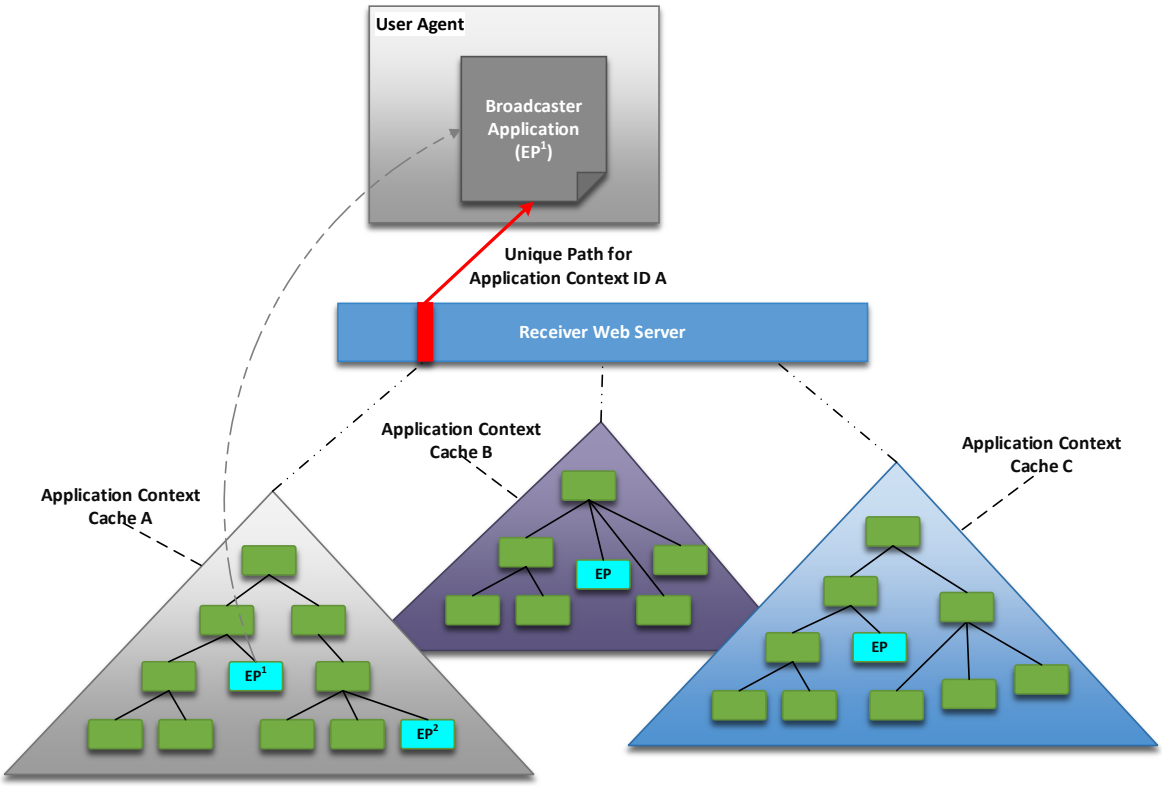


Figure 5.2 Application Context Identifier Conceptual Model.

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, $\boldsymbol{EP^1}$. At some point while $\boldsymbol{EP^1}$ is active, application signaling could launch the Entry Page designated as $\boldsymbol{EP^2}$. In this case, the Application Context Cache A and access to it would remain constant with the User Agent loaded with $\boldsymbol{EP^2}$. The Receiver may or may not provide access to the other Application Context Caches corresponding to different Application Context Identifiers. Broadcaster Applications should restrict access to resources within their own Application Context Cache 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 extended caching capabilities to maintain resources across tuning events. This allows broad flexibility in delivering resources on multiple broadcasts for related Broadcaster Applications.

6. BROADCASTER 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.

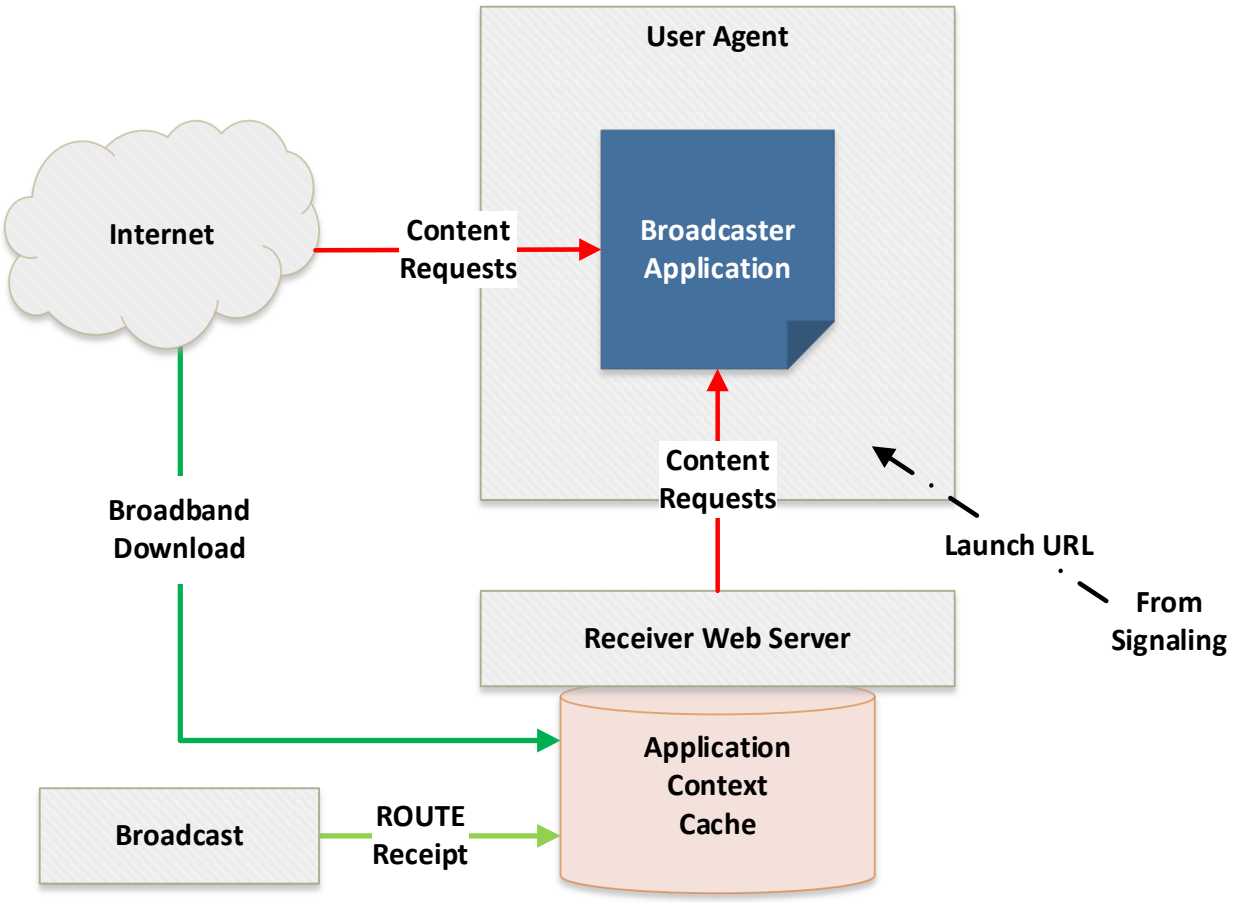


Figure 6.1 Receiver Conceptual Architecture.

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 Broadcaster Application Entry Page from the URL. Note that the URL may point to an Internet server or to the Receiver Web Server depending on how it is formatted in the service application signaling, specifically, the `HTMLEntryPackage@bcastEntryPageUrl` or `HTMLEntryPackage@bbandEntryPageUrl` attributes of the HELD [6]. The specific mechanism of communicating the Broadcaster Application entry URL to the User Agent is a Receiver implementation detail. However, the entry URL has specific arguments that must be provided as described in Section 8.2.

Once the main Broadcaster Application Entry Page has been loaded, it may begin requesting content from various local or external URLs. This may be done through JavaScript or standard HTML5 `href` requests in the W3C-compliant fashion. It is assumed that any content received over broadcast via ROUTE file delivery is available through the Application Context 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 HTTP 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. The internal W3C-compatible storage mechanisms implemented within the User Agent should not be confused with the Application Context Cache shown separately in Figure 6.1. The Broadcaster Application can use standard W3C interfaces to discover and use the various User Agent storage facilities.

6.2 Application Context Cache Management

6.2.1 Signaling Intent for File Caching

All files delivered over broadcast to the Application Context Cache are carried in multipart/signed packages as described in A/331 [3] and required by Section 6.5.3 of this standard. From the ROUTE standpoint, each package is an opaque file object so the subsequent `File` elements within the `FDT-Instance` element of the `EFDT` describe only the multipart/signed package object.

A main header is defined within the multipart/signed package that describes various parameters including the necessary boundary text that delineates files within the package. The file data resides within these boundary-separated blocks which, in turn, include header elements, referred to herein as a boundary header, prior to the individual file data, that can provide metadata specific to the file within the package block.

To provide a manifest for files contained within the package, a `metadataEnvelope,` as defined in A/331 Section 7.1.6, shall be included as the first object in the package. Content shall not be embedded in the `metadataEnvelope`; the "referenced" mode shall be used.

Within the `metadataEnvelope` fragment, a `metadataEnvelope.item` element shall be present corresponding to each file within the package. The `metadataEnvelope.item` attributes shall be interpreted as follows:

- The required `@metadataURI` attribute shall provide the relative path of the file referenced by this `metadataEnvelope.item`. The URI value shall match the relative path supplied in the `Content-Location` parameter included as part of the boundary header for the referenced file. This is expected to provide the relative path of the file within the Application Context Cache. The `EFDT.FDT-Instance.File@Content-Location` values,

normally used to set the relative path for non-multipart resources, shall be ignored when the resource is multipart.

- The `@version` attribute increments when a new version of the referenced file has been provided in the package. The Receiver is expected to rely on the `@validFrom` attribute when detecting file version changes and can safely ignore the `@version` attribute.
- The `@validFrom` attribute shall be required and shall indicate when the referenced file was last modified. A new version of a file shall be signaled by updating this time stamp within the `metadataEnvelope.item` associated with the file. This value is expected to be provided as the `Last-Modified` HTTP header parameter supplied when accessing the file through the Receiver Web Server unless the boundary header contains an `ATSC-HTTP-Attributes` parameter as described in Section 6.2.1.1 that overrides this default. This value is expected to be used when calculating the age of the file.
- The date and time values supplied in the optional `@validUntil` attribute are expected to be used to indicate when the file is no longer needed and can be released from the Application Context Cache. The Broadcaster Application can read the expiration time of the file using the `Expires` HTTP parameter supplied when accessing the file through the Receiver Web Server unless the boundary header contains an `ATSC-HTTP-Attributes` parameter as described in Section 6.2.1.1 that overrides this default.
- The required `@contentType` attribute shall provide the MIME type of the referenced file. This attribute shall match the `Content-Type` value defined as part of the boundary header within the package, if provided. Note that the `EFDT.FDT-Instance.File` also contains a `Content-Type` definition, but this should always be `multipart/signed` indicative of the referenced package object. This value may be accessed through the `Content-Type` HTTP header parameter supplied when accessing the file through the Receiver Web Server.

This specification defines an additional attribute that extends the `metadataEnvelope.item` specification defined in A/331 [3]. Table 6.1 provides an informative definition of the ATSC extended attribute when included within a signed package destined for the Application Context Cache. The normative semantics of the attribute are provided below the table.

Table 6.1 ATSC-Defined Extension to the `metadataEnvelope.item` Element

Attribute Name	Cardinality	Data Type	Description
`@contentLength`	0..1	long	Provides the length in bytes of the referenced file. This value may be accessed through the `Content-Length` HTTP attribute in a response to a User Agent request.

`@contentLength` – The optional `@contentLength` attribute shall define the length in bytes of the referenced file within the package. When defined, the length value shall be made available as part of the `Content-Length` HTTP header element provided by the Receiver Web Server when the referenced file is accessed.

6.2.1.1 Boundary Header HTTP Attribute Definition

A boundary header element, `ATSC-HTTP-Attributes`, may optionally be supplied in the boundary header of a file within the multipart/signed package. This element provides a list of HTTP header elements that are expected to be provided whenever the associated file is requested through the Receiver Web Server. The syntax of this attribute shall be defined as follows:

```
attributes := "ATSC-HTTP-Attributes" ":" parameter[";" parameter]*
```

where each `parameter` is an HTTP header field as described by RFC 7230 [16] except that the colon, ":", used by the HTTP header field shall be replaced with an equals sign, "=", to comply with constraints imposed by the multipart standard, RFC 2045 [20].

6.2.2 Application Context Cache Hierarchy Definition

All interactive content is carried in signed packages of files and transmitted via ROUTE (Section 6.5). All signed packages whose application signaling denotes a particular Application Context Identifier are expected to be provided in a single hierarchy accessible to the Broadcast Application with a unique Base URI. The Base URI concept and its semantics are described in Section 5.3.

The choice of whether a package or file is immediately stored to the Application Context 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 a Receiver implementation decision. 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` is returned indicating that some error condition occurred [18]. Typically, this is 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 not contained within their Entry Package.

Similarly, the present document does not specify how frequently packages needed for the Broadcaster Application are transmitted nor how frequently application signaling metadata is sent. These decisions depend on several factors such as what time the Broadcaster Application functionality is actually needed during content viewing, 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 are likely different for every Broadcaster Application depending on its overall purpose. The Application Signaling standard, A/331 [3], requires broadcast packages to be available when the HELD signals the Broadcaster Application. In addition, A/331 also defines a Distribution Window Description fragment (DWD) that provides a mechanism to signal when packages will be available in the broadcast at times other than when the Broadcast Application is signaled in the HELD.

The broadcaster shall be responsible for defining and managing any hierarchy below the Application Context Cache root directory through use of the directory and path mechanisms described in Section 6.2.1 above. Any hierarchy below the Application Context Cache Base URI level is up to the broadcaster to define.

An example of how such a hierarchy could be defined is described below. Figure 6.2 shows an example of such a hierarchy for **`BaseURI B`**.

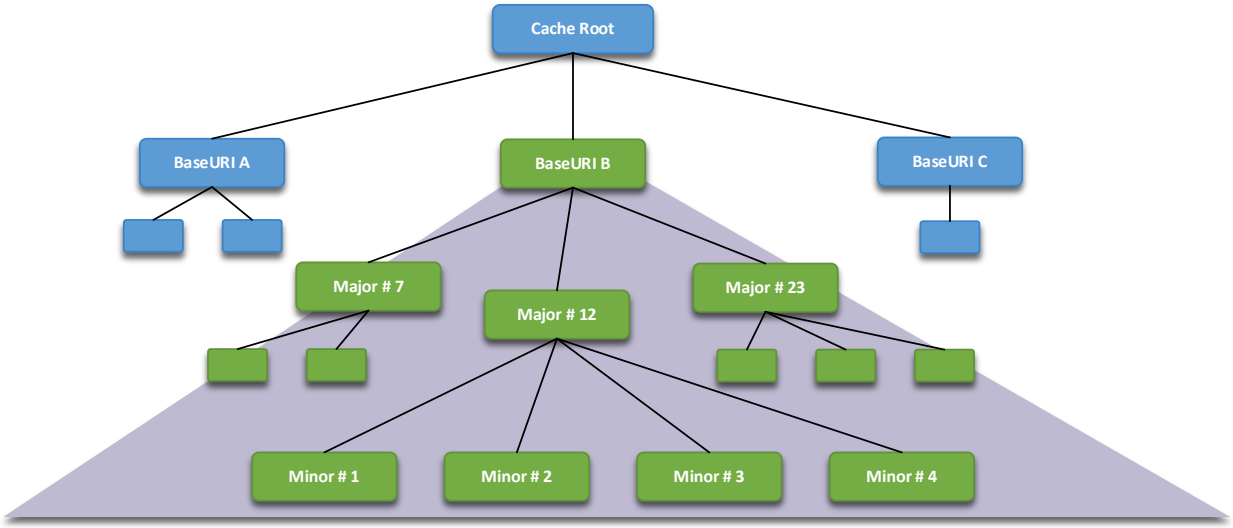


Figure 6.2 Example Application Context Cache Hierachy.

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 Entry Page is called `"main.html"`. To launch the Broadcaster Application, the application signaling in this example provides the relative URI of `"12/2/main.html"` in the `HTMLEntryPackage@bcastEntryPageUrl` attribute of the HELD [6]. 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:

<BaseURI>`/12/2/main.html`

Note that the Receiver-created **<BaseURI>** portion of the URL is highlighted as bold. The Broadcaster Application would be able to reference any directory corresponding to its `AppContextID` hierarchy as designated by the green boxes in Figure 6.2 under the "BaseURI B" box.

The Receiver provides the Base URI information associated with the current Application Context Identifier to the Broadcaster Application through a Receiver WebSocket Server API (see Section 9.2.7). Note that this Base URI is available through the normal W3C document object model (DOM) if the Broadcaster Application is sourced from broadcast. In this case, the Broadcaster Application can simply access content using relative URLs or constructing full URLs using `Document.location`. The WebSocket API is most applicable for Broadcaster Applications hosted from broadband allowing them to gain access to resources received over broadcast.

6.2.3 Active Service Application Context Cache Priority

Once the packaged resources, known as the Entry Package, of the Broadcaster Application (see Section 6.3) have been acquired and placed in the Application Context Cache, the Broadcaster Application can assume that the Entry Package resources, that is, those files delivered along with the Broadcaster Application Entry Page in a single package, are expected to remain available as long as the Broadcaster Application is loaded in the User Agent. If the Receiver cannot provide the entire Entry Package to the Broadcaster Application reliably, the Broadcaster Application is not expected to be launched. This allows broadcasters to determine the required files of their Broadcaster Application and send them all as one package thereby guaranteeing that those files

are available if the Broadcaster Application has been launched. In addition, Receivers are expected to make every effort to cache files delivered with a particular service for the Broadcaster Application(s) associated with the current service Entry Package.

Note that the Receiver may have to release 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 assumes that the current service content preempts all other content from the user perspective.

A Broadcaster Application may limit the amount of cache required by using the Filter Codes mechanism. Filter Codes provide a way to differentiate application content allowing a Broadcaster Application to indicate which packages it wishes to be placed in the Application Context Cache and which packages it wishes to ignore. A set of WebSocket APIs is provided to allow the Broadcaster Application to manage the Filter Codes (see Section 9.10).

A Broadcaster Application can mark content as unused to hint to a Receiver that the resources are no longer needed. A WebSocket API is provided (see Section 9.8) that allows files or entire directory hierarchies to be marked as unused. Subsequent access to resources that are marked as unused is expected to result in an error. The HELD (A/331 [3]) also provides an attribute, `@clearAppContextCacheDate`, associated with the Broadcaster Application that indicates that any file in the Application Context Cache created before the specified date and time should be removed. This enables older files present in an Application Context Cache to be marked as obsolete.

6.2.4 Cache Expiration Time

The `@validUntil` attribute of a file, as defined in Section 6.2.1, shall indicate that the broadcaster expects the file to remain in the Application Context Cache until the expiration time has been reached regardless of whether the Broadcaster Application is currently loaded in the User Agent. However, the storage requirements of the loaded Broadcaster Application take precedence over the `@validUntil` attribute, so the Receiver may be forced to release files from other services prior to their `@validUntil` time to provide storage to the current service. The Broadcaster Application must be prepared to deal with the possibility that a resource may not be available. Receivers may elect to leave the resource within the Application Context Cache or purge it whenever is convenient after the `@validUntil` date and time has been passed, depending on other cache management criteria.

Note that for resources that are part of the Entry Package, the Broadcaster Application can assume that those files are available at startup and remain available if the Broadcaster Application is running regardless of the `@validUntil` time as described in Section 6.2.3.

6.2.5 Advanced Emergency Alert Enhancement Content Considerations

The AEAT may reference AEA enhancement content through URLs. This content is delivered either in the broadcast as a separate ROUTE stream, in which case the referencing URL will be relative, or over broadband where a fully qualified URL will be provided.

In the broadcast case, the broadcaster is responsible for providing the Application Context Id of any Broadcaster Applications that need to access the AEA enhancement content as part of the EFDT fragment describing the signed package containing the AEA enhancement content. The Receiver is expected to treat this content as it would normal ROUTE-delivered content by making it available in the Application Context Caches corresponding to the listed Application Context IDs. Note that since the AEA enhancement content is expected to occupy the same hierarchy as other

ROUTE-delivered data, the broadcaster must ensure that no conflicts occur when defining the corresponding file hierarchies.

Similarly, any cache management performed by the Broadcaster Application should take the life span of the AEA event into account when signaling `@clearAppContextCacheDate` or using the Mark Unused API (see Section 9.8) to indicate the content is no longer needed. Any broadcast NRT data associated with an AEA event is expected to have a `@validUntil` attribute defined to be at least the duration of the AEA event.

6.3 Broadcaster Application Lifecycle

This section prescribes the Broadcaster Application life cycle. For syntax and semantics of the HELD, see A/331 [3] Section 7.1.8. See Annex A for an informative description of the Broadcaster Application lifecycle. This lifecycle applies to all `SLT.Service@serviceCategory` values.

The term, "load" in its various forms, means the resources necessary for execution have been brought into Receiver memory and execution is started or continued. There is no dependency on `serviceCategory`.

The Broadcaster Application lifecycle is initially indicated by the presence or absence of a valid HELD and the entries in it. When a Broadcaster Application is part of a service, the HELD shall be present in all instances of the SLS for that service at least whenever the HELD is valid. A HELD is "valid" if its syntax is conformant, and the current UTC time is not before `@validFrom` (if present), and not after `@validUntil` (if present), for at least one entry in the HELD. In the case of a HELD obtained using the content recovery methods described in ATSC A/336, the current presentation time on the Recovery Media Timeline [5] is expected to be used instead of the current UTC time for evaluating `@validFrom` and `@validUntil`.

Upon receipt of a HELD, the Receiver is expected to build a list of candidate Broadcaster Applications as indicated by one or more entries in a HELD. The Receiver is expected to exclude all entries which indicate a `@validUntil` in the past and exclude all entries which indicate a `@validFrom` in the future. Similarly, the Receiver is expected to exclude all entries which indicate `@requiredCapabilities` which are not satisfied by the Receiver's capabilities. With this filtered list, the Receiver is expected to identify a single HELD entry identifying the Broadcaster Application to run:

- If the list has an entry with the same `@appId` and `@appContextId` of the Broadcaster Application that is currently loaded, then that entry is chosen, and the current Broadcaster Application is allowed to continue execution (no "re-loading" is necessary).
- If the list has an entry that matches a user-selected `@appId` and `@appContextId`, then that entry is chosen and allowed to continue execution (no "re-loading" is necessary).
- If the list has exactly one entry, or if exactly one entry is indicated as `@default=True` and none of the preceding conditions apply, then that entry is the candidate to be loaded.
- If this list of entries is empty, the Receiver is expected to unload the current Entry Page if a Broadcaster Application is currently loaded.
- If this list of entries has more than one candidate, and none of them are marked as default, the behavior is undefined.

A Broadcaster Application is "loaded" when the Entry Package is completely available within the Application Context Cache. For external broadband references where a full URL is provided, the URL supplied in the `HTMLEntryPackage@bbandEntryUrl` attribute for the Broadcaster Application section of the HELD is expected to be launched directly.

If a Broadcaster Application is currently loaded and a service change is initiated (either by the Receiver or the Broadcaster Application), the Receiver may request specific resources to be released by the Broadcaster Application prior to the Receiver beginning channel change processing by using the Prepare for Service Change API (see Section 9.16).

If a Broadcaster Application is currently loaded, and the candidate entry has identical values of `@appContextId` and `@appId`, then the currently loaded Broadcaster Application is the same as the candidate entry, the running Broadcaster Application persists, and receives a Service Change notification if it has requested one.

Similarly, if a Broadcaster Application is currently loaded, and the `@validUntil` property for that Broadcaster Application has been reached, then the Receiver is expected to unload the current Broadcaster Application and proceed as below.

If a Broadcaster Application is currently loaded, and the Broadcaster Application is different from the candidate entry, then the Receiver is expected to unload the current Entry Page and load the candidate entry Broadcaster Application's Entry Page.

If there is no Broadcaster Application currently loaded, the Receiver is expected to load the candidate entry Broadcaster Application's Entry Page.

Note that logic within a given Entry Page may load and unload sub-pages as described in above; such actions are Broadcaster Application-specific and are not governed by the HELD.

When the HELD is no longer present in the SLS or no longer valid, the Receiver is expected to unload the current Entry Page.

When a Broadcaster Application is executing, if the user or Broadcaster Application requests another service with a common `@appContextId` and, if present, `@appId` then the Receiver is expected to issue a Service Change Notification (see Section 9.3.3) event with `"requested"="true"`. In that case, the Broadcaster Application should promptly:

1) Cease all AMP activity and release all AMP-related resources, and
2) If the Set RMP URL API (see 9.7.3) is actively using the RMP, issue a Set RMP URL API request with `"operation"="stopRmp"`,
3) Clear all broadcast program-specific information,
4) Reset video scaling and positioning (see 9.7.2), and
5) Begin requesting required resources from the Application Context Cache.

After the requested service has been acquired, when the requested service has a common `@appContextId` and, if present, `@appId`, then it is expected that the Receiver will issue a Service Change Notification (see Section 9.3.3) with `"requested" = "false"`.

If the user or Broadcaster Application requests another service with different `@appContextId` and, if present, `@appId`, no Service Change Notification event is fired, and the Broadcaster Application is terminated.

In addition to managing Broadcaster Application changes across services, since the HELD is capable of signaling multiple Broadcaster Applications, it is possible for a Broadcaster Application to initiate a switch to a new Broadcaster Application on the same service by using the Launch Broadcaster Application API (see Section 9.7.6). This can be in the same or different Application Contexts.

If the new Broadcaster Application is valid and its resources are available, then the calling Broadcaster Application is terminated and the new Broadcaster Application is loaded. The calling Broadcaster Application should release notifications, subscriptions and all other resources before calling the Launch Broadcaster Application API. If the `appContextId` is different between the

Broadcaster Applications, then the starting Broadcaster Application is not expected to have access to the calling Broadcaster Application resources.

6.4 Broadcaster Application Events (Static / Dynamic)

Actions to be taken by Broadcaster Applications can be initiated by notifications delivered via broadcast or broadband or, in a redistribution setting, via watermarks. A/337 [6] uses the term "Events" for such notifications.

Broadcast delivery of events is defined in Section 4.1 of A/337 [6] including delivery for ROUTE/DASH-based services and MMT-based services.

Broadband delivery of events is defined in Sections 4.2 and 4.5 of A/337 [6] including delivery for ROUTE/DASH-based services and MMT-based services.

Both the broadcast and broadband delivery of events defined in A/337 [6] support batch (static) and incremental (dynamic) delivery.

In a redistribution setting, events can be also delivered via video and audio watermarks as described in Section 4.3 of A/337 [6].

Detailed specification of the WebSocket APIs used to register for and receive event stream notifications is provided in Section 9.6.

6.5 Broadcaster Application Delivery

The file delivery mechanism of ROUTE, described in A/331 [3], 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. Furthermore, the application signaling [6] determines the source of the Broadcast Application Entry Page, and the location of any other files and packages that are delivered by broadcast.

1) A relative Entry Page URI indicates that the source of the Broadcaster Application Entry Page is broadcast ROUTE data.
2) An absolute Entry Page URL indicates that the Broadcaster Application Entry Page should be sourced from broadband.
3) If any files or packages are delivered by broadcast, the application signaling identifies the LCT channels that are used to deliver the files and/or packages.

Note that while the initial Entry Page may be sourced from either broadband or broadcast according to the `HTMLEntryPackge@bbandEntryPageUrl` or `HTMLEntryPackage@bcastEntryPageUrl` attributes in the HELD, there is no constraint regarding using either broadcast or broadband resources within the Broadcaster Application itself. Hybrid delivery of Broadcaster Application files is allowed and expected.

6.5.1 Broadcaster Application Packages

The file components comprising the Broadcaster Application shall be delivered over broadcast within one or more multi-part MIME packages using ROUTE or over broadband as individual files using HTTPS. All files made available through the Receiver Web Server shall be delivered to the Receiver as signed packages as described in A/331 [1].

It is not required that all resources used by a Broadcaster Application be delivered in a single ROUTE package when delivered over broadcast. The broadcaster may choose to send a relatively small Entry Page in a signed Entry Package 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 Entry Package containing the Entry Page is expected to be received in its entirety before launching the Broadcaster Application (per Section 6.2.3), 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. Control for selecting specific signed packages to be made available is provided using the Filter Codes API (see Section 9.10).

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 and may not have a broadband connection.

6.5.2 Broadcaster Application Package Changes

Broadcaster Application resource files and packages may be updated at any time. Mechanisms for determining that a new file or package is being delivered are defined in FLUTE, which is the underlying standard used by ROUTE [3]. 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.

6.5.3 Content Caching Control via Filter Codes

Filter Codes provide a way for the Broadcaster Application to control which NRT data is stored in the associated Application Context Cache. The broadcast must include all NRT data for all possible instances of the Broadcaster Application executing on a variety of Receivers; however, an individual Broadcaster Application may only need certain NRT data depending on the service, user, Receiver instance or some other constraint.

A Filter Code Instance is a value and optional expiration time associated with an Application Context Cache. A Filter Code Instance persists along with an Application Context Cache until its expiration time is exceeded or for the life span of the Broadcaster Application, if no expiration time was specified. If the Receiver reclaims the resources of an Application Context Cache, any associated, unexpired Filter Code Instances are also expected to be reclaimed. Once reclaimed, neither the Application Context Cache nor the associated Filter Code Instances are expected to be defined, regardless of any Filter Code Instance expiration time.

To control the NRT data storage using Filter Codes, it must be clear how a Receiver is expected to process the incoming signaling and NRT data into the Application Context Cache so that the broadcast and Broadcaster Application can be engineered in the most effective way. Figure 6.3 provides a flowchart diagram of the conceptual process a Receiver is expected to use to place content into the Application Context Cache based on the NRT data and the Filter Code Instances. Note that the Broadcaster Application running state is also depicted since its launch relies on the receipt of a non-filtered NRT data package and Filter Code Instances may not exist until the Broadcaster Application has defined them.

For the purposes of this discussion, it is assumed that the Broadcaster Application is intended to execute from the local Application Context Cache and that all data is received via ROUTE streams within the broadcast instead of from broadband sources. Note that while Filter Code

processing may still be used to filter selected NRT data from the broadcast stream, broadband sources of both the Broadcast Application and needed data may obviate some of the decisions shown in the flowchart diagram of Figure 6.3.

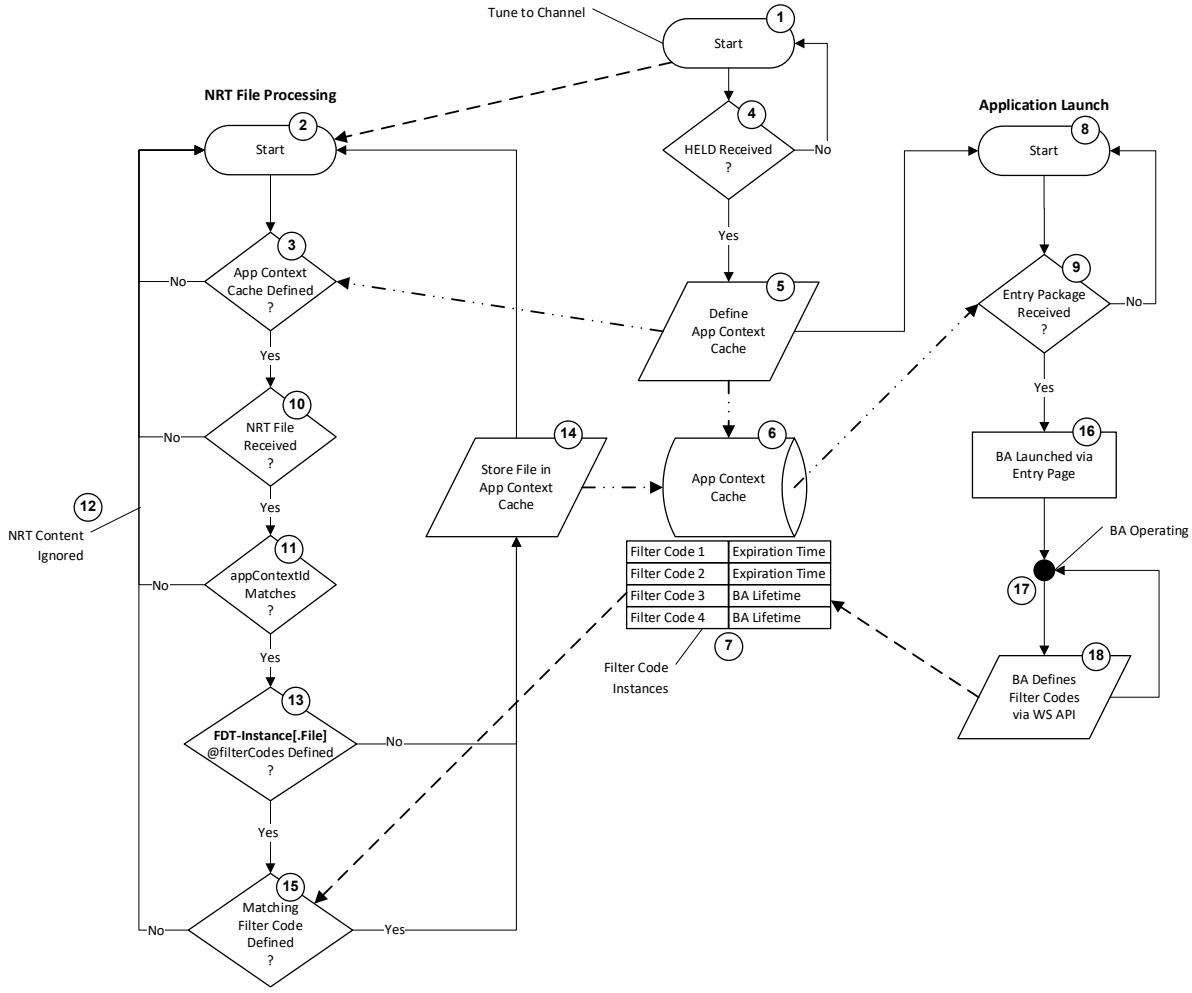


Figure 6.3 Filter Code Processing Flowchart.

The following description is based on the annotated steps in Figure 6.3 above.

The processing starts, (1), when the Receiver is tuned to a channel. As part of the initial discovery process, Service Level Signaling (SLS) associated with the selected service may indicate that one or more ROUTE sessions are available carrying NRT data. The presence of ROUTE NRT data streams launches the NRT File Processing, (2), depicted at the left side of Figure 6.3. For NRT content destined for an Application Context Cache, an Application Context Cache must be defined before any content may be stored there (3). The Application Context Cache may already be present if the Receiver has not reclaimed its resources, but this is dependent on many factors including the time between channel selections and the amount of resources available. Note that other NRT data may be received for other purposes but those use cases are beyond the scope of this discussion.

As part of the SLS processing, the service signaling may include a HELD table [3] indicating that a Broadcaster Application is associated with the selected service. When the HELD is received, (4), it is processed to determine the Application Context ID defined by **HELD.HTMLEntryPackage**@appContextId [3]. The Application Context Cache (6) is then allocated, (5), where the associated NRT data can be stored. Again, the Application Context Cache may already exist if a previous Broadcaster Application used the same Application Context Identifier and the Receiver has not yet reclaimed the associated resources.

Similarly, Filter Code Instances may also be associated with the Application Context Cache if they were set by a previous Broadcast Application instance with an expiration time that has not yet occurred. These Filter Code Instances are represented in Figure 6.3 as "Filter Code 1" and "Filter Code 2" with associated Expiration Times annotated with a (7). Filter Code Instances with Filter Codes 3 and 4 in the diagram are described below.

The HELD table also causes the Application Launch process to start (8) via the **HELD.HTMLEntryPackage**@bcastEntryPageUrl [3]. The Receiver must then wait (9) for the Entry Package to be received and the files made available in the Application Context Cache before launching the Broadcaster Application. Note that if the Application Context Cache is still present and contains the appropriate Entry Package, the Broadcaster Application could start immediately.

While the Receiver is waiting for the Entry Package to be available (9), the NRT File Processing logic is waiting for a file to be received (10). Note that the process of receiving a file is complicated and depends on signaling within the ROUTE stream. It is likely the case that each portion of a file will be examined based on the tests shown in the diagram but for simplicity the flowchart shows the processing at the file level.

When a file is received, it is first checked, (11), to see if it should be saved in the Application Context Cache, as indicated by the presence of the matching appContextId in the **FDT-Instance**@appContextIdList or **FDT-Instance.File**@appContextIdList [3]. If not, then it is ignored, and the processing continues (12).

If an appContextId does match, then the Receiver checks the file for the presence of Filter Codes (13). Filter Codes are defined either overall for all files in the source flow in the **FDT-Instance**@filterCodes list attribute or specifically for a file in the **FDT-Instance.File**@filterCodes list attribute [3]. If after processing the Filter Code list attributes there are no Filter Codes assigned to the file, then the file is stored (14) in the Application Context Cache (6). Note that the **File**@filterCodes list preempts the **FDTInstance**@filterCodes list. If any filter codes are defined on a file, then the processing moves to test whether any Filter Code Instances have been set (15).

It is expected that the initial Entry Package will not have any filter codes defined in the ROUTE signaling. This results in the Entry Package always being stored in the Application Context Cache once received. The Application Launch process waits (8) for the Entry Package to be received and then launches, (16), the Broadcaster Application using the **HELD.HTMLEntryPackage**@bcastEntryPageUrl [3].

The Broadcaster Application begins execution (17) eventually issuing the Set Filter Code Instances API (Section 9.10.1) to define which NRT files it requires (18). In Figure 6.3, new Filter Code Instances defined by the Broadcaster Application are shown (7). Filter Code Instances containing "Filter Code 3" and "Filter Code 4" have an expiration labeled "BA Lifetime" indicating that they are expected to expire when the Broadcaster Application terminates.

Once one or more Filter Code Instances are defined, (7), the check for a match between the Filter Codes signaled on the NRT files and the Filter Code Instances may succeed (15) resulting

in the corresponding file being saved (14) to the Application Context Cache (6). Specifically, the test logic is as follows:

if `FDT-Instance.File@filterCodes` exists and the intersection of the Filter Code Instance values and the `FDT-Instance.File@filterCodes` list is not empty, then save the file to the Application Context Cache, otherwise,

if `FDT-Instance@filterCodes` exists and the intersection of the Filter Code Instance values and the `FDT-Instance@filterCodes` list is not empty, then save the file to the Application Context Cache, otherwise,

ignore the file.

Note that the `FDT-Instance.File@filterCodes` are checked first since A/331 requires that they preempt any Filter Codes set on the overall FDT-Instance, namely, `FDT-Instance@filterCodes` (see A/331 Section A.3.3.2.3 [3]). This means it is possible to remove Filter Codes defined by `FDT-Instance@filterCodes` from a specific file by providing an empty `FDT-Instance.File@filterCodes` attribute.

While the Broadcaster Application and NRT File Processing continue operating, new Filter Codes can arrive with the NRT files, Filter Code Instances may expire, and the Broadcaster Application can define new Filter Code Instances. The NRT File Processing loop continuously operates on received files matching the current `appContextId` storing those files without Filter Codes signaled as well as files with at least one Filter Code set that matches at least one Filter Code Instance value. All other files are expected to be ignored.

Note that the Receiver is not expected to keep track of files in the Application Context Cache that were stored based on Filter Codes. In other words, if the Broadcaster Application changes its Filter Code Instances, the Receiver is not expected to purge files from the cache that would no longer match the currently defined Filter Code Instances. If such files are no longer needed by the Broadcaster Application, it may use the Mark Unused API (Section 9.8) to designate the files as unnecessary.

6.6 Security Considerations

All Broadcaster Application files delivered over broadcast shall be delivered using the ROUTE Signed Package mechanism described in A/331 Section A.3.3.5 [3] and A/360 Section 5.2 [8]. A/331 describes the encapsulation of Broadcaster Application files in MIME multipart packages while A/360 describes the encapsulation of that MIME multipart package into an S/MIME wrapper to secure the Broadcaster Application files. Files are deemed to be part of the Broadcaster Application if they are accessible from the Application Context Cache through the Receiver Web Server interface. For Receivers that support signing, it is expected that they only make content from correctly signed packages available through the Receiver Web Server interface.

The Broadcaster Application files may be delivered over broadcast in as many signed packages as desired – there is no restriction on dividing files among signed packages. In fact, it may be typical that core functions of the Broadcaster Application are delivered in the Entry Package while extended content and functionality are delivered in separate packages. In addition, the Filter Codes mechanisms (see Section 9.10) can be used to select various packages as user preferences or other selection criteria are discovered. Regardless of how files are partitioned into separate packages, these packages must be signed per the requirement in the previous paragraph.

Broadcaster Application files delivered over broadband shall be secured using standard W3C mechanisms. All connections to broadband servers shall use a secure connection as described in

A/360 Section 5.1 [8]. The content received over broadband using a secure connection shall be considered trusted as if it had been received in a signed package over broadcast.

6.7 Companion Device Interactions

The ATSC 3.0 Companion Device standard (A/338 [37]) specifies how a separate device, known as the Companion Device (CD), interacts with the Receiver, known as the Primary Device (PD) in the A/338 standard. The A/338 standard extends the APIs defined in Sections 8 and 9 of the present standard to provide CD Manager APIs that allow the Broadcaster Application to discover and launch CD Applications operating on the Companion Device. These CD Manager APIs also provide a mechanism for the Broadcaster Application to obtain WebSocket service end points that allow application-to-application communication between the CD Applications and the Broadcaster Application. The Broadcaster Application may support multiple connections by requesting multiple end points. To use the CD Manager APIs, the Broadcaster Application can obtain the WebSocket URL with the mechanism defined in Section 8.2.1.

Complete details of the Companion Device Interactions regarding the discovery and launch of Companion Device applications, application-to-application WebSocket communication mechanisms, and Receiver WebSocket APIs available to the CD Application can be found in the ATSC 3.0 Companion Device standard (A/338 [37]).

7. MEDIA PLAYER

In the ATSC 3.0 Receiver environment, there are two software components that can play out media content delivered via either broadcast, broadband, or redistribution. For the purposes of this specification, these two logical components are referred to as Application Media Player (AMP) and Receiver Media Player (RMP), and these are described further in this section. The AMP is JavaScript code (e.g., DASH.js), which is part of an HTML5 Broadcaster Application, while the RMP is a Receiver-specific implementation. The AMP uses the HTML5 `<video>` tag and MSE to play out media content regardless of the content origination or delivery path. Details of the RMP design and implementation are out of scope for this specification and any design descriptions provided in this specification are only as informative reference. Whether AMP or RMP is used to play out a media content, there are several use cases:

- **Broadcast or Hybrid Broadband / Broadcast Live Streaming** – The content segments arrive either via broadband or broadcast.
- **Broadband Media Streaming** – Media content streaming over broadband (on-demand or linear service).
- **Downloaded Media Content** – Media content downloaded over broadcast or broadband ahead of time. Details of how media content is downloaded over broadband or broadcast is described in Section 9.3.14 of this specification.

The type of media streams played depends on signaling in the MPD of live broadcast streams, or specific Broadcaster Application logic.

The DASH Client specification [41] provides the expectations for behavior of such players and is not further described here.

In redistribution scenarios, media content arrives using a method that does not employ ATSC 3.0 protocols (e.g., HDMI). Redistribution services are presented by an RMP.

7.1 Utilizing RMP

The RMP can be triggered to play out media content streamed over broadcast by Receiver logic or by an explicit request from a Broadcaster Application. These distinctions are further described in this section.

7.1.1 Broadcast or Hybrid Broadband and Broadcast Live Streaming

When tuned to a new service, the RMP determines whether to play out the media stream or whether to wait for the Broadcaster Application to determine whether to play out the media stream. The HTML Entry page Location Description (HELD) specified in A/331 [3] signals which media player, AMP or RMP, is intended to play the media stream; however, Receiver logic can choose to play out the media stream, regardless of what is signaled in HELD. The information in the MPD for ROUTE/DASH or in the MP Table for MMT determines whether the media stream segments are to be played out from broadcast or from a combination of broadband and broadcast.

7.1.2 Broadband Media Streaming

The RMP can play out a service delivered by broadband media streaming if the service signaling indicates a broadband MPD URL, or if the Broadcaster Application requests that the RMP play out the stream. For the purposes of this document, there is no distinction between play-out of live broadband streaming vs. on-demand over broadband. The differentiation on how the MPD is organized for these two use cases is described further in the DASH client specification [41].

7.1.3 Downloaded Media Content

Depending on the request from the Broadcaster Application, the RMP can play out downloaded media content that was delivered via broadband or broadcast. The Broadcaster Application can make such a request using the Set RMP URL WebSocket API as described in Section 9.7.3.

7.1.4 Redistribution

Interactive content can be supported in redistribution scenarios. The Receiver can obtain the ATSC 3.0 service from a redistribution source, present the service using an RMP, acquire the service and application signaling as described in A/336 [5], and acquire the Broadcaster Application package via broadband as described in Section 6.5.

7.2 Utilizing AMP

7.2.1 Broadcast or Hybrid Broadband and Broadcast Live Streaming

Although broadcast or hybrid live media streaming is typically played out by the RMP, it is possible for the AMP to request playback of the content. A flag in the HELD indicates whether the RMP can immediately play out the media content, or whether the service expects the AMP to play out the live media streaming. The Receiver can ignore this signaled expectation, in which case the RMP can immediately play out the live media streaming. There are several possible methods by which an AMP can play out media as described in the following sections.

7.2.2 Broadband Media Streaming

There is no special consideration for playing media streams delivered over broadband other than what is provided in the DASH client specification [41]. Other media types may be supported by a Receiver as reported in the device capabilities provided by the Query Device Info API (see Section 9.12).

7.2.3 Downloaded Media Content

The AMP can play out downloaded media content from either broadcast, using NRT, or broadband, using the Cache Request APIs (see Section 9.3.14). In either case, the resultant media content is placed in the Application Context Cache. The Broadcaster Application may then use the HTML5 `<video>` tag in conjunction with MSE or EME to initiate playout.

7.2.4 AMP Utilizing the Pushed Media WebSocket Interface

This mechanism allows the AMP to play content delivered through the broadcast. The Broadcaster Application opens the binary WebSocket connections specified in Section 8.2.1. Opening these WebSocket connections is an implicit request that the Receiver retrieve the Initialization and Media Segments of the media content and pass them to the Broadcaster Application via these connections. The Broadcaster Application then uses an HTML5 `<video>` tag in conjunction with MSE or EME to pass the media content to the Receiver's decoders for decoding and presentation. In the broadcast case, the media are retrieved from the broadcast via the ROUTE Client and pushed to the AMP via the WebSocket server. In the broadband case, the media are retrieved from a remote HTTP server via the HTTP Client and pushed to the AMP via the WebSocket server.

8. ATSC 3.0 WEBSOCKET INTERFACE

8.1 Introduction

A Broadcaster Application on the Receiver may wish to exchange information with the Receiver platform to:

- Retrieve user settings
- Receive an event from the Receiver to the Broadcaster Application
  - Notification of change in a user setting
  - DASH-style or MPU-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 Remote Procedure Call (RPC) calls. These RPC calls can be used to exchange information between a Broadcaster Application running on the Receiver and the Receiver platform. Figure 8.1 shows the interaction between these components.

In the case of a centralized Receiver architecture, the Receiver Web Server typically can be accessed only from within the Receiver by Broadcaster Applications in the User Agent.

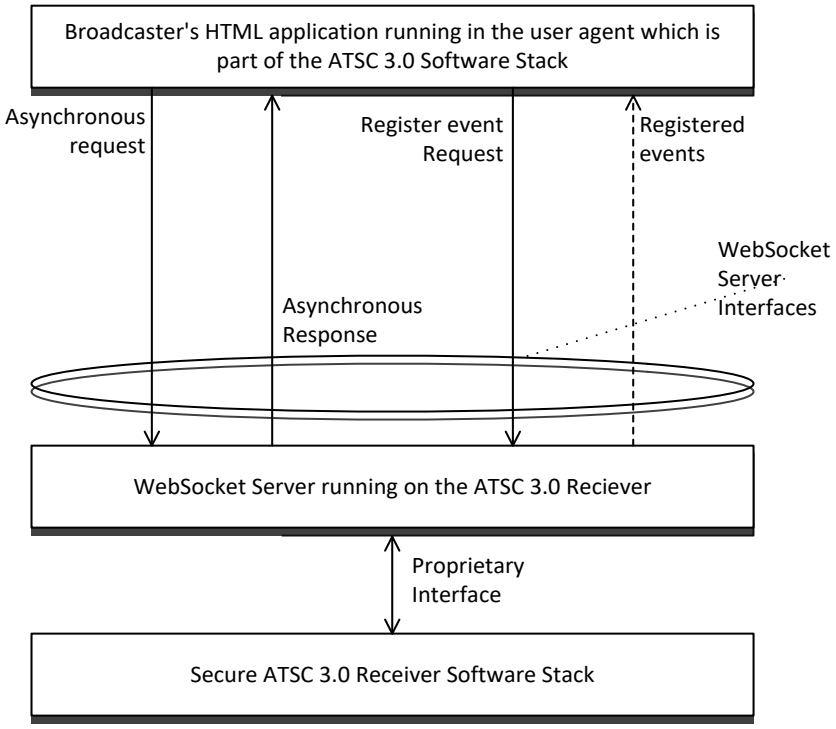


Figure 8.1 Communication with ATSC 3.0 Receiver.

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 Broadcaster Application.

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. To hide these differences from the Broadcaster Application, the Broadcaster Application Entry Page URL is launched with a query term parameter providing information regarding the location of the Receiver WebSocket Server.

When an Entry Page of a Broadcast Application is loaded on the User Agent, the URL is expected to include a query term providing the Base URI of the ATSC 3.0 WebSocket Server Interface supported by Receivers. Similarly, the Receiver is expected to report the current version of the supported WebSocket APIs by providing another query term containing the release date of this standard. An additional optional query term is `callerIdQuery`. This term is expected to be present when the Broadcaster Application was started by another Broadcaster Application.

Using the ABNF syntax as defined in RFC 5234 [12], the query component shall be as defined below:

```
query = ((wsQuery "&" revQuery) / (revQuery "&" wsQuery)) [callerIdQuery]
wsQuery = "wsURL=" ws-url
revQuery = "rev=" yyyymmdd
callerIdQuery = "&callerId=" appId
```

The `ws-url` is the base WebSocket URI and shall be as defined in RFC 6455 [24]. The `yyyymmdd` value shall contain the year (`yyyy`), month (`mm`) and day (`dd`) when the present standard was released. For example, the first release of this standard was 18 December 2018. That value is represented as `'20181218'`. The date used for any given release shall be taken from the corresponding entry in the 'Date' column of the Revision History table at the beginning of this document. The `appId` shall be the `HELD@appId` (see A/331 [3]) of the Broadcaster Application that called the Launch Broadcaster Application API (see Section 9.7.6). Note that this provides only one level of return.

The following shows an example of how such a query string is used to launch the Broadcaster Application. In this example, if the Entry Page URL is:

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

the WebSocket APIs are based on the revision of the standard as released on 20 July 2018 and the Broadcaster Application was launched by the previous Broadcaster Application with `appId="pbs.org/kids/1"`, the Broadcaster Application is launched as follows:

```
http://localhost/xbc.org/x.y.z/home.html?wsURL=wss://localhost2:8000
                                &rev=20180720
                            &callerId=pbs.org/kids/1
```

The `wsURL` and `rev` query parameters are 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 `rev` query term is applicable to launching Broadcaster Applications from both broadcast and broadband.

8.2.1 WebSocket Servers

All Receivers are expected to 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. Receivers that support the A/338 Companion Device standard [37] also provide an additional WebSocket interface that allows communication with the CD Manager within the Receiver (see Section 6.7). Table 8.1 describes the five 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
Companion Device	*WSPath*/atscCD	Optional

If an optional WebSocket URL shown in Table 8.1 is not supported, the Receiver is expected to respond with the HTTP status code "404 Not Found" when the Broadcaster Application attempts to connect to the optional interface. Receipt of this status results in the failure of the WebSocket connection to the particular WebSocket API.

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.

8.2.1.1 Initializing Pushed Media WebSocket Connections

Upon establishment of any of the media WebSocket connections listed in Table 8.1 (atscVid, atscAud, atscCap), it is expected that the first data sent by the Receiver over such a connection is a text message (opcode 0x1, as defined in Section 5.2 of IETF RFC 6455 [24]) with the payload "IS" followed by an Initialization Segment. After the Initialization Segment, the Receiver is expected to send another text message with payload "IS_end" followed by Media Segments. If a new Initialization Segment is received after establishment of the media-delivery WebSocket connection, then the Receiver is expected to send a text message over the same WebSocket connection with the payload "IS" immediately after the last Media Segment associated with the previous Initialization Segment. Then the Receiver is expected to send the new Initialization Segment followed by the text message with payload "IS_end" and then ensuing Media Segments.

8.2.1.2 Media WebSocket Connection Operation

When a Broadcaster Application requests a connection to a media WebSocket, the Receiver is expected to begin sending the appropriate content once the connection is established. The data sent by the Receiver over a media WebSocket connection is expected to be matched to the type of media indicated by the WebSocket Interface Function as provided in Table 8.1. Therefore, video compliant with formatting described in A/331 [3] is expected to be sent by the Receiver over the WebSocket identified by the URL *WSPath*/atscVid. Similarly, audio compliant with formatting described in A/331 [3] and captions compliant with formatting described in A/331 [3] is expected to be sent over *WSPath*/atscAud and *WSPath*/atscCap WebSocket connections, respectively. For media WebSocket connections that are currently open, content may not be sent at all times, for example, if no captions are present at a given time.

8.3 Data Binding

Once the connection is established to the Receiver WebSocket command and control Server, messages can be sent and received. However, since the WebSocket interface is just a plain bidirectional interface with no structure other than message framing, a message format needs to be defined.

The WebSocket interface for command and control shall be the JSON-RPC 2.0 Specification in Annex B, except that the WebSocket interface need not include the features described in Annex B, Section 6 (batch mode operation). 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 [22].

This section defines the basic formatting of messages, and the following section defines the specific messages that are supported. This document describes message semantics while separate schema files provide the normative syntax. The message syntax specified in the separate schema files shall be as defined in the JSON Schema specification [19]. Additional supporting information for JSON schemas may be found at [47].

Note that, once opened, the connection to the Receiver WebSocket command and control Server is expected to remain open for the lifetime of the Broadcaster Application. Closing the command-and-control WebSocket interface and reestablishing the connection is undefined and may result in the loss of state, e.g., key timeouts may not be received from the Receiver, Receiver resources may be re-allocated, etc.

In the event of any discrepancy between the JSON schema definitions implied by the tables that appear in this document and those that appear in the JSON schema definition files, those in the JSON schema definition files are authoritative and take precedence.

The terms in the "Data Type" column of the semantic API tables are a shorthand for datatypes defined in the JSON Schema [19] and shall be as defined there. In order to provide flexibility for future changes in the schema, decoders of JSON documents defined in the present document should ignore any datatypes they do not recognize, instead of treating them as errors.

JSON schemas and messages shall be encoded stringified as UTF-8. The Receiver is expected to parse the JSON message 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 are used for delivery of binary data from the Receiver to the Broadcaster Application.

The notation used to describe the flow of data in the examples in this specification is 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 either the Receiver or 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]
}
```

Note that the lack of an `'id'` property indicates that no response is expected in the case of a notification.

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 General JSON Property Considerations

The Cancel Request Command described in the next subsection and the APIs defined in Section 9 use a JSON schema [19] to describe the acceptable syntax for the JSON objects used in request and response methods. The JSON schemas are maintained in separate files, one for each schema. Those schemas and the text describing the elements and properties specified within the schemas are normative. Examples are informative and are intended to clarify the meaning and usage of the API. Any differences in examples and normative schema are unintentional.

The schema for an API may specify that some properties are required. This means that the property shall be present as a key in the JSON structure compliant with the schema. For simple types, unless explicitly stated otherwise, a `"null"` value may be used for the required property if the Receiver or Broadcaster Application does not have sufficient information to provide a value

for the required key. Semantically, a "null" value indicates that the implementation is aware that the key is required but cannot supply meaningful data. For required object and array structures, if no information is available, an empty structure shall be provided unless explicitly stated otherwise or properties within those structures are required by the structure's schema.

8.3.2 Cancel Request Command

The Cancel Request Command may be used to terminate a selected number or all outstanding JSON-RPC requests. The request message supplies a list of IDs corresponding to the request IDs. The response message lists the requests that have been terminated. If no requests matching the requested IDs can be found, an error response is provided. If a request is successfully canceled, the Receiver is expected to issue a response to that request with an error code indicating that the request was canceled. A cancel request is not expected to be used to cancel a previous cancel request.

The Cancel Request Command semantics are as defined in Table 8.2 and the syntax shall be as defined in the schema file `cancel-request.json`. Additional semantic definitions of parameters follow the table.

Table 8.2 Cancel Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"cancel"
params	0..1		
requestIDs	1		An array of one or more request IDs to be canceled.
items	1..N	integer	

`requestIDs` – This list in the optional `params` object shall contain one or more IDs of outstanding requests. If the `params` object and its child `requestIDs` list are not supplied, then all outstanding requests are expected to be canceled.

The Cancel Response semantics are defined in Table 8.3 and the syntax shall be as defined in the schema file `cancel-response.json`. Additional semantic definitions, if any, follow the table.

Table 8.3 Cancel Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
cancelList	1		An array of requests and their disposition in response to the cancel request
items	1..N		
requestID	1	integer	The request ID of a request that was requested to be canceled
disposition	1	enum	One of "CANCELED", "UNKNOWN", or "FAILED"
description	0..1	string	Information regarding the cancel operation
error	oneOf X		See Section 8.3.3

The Receiver is expected to accumulate the canceled request IDs so that a single response can be supplied. Note that each request receives a separate error response indicating that the request was canceled.

`cancelList` – This required property shall provide a list of objects in response to the cancel request. The list is expected to have the same number of objects as the number of request IDs in the cancel command or the number of outstanding requests if no request IDs were supplied requesting that all outstanding requests be canceled.

`requestID` – This required property shall contain one of the request IDs provided in the cancel command or the `requestID` of a canceled request if all requests were to be canceled.

`disposition` – This required property shall contain one of the following values:

- `CANCELED` – Indicates that the request corresponding to the object's `requestID` was successfully canceled.
- `UNKNOWN` – Indicates that the request corresponding to the object's `requestID` was not found. The request was not canceled because the Receiver could not identify an outstanding request corresponding to the given `requestID`, or because the request with that ID value was completed before the cancel request was processed.
- `FAILED` – Indicates that the request corresponding to the object's `requestID` could not be canceled. In this case, the Receiver found the specified request but could not successfully cancel the request.

`description` – This optional property shall contain a description of the cancel operation.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query request with ID '12' and, due to a user action removing the need for the query, the Broadcaster Application cancels the request with the following command:

```
--> {
    "jsonrpc": "2.0",
    "method": "cancel",
    "params": {
        "requestIDs": [12]
    },
    "id": 913
}
```

The Receiver might respond to the cancel request as follows:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "cancelList": [
          { "requestID": 12,
            "disposition": "CANCELED",
            "description": "Request canceled successfully" }
        ]
    },
    "id": 913
}
```

The Receiver might also issue the following response to query request with ID 12:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -20, "message": "Request Canceled" },
    "id": 12
}
```

As a further example, the Broadcaster Application may wish to cancel all outstanding requests of a function because the user has switched modes of operation. In this case, the Broadcaster Application attempts to cancel the corresponding outstanding requests as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "cancel",
    "params": {
        "requestIDs": [42, 216, 922]
    },
    "id": 226
}
```

The Receiver might respond to the cancel request as follows:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "cancelList": [
          { "requestID": 42,
            "disposition": "UNKNOWN",
            "description": "Request is not currently pending" },
          { "requestID": 216,
            "disposition": "CANCELED",
            "description": "Request canceled successfully" },
          { "requestID": 922,
            "disposition": "CANCELED",
            "description": "Request canceled successfully" }
        ]
    },
    "id": 226
}
```

The Receiver issues error responses to query requests with IDs 216 and 922. Note that the `UNKNOWN` disposition for `requestID` 42 may not be an error since the request could have completed just as the cancel command was issued.

As a further example, the Broadcaster Application wishes to shut down all operations and terminate all outstanding requests. In this case, the Broadcaster Application attempts to cancel all outstanding requests as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "cancel",
    "id": 226
}
```

The Receiver might respond to the cancel request as follows:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "cancelList": [
          { "requestID": 324,
            "disposition": "CANCELED",
            "description": "Request canceled successfully" },
          { "requestID": 167,
            "disposition": "CANCELED",
            "description": "Request canceled successfully" }
        ]
    },
    "id": 226
}
```

The Receiver also issues error responses to query requests with IDs 324 and 167. The Broadcaster Application can assume that all outstanding requests have been canceled.

8.3.3 Error Handling

JSON-RPC 2.0 defines a set of reserved error codes. See the table in Annex B Section 5.1. The semantics of the error structure are also defined in in Annex B Section 5.1 with the structure show in Table 8.4.

Table 8.4 Error Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request with structure dependent on the API otherwise the error structure is returned
`error`	oneOf X		See Annex B Section 5.1
`code`	1	integer	The error code indicating what problem occurred.
`message`	1	string	A concise message describing the error
`data`	0..1		An optional primitive or object that contains additional information about the error

`error` – Provided instead of the `"result"` structure if an error occurs. See Annex B Section 5.1 Error object for a normative description of an error return. General JSON RPC error codes are defined in Annex B. Specific error codes provided by individual API responses are detailed in the response definition for each API. A summary of the ATSC-defined error codes from the Receiver are listed in Table 8.5. The asterisk column indicates that the associated error code may occur in an error response to any API request.

Table 8.5 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 RMP URL API, the MPD referenced in the URL provided cannot be accessed.
-12		The content cannot be played	In response to the Set RMP URL API, the requested content cannot be played.
-13		The requested MPD Anchor cannot be reached	In response to the Set RMP URL API, the MPD Anchor indicated cannot be reached (e.g., beyond the end of the file).
-14		Unsupported or Unknown Content Protection System	The specified content protection system is not supported by or unknown to the Receiver.
-15		Illegal URL Format	The URL format specified in `sourceURL` or `targetURL` of the request is illegal.
-16		Illegal URL Format	The URL format specified in one or more URLs in the requested list is illegal.
-17		Malformed DASH Period	The format of the MPEG DASH fragment specified in the Period is illegal.
-18		MPD not found	The referenced MPD file cannot be found.
-19		The synchronization specified by `rmpSyncTime` cannot be achieved	In response to the Set RMP URL API with `rmpSyncTime`, the synchronization indicated by `rmpSyncTime` cannot be achieved.
-20	*	Request Canceled	The Broadcaster Application issued the cancel request command (Section 8.3.1) with the ID corresponding to this request.

-21		Changing RMP playback from the current source is not supported	In response to the Set RMP URL API, the Receiver does not support changing playback from the current source to an alternate source (e.g., broadband or locally cached content).
-22		dialogEnhancement failed	The request to set or receive Dialog Enhancement data failed.
-23		appId not found	The requested Broadcaster Application appId was not in the HELD.
-24		Not subscribed	The Broadcaster Application was not subscribed to any of the requested notifications.
-25		Referenced BA not available	The `appId` was found in the HELD but is not available or is broadcast-only and has not yet been acquired
-26		Cannot access referenced broadband BA	The `appId` was found in the HELD and is broadband-only but is not available due to lack of network connectivity
-27		No Receiver capability	The Receiver does not support the required capabilities
-28		Unknown filter codes	One or more of the supplied filter codes is currently undefined
-29		Deprecated (Unknown DRM system)	Deprecated, see Error Code -14.
-30		Too late	The request was not received in time to replace the default content
-31		Unknown XLink	The provided `elementType` and/or `elementId` were not found
-32		Invalid element	The provided MPD element is not valid
-33		Asset cannot be selected	Asset cannot be selected. The media asset identified in the MMT Media Asset Selection API cannot be found or selected.
-34		MMT Asset not found	The MMT Asset described by `assetLink`, `assetType` and/or `assetId` not found.
-35		Replacement MMT Asset not valid	The MMT Asset content provided as replacement content is not valid.
-100		EME TypeError	See EME Section 6.5 [31]
-101		EME NotSupportedError	See EME Section 6.5 [31]
-102		EME InvalidStateError	See EME Section 6.5 [31]
-103		EME QuotaExceededError	See EME Section 6.5 [31]
-200..-299		Reserved	Reserved for A/338 Error Codes [37]

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 8. 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.

Schemas and examples for the APIs described in this section and the cancel API described previously in Section 8.3.2 can be found at
https://atsc-schemas.org/atsc3.0/a344/20250226/

Note that the schemas found in the schema repository are authoritative. Syntax and examples provided in this document are considered informative.

9.1 API Revision Control

To avoid issues with backwards compatibility, the present revision of the standard does not make changes to the semantics of keys within an API but, instead, deprecates the entire API and creates a new API to support the changed functionality. Further, no keys within an API have been deleted and only new keys have been added. For some APIs, values for existing keys may have been added, datatypes for key values may have changed where the types are compatible, and additional enumeration values may have been defined. The Broadcaster Application and Receiver are expected to gracefully ignore unknown keys and unknown values for existing keys, including unknown enumeration values.

Table 9.1 lists the APIs and groups of APIs and indicates whether they are applicable to AMP operation, RMP operation, or both. The Changes column of Table 9.1 provides a notice that the associated API has changed in a revision of this standard. The following list provides the possible entries for the Changes column and their meaning.

No Entry – The API has no modifications in this revision.

Added – A new API has been defined in this revision.

Deprecated – The associated API has been deprecated. An alternate API is referenced in the Reference column. If an API has been deprecated, the Receiver may continue to support the API, but the Broadcaster Application cannot rely on its availability in a future revision and should migrate to the referenced API.

Description Changed – The text supporting the description of the API has changed. No substantive changes have been made to the semantics or syntax of the API.

Extended – The API has been extended by adding new keys or values. Previous keys and values remain as previously defined.

Table 9.1 API Applicability

WebSocket APIs	Reference	Applicability	Changes
Cancel Request Command	Section 8.3.1	Always	Added in A/344:2020
Receiver Query APIs	Section 9.2	Always	Extended in A/344:2019
Query Content Advisory Rating API	Section 9.2.1	Always	Deprecated `"rating"` parameter in A/344:2020
Query MPD URL API	Use Section 9.2.10	RMP	Deprecated in A/344:2020
Query Alerting URL API	Section 9.2.8	Always	Deprecated in A/344:2019
Query Alerting Signaling API	Section 9.2.8	Always	Added in A/344:2019
Query Service Guide URLs API	Section 9.2.9	Always	Added in A/344:2020
Query Signaling Data API	Section 9.2.10	Always	Added in A/344:2020
Query Dialog Enhancement Preferences API	Section 9.2.11	Always	Added in A/344:2020
Content Advisory Rating Change Notification API	Was Section 9.3.2	Always	Deprecated in A/344:2020
MPD Change Notification API	Use Section 9.3.11	RMP	Deprecated in A/344:2020

Alerting Change Notification API	Section 9.3.8	Always	Deprecated in A/344:2019†
Alerting Change Notification API	Section 9.3.8	Always	Added in A/344:2019†
Service Guide Change Notification API	Section 9.3.10	Always	Added in A/344:2020
Signaling Data Change Notification API	Section 9.3.11	Always	Added in A/344:2020
Dialog Enhancement Preference Change Notification API	Section 9.3.12	Always	Added in A/344:2020
Dialog Enhancement Limit Change Notification API	Section 9.3.13	Always	Added in A/344:2020
Cache Request APIs	Section 9.3.14	Always	
Query Cache Usage API	Section 9.5	Always	
Event Stream APIs	Section 9.6	RMP	
Acquire Service API	Section 9.7.1	Always	
Video Scaling and Positioning API	Section 9.7.2	RMP	Extended in A/344:2019
XLink Resolution API	Use Sections 9.3.1 and 9.15	RMP	Deprecated in A/344:2019
Subscribe MPD Changes API	Use Section 9.3.1.1	AMP	Deprecated in A/344:2019
Unsubscribe MPD Changes API	Use Section 9.3.1.2	AMP	Deprecated in A/344:2019
Set RMP URL API	Section 9.7.3	RMP	Extended in A/344:2019 and A/344:2020
Audio Volume API	Section 9.7.4	RMP	
Dialog Enhancement API	Section 9.7.5	RMP	Added in A/344:2020
Launch Broadcaster Application API	Section 9.7.6	Always	Added in A/344:2020
Integrated Subscribe API	Section 9.3.1.2	Always	Added in A/344:2019, Extended in A/344:2020
Integrated Unsubscribe API	Section 9.3.1.2	Always	Added in A/344:2019, Extended in A/344:2020
Subscribe Alerting Changes	Use Section 9.3.1.1	Always	Deprecated in A/344:2019
Unsubscribe Alerting Changes	Use Section 9.3.1.2	Always	Deprecated in A/344:2019
Subscribe Content Changes	Use Section 9.3.1.1	Always	Deprecated in A/344:2019
Unsubscribe Content Changes	Use Section 9.3.1.2	Always	Deprecated in A/344:2019
Subscribe RMP Media Time Change Notification	Use Section 9.3.1.1	RMP	Deprecated in A/344:2019
Unsubscribe RMP Media Time Change Notification	Use Section 9.3.1.2	RMP	Deprecated in A/344:2019
Subscribe RMP Playback State Change Notification	Use Section 9.3.1.1	RMP	Deprecated in A/344:2019
Unsubscribe RMP Playback State Change Notification	Use Section 9.3.1.2	RMP	Deprecated in A/344:2019
Subscribe RMP Playback Rate Change Notification	Use Section 9.3.1.1	RMP	Deprecated in A/344:2019
Unsubscribe RMP Playback Rate Change Notification	Use Section 9.3.1.2	RMP	Deprecated in A/344:2019
Media Track Selection API	Section 9.7.7	RMP	
Mark Unused API	Section 9.8	Always	Description changed in A/344:2019
Content Recovery APIs	Section 9.9	Always	
Get Filter Code API	Was Section 9.11.1	Always	Deprecated in A/344:2020
Set Filter Code Instances API	Section 9.10.1	Always	Moved from Section 9.11.2, renamed and clarified in A/344:2020
Clear Filter Code Instances API	Section 9.10.2	Always	Added in A/344:2020

Keys APIs	Section 9.11	Always	
Query Device Info API	Section 9.12	Always	Extended in A/344:2019 and A/344:2020
RMP Content Synchronization APIs	Section 9.13	RMP	
Query RMP Wall Clock API	Section 9.13.2	RMP	Deprecated in A/344:2022-03
Query RMP UTC Time API	Section 9.13.2	RMP	Added in A/344:2023-02 Deprecated
Digital Rights Management (DRM) APIs	Section 9.13.8	Always	Added in A/344:2019
XLink Management APIs	Section 9.15	RMP	Added in A/344:2019
Prep Service Change API	Section 9.16	AMP, RMP	Added in A/344:2023-02

† Note that the Alerting Change Notification API was substantially changed in A/344:2019 resulting in the previous API definition being deprecated with a new API defined with the same name. Essentially, the API was modified to pass the alerting XML fragment(s) instead of a URL reference to the alerting XML fragment(s).

9.2 Receiver Query APIs

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

If these settings are not available from the Receiver, the Broadcaster Application may use default values based on its own business policy and logic. A Broadcaster Application may choose to provide its own settings user interface and store the collected settings as cookies on the Receiver.

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

9.2.1 Query Content Advisory Rating API

The Broadcaster Application may wish to discover the current content advisory rating as signaled in the content currently being rendered by the Receiver and whether that content is being blocked or not. It is assumed that when content is blocked, the Broadcaster Application is not blocked and continues to execute, although access to certain APIs may be restricted.

The Query Content Advisory Rating Request semantics are defined in Table 9.3 and the syntax shall be as defined in the schema file `org.atsc.query.ratingLevel-request.json`.

Table 9.2 Query Content Advisory Rating Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.ratingLevel"

The Query Content Advisory Rating Response semantics are defined in Table 9.3 and the syntax shall be as defined in the schema file `org.atsc.query.ratingLevel-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.3 Query Content Advisory Rating Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`blocked`	1	boolean	Indicates, if "true", that the current content is blocked from being displayed
`contentRating`	1	string	The rating as found on the currently playing content
`error`	oneOf X		See Section 8.3.3

`blocked` – This required Boolean value shall indicate whether the Receiver is currently blocking the content due to the content advisory rating of the service being higher than the content advisory rating preference.

`contentRating` – This required string value shall provide the content advisory rating of the content currently being rendered by the Receiver Media Player in string format, as defined in A/331 [3], Section 7.3. Note that the content rating string shall contain all of the rating values including from multiple rating regions if appropriate. To specify content advisory information data for multiple rating regions, additional three-part strings (one for each region) shall be concatenated to create one string consisting of multiple concatenated three-part strings. In this case, the third part of each content advisory information string except the last shall be followed by a comma (","). Thus, the last character of the entire content advisory ratings string is a right curly brace ("}"). If there is no Content Rating, this property shall be present and set to an empty string, i.e., `"contentRating":""`.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, consider the case that the content advisory rating setting is TV-PG-D-L for the US Rating Region 1, and the current content advisory rating is TV-G. The Broadcaster Application can make a request:

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

The Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "blocked": false,
        "contentRating": "1,'TV-G', {0 'TV-G'}"
    },
    "id": 37
}
```

9.2.2 Query Closed Captions Enabled/Disabled API

The Broadcaster Application may wish to know whether the user has turned on closed captions. The Broadcaster Application requests the closed caption setting from the Receiver via Receiver WebSocket Server interface.

The Query Closed Captions Enabled/Disabled Request semantics are defined in Table 9.4 and the syntax shall be as defined in the schema file `org.atsc.query.cc-request.json`.

Table 9.4 Query Closed Captions Enabled/Disabled Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.cc"

The Query Closed Captions Enabled/Disabled Response semantics are defined in Table 9.5 and the syntax shall be as defined in the schema file `org.atsc.query.cc-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.5 Query Closed Captions Enabled/Disabled Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`ccEnabled`	1	boolean	Indicates whether closed captioning is enabled or not.
`error`	oneOf X		See Section 8.3.3

`ccEnabled` – This required Boolean shall indicate true if closed captions are currently being displayed and false otherwise.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

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
}
```

9.2.3 Query Service ID API

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

The Query Service ID Request semantics are defined in Table 9.6 and the syntax shall be as defined in the schema file `org.atsc.query.service-request.json`.

Table 9.6 Query Service ID Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.service"

The Query Service ID Response semantics are defined in Table 9.7 and the syntax shall be as defined in the schema file `org.atsc.query.service-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.7 Query Service ID Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`service`	1	string (uri)	Specifies the `globalServiceID` of the currently selected service
`error`	oneOf X		See Section 8.3.3

`service` – This required property shall indicate the `globalServiceID` associated with the currently selected service as given in the SLT in **`SLT.Service`**`@globalServiceID`. See A/331 [3] Section 6.3. Note that `globalServiceID` is required for "normal" service types that can be selected by the user (e.g., `@serviceCategory` 1 and 2) and must be present in the SLT. For service types

that do not require `globalServiceID` (e.g., DRM, ESG, NRT), the Receiver shall return a `"null"` value consistent with the behavior described in Section 8.3.1.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the `globalServiceID` for the currently selected services is `"https://doi.org/10.5239/8A23-2B0B"`, and the Broadcaster Application issues a request to the Receiver:

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

The Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"service": "https://doi.org/10.5239/8A23-2B0B"},
    "id": 55
}
```

9.2.4 Query Language Preferences API

The 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 Broadcaster Application may use the Query Language Preferences API to determine these settings.

The Query Language Preferences Request semantics are defined in Table 9.8 and the syntax shall be as defined in the schema file `org.atsc.query.languages-request.json`.

Table 9.8 Query Language Preferences Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.languages"

The Query Language Preferences Response semantics are defined in Table 9.9 and the syntax shall be as defined in the schema file `org.atsc.query.languages-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.9 Query Language Preferences Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`preferredUILang`	1	string	Provides the preferred language of the Receiver User Interfaces

	preferredAudioLang	0..1	string	Provides the preferred language for the audio output
	preferredCaptionSubtitleLang	0..1	string	Provides the preferred language of the closed captions or subtitles
error		oneOf X		See Section 8.3.3

`preferredUILang`, `preferredAudioLang`, `preferredCaptionSubtitleLang` – Each of these strings indicates the currently set language preference of the respective item, coded according to BCP 47 [21]. At minimum, Receivers shall provide the current UI language as the `preferredUILang`.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster 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
}
```

9.2.5 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 Broadcaster Application may use the Query Caption Display Preferences API to determine these settings.

The Query Caption Display Preferences Request semantics are defined in Table 9.10 and the syntax shall be as defined in the schema file `org.atsc.query.captionDisplay-request.json`.

Table 9.10 Query Caption Display Preferences Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.query.captionDisplay"

The Query Caption Display Preferences Response semantics are defined in Table 9.11 and the syntax shall be as defined in the schema file `org.atsc.query.captionDisplay-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.11 Query Caption Display Preferences Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
cta708	0..1	object	See semantic definition in Section 9.2.5.1
imsc1	0..1	object	See semantic definition in Section 9.2.5.2
error	oneOf X		See Section 8.3.3

One or more of the caption scheme objects defining the closed caption display preferences may be included in the notification message. These objects are defined in the following subsections.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

9.2.5.1 CTA 708 Semantics

The "cta708" object, if present, is expected to provide one or more of the closed caption display preferences as described in Table 9.12.

Note: The use of "cta708" is for backwards syntactical compatibility. Although there is some alignment with CTA 708 syntax, the syntax is not necessarily conformant to CTA 708.

Table 9.12 Caption Display Preferences CTA 708 Object Semantics

Property Name	Use	Data Type	Short Description
cta708	0..1	object	The object containing caption display preference properties
characterColor	0..1	string	A string representing the color of the characters
characterOpacity	0..1	number	An integer or fixed-point number in the range 0 to 1 inclusive representing the opacity of the characters
characterSize	0..1	integer	A percentage multiplier of the default font size
fontStyle	0..1	examples	A string value indicating the preferred font style
backgroundColor	0..1	string	A string defining the color of the character background
backgroundOpacity	0..1	number	An integer or fixed-point number in the range 0 to 1 inclusive representing the opacity of the character background
characterEdge	0..1	examples	A string value indicating the preferred character edge
characterEdgeColor	0..1	string	A string specifying the color of the character edges, if applicable
windowColor	0..1	string	A string representing the color of the caption window background

	windowOpacity	0..1	number	An integer or fixed-point number in the range 0 to 1 inclusive that represents the opacity of the caption window

characterColor – This parameter corresponds to CEB35 [10], Section 7.3 "Pen Styles, character foreground color", and shall be 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 [9] using the "#" 24-bit sRGB notation. For example, red is represented as "#FF0000".

characterOpacity – This parameter corresponds to CEB35 [10], Section 7.4 "Background Color and Opacity, character foreground opacity", and shall be 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 means 33% opaque, while a value of 0 means completely transparent.

characterSize – This parameter corresponds to CEB35 [10], Section 7.1 "Pen Size", and shall be a percentage multiplier of the default font size where 100 has no change, 50 is ½ size and 200 is double the size. The syntax and semantics shall be consistent with IMSC1 as defined in A/343 [7].

fontStyle – This string corresponds to CEB35 [10], Section 7.2 "Font Styles", and shall indicate the style of the preferred caption font. The eight possible choices shall correspond to the CEB35 [10] numbered font styles in Section 7.2:

"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 corresponds to CEB35 [10], Section 7.4 "Background Color and Opacity, Background Color", and shall represent the color of the character background, given in the same CSS-compatible format as characterColor.

backgroundOpacity – This parameter corresponds to CEB35 [10], Section 7.4 "Background Color and Opacity, Foreground Opacity", and shall be an integer or fixed-point number in the range 0 to 1 inclusive 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 corresponds to CEB35 [10], Section 7.5 "Character Edges, type attributes", and shall indicate the preferred display format for character edges. The preferred color of the edges (or outlines) of the characters shall be as given in characterEdgeColor. Edge opacities shall have the same attribute as the character foreground opacities. The choices are as specified in CEB35 [10], Section 7.5: "None", "Raised", "Depressed", "Uniform", "LeftDropShadow", and "RightDropShadow".

characterEdgeColor – This parameter corresponds to CEB35 [10], Section 7.5 "Character Edges, edge color", and shall represent the color of the character edges, if applicable, given in the same format as characterColor.

`windowColor` – This parameter corresponds to CEB35 [10], Section 8.1 "Window, color", and represents the color of the caption window background, given in the same format as `characterColor`.

`windowOpacity` – This parameter corresponds to CEB35 [10], Section 8.1 "Window, opacity", and shall be an integer or fixed-point number in the range 0 to 1 inclusive 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.

9.2.5.2 IMSC1 Extensions Semantics

The key/value pairs defined in this section provide a means to represent any IMSC1 (as defined in A/343 [7]) attribute preference within the `"imsc1"` object, if present.

The key/value pairs for IMSC1 preferences shall take the form:

```
"<imsc1_key>": "<imcs1_value>"
```

The syntax of the `<imsc1_key>` shall be as specified below using the Augmented Backus-Naur Form (ABNF) grammar defined in RFC 5234 [12]:

```
<imsc1_key> = ("region_" / "content_") + imsc1_attribute
```

`imsc1_attribute` – This part shall be the name of any IMSC1-defined attribute.

`<imsc1_key>` – The value data type range of values and their encodings shall be those supported by IMSC1 for the attribute named in `imsc1_attribute`.

The second part of `<imsc1_key>` shall indicate `"region_"` when the `<imsc1_value>` applies to regions, and `"content_"` when it applies to content (text).

Valid examples of `<imsc1_key>` are `"region_backgroundColor"` and `"content_backgroundColor"` indicating the background color of either a region or of content (text), respectively.

9.2.5.3 Caption Display Preferences Query Example

The Broadcaster Application may make the following query to obtain the caption display preferences:

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

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "cta708": {
            "characterColor": "#F00000",
            "characterOpacity": 0.5,
            "characterSize": 80,
            "fontStyle": "MonospacedNoSerifs",
            "backgroundColor": "#808080",
            "backgroundOpacity": 0,
            "characterEdge": "None",
            "characterEdgeColor": "#000000",
            "windowColor": "#000000",
            "windowOpacity": 0
        },
        "imsc1": {
            "region_textAlign": "center",
            "content_fontWeight": "bold"
        }
    },
    "id": 932
}
```

9.2.6 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 Broadcaster Application may use the Query Audio Accessibility Preferences API to determine these settings.

The Query Audio Accessibility Preferences Request semantics are defined in Table 9.13 and the syntax shall be as defined in the schema file `org.atsc.query.audioAccessibilityPref-request.json`.

Table 9.13 Query Audio Accessibility Preferences Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.audioAccessibilityPref"

The Query Audio Accessibility Preferences Response semantics are defined in Table 9.14 and the syntax shall be as defined in the schema file `org.atsc.query.audioAccessibilityPref-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.14 Query Audio Accessibility Preferences Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`videoDescriptionService`	0..1		

		enabled	0..1	boolean	Indicates whether or not a video description service is enabled
		language	0..1	string	The preferred language of the video description service
	audioEIService		0..1		
		enabled	0..1	boolean	Indicates whether or not emergency information audio is enabled
		language	0..1	string	The preferred language of the emergency information audio
error			oneOf X		See Section 8.3.3

`result` – If neither the video description service nor audio emergency information rendering is enabled, the `result` structure shall contain no elements. In JSON, this is represented as `"result": {}`.

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

`videoDescriptionService.language` – A string that shall indicate the preferred language of VDS rendering, coded according to BCP 47 [21].

`audioEIService.language` – A string that shall indicate the preferred language of audio/aural representation of emergency information rendering, coded according to BCP 47 [21].

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

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

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

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

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

9.2.7 Query Receiver Web Server URI API

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

This API is useful in the situation where the Broadcaster Application was started from a broadband server in which case it would be unaware of the Application Context Cache URI. A Broadcaster Application executing from Application Context Cache can determine its server location through standard W3C DOM parameters.

The Query Receiver Web Server URI Request semantics are defined in Table 9.15 and the syntax shall be as defined in the schema file `org.atsc.query.baseURI-request.json`.

Table 9.15 Query Receiver Web Server URI Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.baseURI"

The Query Receiver Web Server URI Response semantics are defined in Table 9.16 and the syntax shall be as defined in the schema file `org.atsc.query.baseURI-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.16 Query Receiver Web Server URI Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`baseURI`	1	string (uri)	Provides the URI of where the active Application Context Cache may be accessed
`error`	oneOf X		See Section 8.3.3

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

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

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

The Receiver responds with the URI of the Receiver Web Server for the Application Context Cache defined for the current Application Context Identifier:

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

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

9.2.8 Query Alerting Signaling API

The Broadcaster Application may wish to access the various alerting metadata structures signaled in the current broadcast. The Query Alerting Signaling API returns a list of the specific alerting metadata the Broadcaster Application has requested.

The Query Alerting Signaling Request semantics are defined in Table 9.17 and the syntax shall be as defined in the schema file `org.atsc.query.alerting-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.17 Query Alerting Signaling Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.alerting"
`alertingTypes`	1	array of enum	A list of requested alerting types. An empty list means all.

`alertingTypes` – An array of one or both of the alerting types as follows:

`AEAT` – Requests the most recent AEAT XML fragment, if any.

`OSN` – Requests the most recent OSN XML fragment, if any.

An empty list is equivalent to supplying all values.

The Query Alerting Signaling Response semantics are defined in Table 9.18 and the syntax shall be as defined in the schema file `org.atsc.query.alerting-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.18 Query Alerting Signaling Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
alertList	1	array	A list of alerting fragments based on the request. The list may be empty if no alerting signaling matching the requested types is active.
items	0..N		
alertingType	1	enum	"AEAT" or "OSN"
alertingFragment	1	string (xml)	The XML fragment of the associated alerting type
receiveTime	0..1	string (date-time)	If alertingType = "OSN", the date and time when the fragment was received
filteredEventList	0..1	array	Provides an array of AEA IDs that have been filtered out by the Receiver
	items 1..N	string	
error	oneOf X		See Section 8.3.3

`alertList` – An array of alerting signaling fragments as specified in the request. The array may be empty if none of the requested alerting signaling is active.

`alertingType` – This required parameter shall contain one of the alerting types, `"AEAT"` or `"OSN"`. The corresponding `alertingFragment` shall contain the data corresponding to the type of alerting metadata fragment indicated.

`alertingFragment` – This required string shall contain the alerting XML fragment for the associated `alertingType`. The AEAT XML and OSN XML fragments are extracted from their respective LLS tables that are described in A/331 [3].

`receiveTime` – The date and time when the alerting fragment was received. This value shall be provided when the object is `"OSN"`. (Note: The `OnscreenMessageNotification` element includes a `KeepScreenClear@notificationDuration` attribute which is the duration of the `KeepScreenClear` message starting from the time the OSN was received. Thus, the time the OSN was received is necessary for the Broadcaster Application to fully utilize the OSN information.) The date-time JSON data type shall be formatted as defined in the JSON Schema specification [19].

`filteredEventList` – Provides a list of AEA events that have been filtered out by the Receiver. The Receiver may elect to filter out an event for a variety of reasons based on user preferences, location or some other criteria. If an `AEA` event has been filtered out, the corresponding `AEAT.AEA@aeaId` shall appear in the `filteredEventList` property. If an `AEA` event has not been filtered out, the corresponding `AEAT.AEA@aeaId` shall not appear in the list. An empty or absent `filteredEventList` indicates that no events have been filtered out by the Receiver. This property is only applicable when the `alertingType` is `"AEAT"`. AEA events that have been "filtered out" are those that have been handled or processed by the Receiver and need not be processed by the Broadcaster Application.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.alerting",
    "params": {
        "alertingTypes": ["AEAT", "OSN"]
    },
    "id": 913
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "alertList": [
          { "alertingType": "AEAT",
            "alertingFragment": "<AEAT>…</AEAT>" },
          { "alertingType": "OSN",
            "alertingFragment": "<OSN>…</OSN>",
            "receiveTime": "2017-01-01T23:54:59.590Z" }
        ]
    },
    "id": 913
}
```

9.2.9 Query Service Guide URLs API

The Broadcaster Application may wish to access the various service guide data structures provided in the current broadcast. The Query Service Guide URLs API returns a list of URLs the Broadcaster Application can use to retrieve (for example, by XHR) the specific service guide data structures provided in the broadcast.

The Query Service Guide URLs Request semantics are defined in Table 9.19 and the syntax shall be as defined in the schema file `org.atsc.query.serviceGuideUrls-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.19 Query Service Guide URLs Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.serviceGuideUrls"
`service`	0..1	string (uri)	Requests the service guide information pertinent to the specified service

`service` – The optional `service` field as defined in the Query Service ID API in Section 9.2.3. When omitted, all service guide fragments are returned for all services. When present, only those fragments related to the provided service are returned.

The Query Service Guide URLs Response semantics are defined in Table 9.20 and the syntax shall be as defined in the schema file `org.atsc.query.serviceGuideUrls-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.20 Query Service Guide URLs Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
urlList	1	array	Lists the set of service guide URLs based on the requested service, if specified, or all service guide URLs if not
items	0..N		
sgType	1	enum	"Service", "Schedule" or "Content"
sgUrl	1	string (url)	The URL of the XML fragment of the associated service guide type
service	1	string (uri)	The URI of the service related to the service guide type
content	0..1	string (uri)	When the sgType = "Content", this parameter is used to provide the unique ID of the content, if available
error	oneOf X		See Section 8.3.3

`urlList` – Provides an array, perhaps empty, of the service guide URLs found in response to the Service Guide URLs request.

`sgType` – One of the service guide XML fragment types. The corresponding `sgUrl` can be used to access the XML fragment corresponding to the type of service guide fragment indicated. Note that there may be multiple fragments within the list of the same `sgType`. The `sgType` may be used to quickly access fragments of interest.

`sgUrl` – A fully-qualified URL that can be used by the Broadcaster Application, for example in an XHR request, to retrieve the current broadcast service guide XML fragment for the associated `sgType`. The service guide is delivered in XML fragments whose syntax shall be as defined in A/332 [4]. This returns exactly one service guide fragment.

`service` – The required `service` field as defined in the Query Service ID API in Section 9.2.3. Note: This is more commonly known as the `globalServiceID` field. For proper operation this requires that `globalServiceID` be present in the SLT. See A/331 [3] Section 6.3 and A/351 [38] Section 5. This returns exactly one service guide fragment.

`content` – A fully-qualified URI, required when `sgType="Content"`, shall provide the `globalContentID` found in the content fragment that can be used by the Broadcaster Application to uniquely identify a specific content item (of which there may be many) on a service. The service guide is delivered in XML fragments whose syntax is defined in A/332 [4]. This returns exactly one service guide fragment. This field may be empty if the `globalContentID` is not present in the service guide content fragment.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – If there is no ESG service, and thus no fragments to return, the error shall be -4 "Content not found".
- -6 – Service not found. The requested service cannot be located.

For example, the Broadcaster Application makes a query:

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

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "urlList": [
          { "sgType": "Service",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Service.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B" },
          { "sgType": "Schedule",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Schedule.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B" },
          { "sgType": "Content",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Content.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B",
            "content": "urn:eidr:10.5240:7791-8534-2C23-9030-8610-5" }
        ]
    },
    "id": 913
}
```

Note that the URLs provided are examples only. The actual URLs used, including the file names, are completely dependent on the Receiver implementation and how it chooses to make the ESG files available through its HTTP server. The Broadcaster Application should make no assumptions regarding the URL path and simply use it to access the fragment data directly.

The referenced service guide files, in this example, `Service.xml`, `Schedule.xml` and `Content.xml`, shall contain the Service, Schedule and Content XML fragments as described in A/332 [4], respectively. The Receiver is expected to extract each XML fragment from the binary SGDU structure before making it available to the Broadcaster Application.

To associate ESG files with Broadcaster Applications, the corresponding Application Context Identifiers shall be provided in the Extended FDT (EFDT) element, **`FDT-Instance`**`@appContextIdList`, defined when sending the ESG files in the LCT channel of the ESG Service ROUTE session. Descriptions of the FDT extensions and the ESG Service can be found in A/331 [3]. Application Context Identifiers need not be included in the EFDT if the ESG data is not needed by the Broadcaster Application.

9.2.10 Query Signaling Data API

The Broadcaster Application may wish to access the various signaling metadata structures from the current broadcast. In the case of Redistribution (in which broadcast signaling metadata is not available), the Broadcaster Application may wish to access the signaling metadata structures that were obtained by the Receiver via content recovery, which includes the Recovery Data Table (RDT) as defined in A/336 [5], Table 5.30 and can include other metadata as enumerated in A/331 [3]. See Table 9.22 below. The Query Signaling Data API returns a list of signaling tables that the

Broadcaster Application can use to extract details not otherwise available such as major and minor channel numbers.

The Query Signaling Data Request semantics are defined in Table 9.21 and the syntax shall be as defined in the schema file `org.atsc.query.signaling-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.21 Query Signaling Data Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.signaling"
`group`	0..1	integer	The group associated with any signaling returned.
`nameList`	1	array	A list of the requested signaling objects
`items`	0..N	string or integer	See `names` definition below

`group` – This optional parameter specifies the signaling group (see [3] Section 5.5) of the metadata objects requested. Requested metadata objects shall only be returned if they are part of the signaling group specified. If no signaling group is specified, the Receiver may choose to send all metadata objects discovered or only metadata objects in the same group as the HELD which launched the Broadcaster Application making the request.

`nameList` – An array of signaling object names as described below in `names`. If empty, no metadata objects are returned.

`names` – This field shall be set to a list of the values specified in the "names" column of Table 9.22. When the `names` field is empty, this request shall return no metadata objects. Some metadata objects are transport dependent (ROUTE versus MMT) and might not be available on a given Receiver. Note that LLS tables can be delivered via a SignedMultiTable. In the Redistribution case, only the RDT and metadata objects downloaded with the RDT are available to be returned.

The `<other>` name is a placeholder for any string or number. If the Broadcaster Application provides a string or number that is unknown to the Receiver, the Receiver is expected to ignore the request. If the string or number is known to the Receiver and the metadata object is present, the Receiver is expected to return the metadata object.

Table 9.22 Signaling Metadata Object Name Definitions

Names	Description	Reference
ROUTE / DASH Signaling		
`USBD`	User Service Bundle Description	[3] Section 7.1.3
`STSID`	Service-based Transport Session Instance Description	[3] Section 7.1.4
`MPD`	DASH Media Presentation Description	[3] Section 7.1.5
`APD`	Associated Procedure Description	[3] Section 7.1.7
MMT Signaling		
`USD`	User Service Description for MMTP	[3] Section 7.2.1
`PAT`	MMT Package Access Table	[3] Section 7.2.3
`MPT`	MMT Package Table	[3] Section 7.2.3
`MPIT`	MMT Media Presentation Information Table	[3] Section 7.2.3

CRIT	MMT Clock Relation Information Table	[3] Section 7.2.3
DCIT	MMT Device Capabilities Information Table	[3] Section 7.2.3
MMT Message Signaling		
AEI	MMT Application Event Information	[6] Section 4.1.2
VSPD	Video Stream Properties Descriptor	[3] Section 7.2.3.2
ASD	ATSC Staggercast Descriptor	[3] Section 7.2.3.3
IED	Inband Event Descriptor	[6] Section 4.1.2
CAD	Caption Asset Descriptor	[3] Section 7.2.3.5
ASPD	Audio Stream Properties Descriptor	[3] Section 7.2.3.4
SSD	Security Properties Descriptor	[3] Section 7.2.4.2
Event Signaling		
EMSG	ROUTE/DASH Application Dynamic Event	[6] Section 4.2
EVTI	MMT Application Dynamic Event	[6] Section 4.1.2
Other Signaling		
HELD	HTML Entry pages Location Description	[3] Section 7.1.8
DWD	Distribution Window Description	[3] Section 7.1.9
RSAT	Regional Service Availability Table	[3] Section 7.1.10
RDT	Recovery Data Table	[5] Section 5.4.1
Low-Level Signaling (LLS)		
SLT or 1	Service List Table, LLS_table_id=1	[3] Section 6.3
RRT or 2	Region Rating Table, LLS_table_id=2	[3] Annex F
STT or 3	SystemTime Table, LLS_table_id=3	[3] Section 6.2
AEAT or 4	Advance Emergency Information Table, LLS_table_id=4	[3] Section 6.5
OSN or 5	Onscreen Message Notifications, LLS_table_id=5	[3] Section 6.6
SMT or 254	Signed Multitable, LLS_table_id=0xFE (254)	[3] Section 6.7
CDT or 6	CertificateData Table, LLS_table_id=6	[8] Section 5.2.2.2
<other>	String or number associated with a metadata object not explicitly named in the "**names**" column.	

The Query Signaling Data Response semantics are defined in Table 9.23 and the syntax shall be as defined in the schema file org.atsc.query.signaling-response.json. Additional semantic definitions of parameters follow the table.

Table 9.23 Query Signaling Data Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
objectList	1	array	Lists the signaling tables based on the requested names, if specified in the request, or all available signaling tables if not.
items	0..N		
name	1	string or integer	See names definition above
version	1	integer	The version of the signaling element

			group	0..1	integer	Required for LLS tables. Provides the LLS group ID.
			table	1	string (XML or JSON or Base64)	The signaling table data
			encoding	0..1	string	The content encoding if not UTF-8
error				oneOf X		See Section 8.3.3

`objectList` – Provides an array, perhaps empty (i.e., the requested table does not exist in the broadcast), of the signaling data found in response to the Signaling Data request.

`name` – See `names` above.

`version` – This is the version of the XML signaling document. For LLS, it shall be set to the value of `LLS_table_version`. For Metadata Object Types and for the RDT, it shall be set to **`metadataEnvelope`**`@version`.

`group` – For LLS tables, this is required and shall be the value of `LLS_group_id`. For Metadata Object Types and for the RDT, this shall be omitted.

`table` – This string shall contain a single object of a type matching `name`, encoded in UTF-8. For objects that are not encoded in UTF-8, they shall first be encoded as Base64 by the Receiver. The XML and JSON documents shall be uncompressed. For both LLS and SLS documents, only the XML document itself is returned. For an RDT, only the JSON document itself is returned. Related packaging including signatures, binary LLS fields (e.g., `group_count_minus1`), MIME separators, etc. shall be removed.

`encoding` – This optional string shall contain a single token, `"Base64"` when the table has been Base64 encoded. For objects that are UTF-8, this field is not needed as the default is UTF-8.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – If there are no tables available, and thus no tables to return, the error shall be -4 "Content not found".

For example, the application makes a query for the SLT and MPD as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.signaling",
    "params": {
        "nameList": [1, "MPD"]
    },
    "id": 913
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "objectList": [
          { "name": 1,
            "version": 23,
            "group": 1,
            "table": "<SLT …… </SLT>" },
          { "name": "MPD",
            "version": 65,
            "table": "<MPD …… </MPD>" }
        ]
    },
    "id": 913
}
```

9.2.11 Query Dialog Enhancement Preferences API

The Broadcaster Application may wish to know the user preferences of the Dialog Enhancement capabilities, including the state and amount of Dialog Enhancement processing. The Broadcaster Application may use the Query Dialog Enhancement Preferences API to determine these settings.

The Query Dialog Enhancement Preferences Request semantics are defined in Table 9.24 and the syntax shall be as defined in the schema file `org.atsc.query.dialogEnhancementPref-request.json`.

Table 9.24 Query Dialog Enhancement Preferences Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.dialogEnhancementPref"

The Query Dialog Enhancement Preferences Response semantics are defined in Table 9.25 and the syntax shall be as defined in the schema file `org.atsc.query.dialogEnhancementPref-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.25 Query Dialog Enhancement Preferences Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`dialogEnhancementPref`	1	integer	The user's dialog enhancement preference gain value in dB
`error`	oneOf X		See Section 8.3.3

`dialogEnhancementPref` – The user's preference gain value in dB for the Dialog Enhancement processing to be applied in the audio decoder.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.dialogEnhancementPref",
    "id": 92
}
```

If the user suffers from slight hearing impairment and has enabled the dialog enhancement in the preferences menu, the Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "dialogEnhancementPref": 6
    },
    "id": 92
}
```

9.2.12 Query Display Components API

The Broadcaster Application might request the Receiver settings for the position and size of the current caption display region and the video window and if the Receiver supports the scaling of either closed captioning or the video window. The Broadcaster Application may use the Query Display Components API to determine these settings.

The Query Display Components Request semantics are defined in Table 9.26 and the syntax shall be as defined in the schema file org.atsc.query.displayComponents-request.json.

Table 9.26 Query Display Component Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.query.displayComponents"

The Query Display Components Response semantics are defined in Table 9.27 and the syntax shall be as defined in schema file org.atsc.query.displayComponents-response.json.

Table 9.27 Query Display Component Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
result	oneOf X		Returned on successful request otherwise the error structure is returned
videoWindowScalingSupported	1	boolean	Indicates, if "true", that scaling of the video window is supported
captionScalingSupported	1	boolean	Indicates, if "true", that scaling of the closed captioning relative to the scaling of the video widow is supported

	activeCaptionRegions	1	array	A list of caption regions that are actively being rendered. The list may be empty if no closed captioning is being displayed
	items	0..N		
	captionRegionOriginX	1	integer	Provides the current origin x-axis coordinate of the caption display region in terms of number of pixels from the upper-left corner of the screen
	captionRegionOriginY	1	integer	Provides the current origin y-axis coordinate of the caption display region in terms of number of pixels from the upper-left corner of the screen
	captionRegionExtentX	1	integer	Provides the width of the current caption display region in number of pixels
	captionRegionExtentY	1	integer	Provides the height of the current caption display region in number of pixels
error		oneOf X		See Section 8.3.3

`videoWindowScalingSupported` – See `videoWindowScalingSupported` above.
`captionScalingSupported` – See `captionScalingSupported` above.
`activeCaptionRegions` – See `activeCaptionRegions` above.
`captionRegionOriginX` – See `captionRegionOriginX` above.
`captionRegionOriginY` – See `captionRegionOriginY` above.
`captionRegionExtentX` – See `captionRegionExtentX` above.
`captionRegionExtentY` – See `captionRegionExtentY` above.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

The `activeCaptionRegions` reported by the Receiver are not intended to represent a snapshot of the precise area of the captions at a given moment. The Receiver can report the largest possible regions based on the current font size and position, as well as the maximum expected number of characters and rows. This helps to avoid a display conflict situation that could otherwise occur as the caption text changes. When captions are disabled, the number of `activeCaptionRegions` should be zero.

9.2.13 Query Announcement Time Limit

The Broadcaster Application might request to know the Receiver setting for a limit to the amount of time that it may present a "call-to-action" to the viewer to announce its presence and provide the viewer an opportunity to engage. The Broadcaster Application may use the Query Announcement Time Limit API to determine these settings.

The Query Announcement Time Limit Request semantics are defined in Table 9.28 and the syntax shall be as defined in the schema file org.atsc.query.announcementTimeLimit-request.json.

Table 9.28 Query Time Limit Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.query.announcementTimeLimit"

The Query Announcement Time Limit Response semantics are defined in Table 9.29 and the syntax shall be as defined in schema file `org.atsc.query.announcementTimeLimit-response.json`.

Table 9.29 Query Time Limit Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`announcementTimeLimit`	1	integer	Provides a current setting for a time value in seconds that limits the amount of time a call-to-action may be presented on screen to the viewer so that a Broadcaster Application may announce itself.
`error`	oneOf X		See Section 8.3.3

`announcmentTimeLimit` – This required integer indicates a time in number of seconds that limits the time graphics from a Broadcaster Application may be presented on screen before a viewer has made a selection to enable the display of Broadcaster Application graphics. This announcement time limit might have been set by the Receiver as a default or a viewer by user selection. An `announcementTimeLimit` set to `0` shall indicate that there is not an announcement time limit available to communicate to the Broadcaster Application. Any non-zero `announcementTimeLimit` value shall be greater than or equal to 3.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

9.3 Asynchronous Notifications of Changes

The types of notifications that the Receiver is expected to provide to the Broadcaster Application through the APIs defined in this section are as specified in Table 9.30. All use the method `org.atsc.notify` and include a parameter called "`msgType`" to indicate the type of notification.

Table 9.30 Asynchronous Notifications

msgType	Event Description	Reference
`ratingBlock`	Content Advisory Rating Block Change – a notification that is provided whenever the user changes the content advisory rating settings in the Receiver such that the current content being decoded goes from blocked to unblocked or unblocked to blocked.	Section 9.3.2
`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.	Section 9.3.3
`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).	Section 9.3.4
`langPref`	Language Preference – a notification that is provided whenever the user changes the preferred language.	Section 9.3.5

msgType	Event Description	Reference
captionDisplayPrefs	Closed Caption display properties preferences.	Section 9.3.6
audioAccessibilityPref	Audio Accessibilities preferences.	Section 9.3.7
alertingChange	Alerting Change – a notification that a new version of either the AEAT or OSN messages has been received or if alert filtering preferences have been changed resulting in events becoming unfiltered.	Section 9.3.8
contentChange	Content Change – a notification that new content has been placed in the Application Context Cache and may be accessed by the Broadcaster Application.	Section 9.3.9
serviceGuideChange	Service Guide Change – a notification that is provided when new ESG fragments have been received.	Section 9.3.10
signalingData	Signaling Data Change – a notification that some new signaling data has been received.	Section 9.3.11
dialogEnhancementPrefChange	Dialog Enhancement Preference Change – a notification that is provided whenever the user changes the state or amount of dialog enhancement processing in the user's preferences.	Section 9.3.12
dialogEnhancementLimitChange	Dialog Enhancement Limit Change – a notification that is provided whenever the incoming audio stream signals a changed limit for dialog enhancement processing.	Section 9.3.13
rfSignalChange	RF Signal Change – a notification that is provided whenever some aspect of the received RF signal changes.	Section 9.3.14
contentRecoveryStateChange	Content Recovery State Change – a notification that is provided whenever use of audio watermark, video watermark, audio fingerprint, and/or video fingerprint for content recovery changes.	Section 9.9.4
displayOverrideChange	Display Override Change – a notification that is provided if the display override state or the state of blocked application access to certain resources changes.	Section 9.9.5
recoveredComponentInfoChange	Recovered Component Info Change – a notification that is provided if a component of the service being received by the Receiver changes at the upstream.	Section 9.9.6
rmpMediaTimeChange	RMP Media Time Change – a notification that is provided periodically during playback.	Section 9.13.5
rmpPlaybackStateChange	RMP Playback State Change – a notification that is provided if the playback state changes.	Section 9.13.3
rmpPlaybackRateChange	RMP Playback Rate Change – a notification that is provided if playback speed changes.	Section 9.13.7
DRM	DRM Notification – a notification that provides messages from the content protection system to the Broadcaster Application.	Section 9.14.1
xlinkResolution	XLink Resolution – a notification that is provided when the RMP encounters a period with an XLink attribute.	Section 9.15.1
assetLinkResolution	Asset Link Resolution – a notification that is provided when the RMP encounters a target Asset available for replacement.	Section 9.17.1

9.3.1 Integrated Subscribe / Unsubscribe API for Notifications

The various types of notifications that the Receiver may provide, except for the Event Stream Notifications defined in Section 9.6, are specified in the remaining subsections of Section 9.2.12. The Broadcaster Application may wish to receive specific notifications. When a Broadcaster

Application starts, no notifications are subscribed, and the Receiver is not expected to send any notifications. The Broadcaster Application may use the subscription API to begin receiving the desired notifications. The Receiver is expected to send subscribed notifications until the Broadcaster Application requests the Integrated Unsubscribe API. In addition, once the Broadcaster Application has been terminated, the Receiver is expected to automatically unsubscribe all the subscribed notifications that were requested by the Broadcaster Application.

Two APIs are needed to support this function:

- Integrated Subscribe API
- Integrated Unsubscribe API

Table 9.31 describes the list of msgTypes to subscribe to each of the notifications. The msgType column list is identical to the enum parameter of the subscribe and unsubscribe APIs.

Note that the event stream notification and its associated subscription methods are specified in Section 9.6. This separate interface allows parameters to be specified to filter various event streams.

Table 9.31 Subscription Parameter List

Notification APIs	Reference	msgType
All Notification APIs	-	All
Rating Block Change Notification API	9.3.2	ratingBlock
Service Change Notification API	9.3.3	serviceChange
Caption State Change Notification API	9.3.4	captionState
Language Preference Change Notification API	9.3.5	languagePref†
Caption Display Preferences Change Notification API	9.3.6	captionDisplayPrefs
Audio Accessibility Preference Change Notification API	9.3.7	audioAccessibilityPref
Alerting Change Notification API	9.3.8	alertingChange
Content Change Notification API	9.3.9	contentChange
Service Guide Change Notification API	9.3.10	serviceGuideChange
Signaling Data Change Notification API	9.3.11	signalingData
Dialog Enhancement Preference Change Notification API	9.3.12	dialogEnhancementPrefChange
Dialog Enhancement Limit Change Notification API	9.3.13	dialogEnhancementLimitChange
RF Signal Change Notification API	9.3.14	rfSignalChange
Content Recovery State Change Notification API	9.9.4	contentRecoveryStateChange
Display Override Change Notification API	9.9.5	displayOverrideChange
Recovered Component Info Change Notification API	9.9.6	recoveredComponentInfoChange
RMP Media Time Change Notification API	9.13.5	rmpMediaTimeChange
RMP Playback State Change Notification API	9.13.6	rmpPlaybackStateChange
RMP Playback Rate Change Notification API	9.13.7	rmpPlaybackRateChange
RMP Media Asset Change Notification API	9.13.8	rmpMediaAssetChange
DRM Notification API	9.14.1	DRM
XLink Resolution Notification API	9.15.1	xlinkResolution
AssetLink Resolution Notification API	9.17.1	assetLinkResolution

†Note that the name of the Language Preference Change Notification API `"msgType"` defined in Section 9.3.5 is `"langPref"`. This has been intentionally maintained to avoid backward-compatibility issues.

9.3.1.1 Integrated Subscribe API

The Subscribe Request semantics are defined in Table 9.32 and the syntax shall be as defined in the schema file org.atsc.subscribe-request.json. Additional semantic definitions of parameters follow the table.

Table 9.32 Subscribe Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.subscribe"
msgType	1	array	A list of the notifications to which the Broadcaster Application is requesting to subscribe
items	0..N	examples	One of the msgTypes from Table 9.31 "msgType" Column
signalingDataList	0..1	array	A list of the requested signaling objects if the "signalingData" msgtType is requested
Items	0..N	string or integer	See names definition in Section 9.2.10

msgType – An array of notification msgTypes from the "msgType" column in Table 9.31 for which the Broadcaster Application is requesting to subscribe. If empty, this request performs no operation. Use the "All" enum value to subscribe to all notifications.

signalingDataList – An array of signaling object names as described in the names description in Section 9.2.10. This field is only applicable if the "signalingData" msgType is included in the msgType list. If empty, no metadata objects are returned.

The Subscribe Response semantics are defined in Table 9.33 and the syntax shall be as defined in the schema file org.atsc.subscribe-response.json. Additional semantic definitions of parameters follow the table.

Table 9.33 Subscribe Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
msgType	1	array	Lists the notifications to which the Broadcaster Application is subscribed
items	0..N	examples	One of the msgTypes from Table 9.31 "msgType" Column
error	oneOf X		See Section 8.3.3

msgType – An array of notification msgTypes from the "msgType" column in Table 9.31 for which the Broadcaster Application is subscribed.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned. Note there is no error for duplicate subscribe requests.

- None – There are no errors specific to this API.

For example, the Broadcaster Application wants to subscribe to the "alertingChange", "ratingBlock" and "contentChange" notifications:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.subscribe",
    "params": {
        "msgType": ["alertingChange", "ratingBlock", "contentChange"]
    },
    "id": 51
}
```

Upon success, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "msgType": ["alertingChange", "ratingBlock", "contentChange"]
    },
    "id": 51
}
```

If the Broadcaster Application may attempt to subscribe to all notifications:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.subscribe",
    "params": {
        "msgType": ["All"]
    },
    "id": 51
}
```

If the Receiver supports all functionality except content recovery, it may respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "msgType": ["ratingBlock", "serviceChange", "captionState",
"languagePref", "captionDisplayPrefs", "audioAccessibilityPref",
"alertingChange", "contentChange", "rmpMediaTimeChange",
"rmpPlaybackStateChange", "rmpPlaybackRateChange", "DRM", "xlinkResolution"]
    },
    "id": 51
}
```

9.3.1.2 Integrated Unsubscribe API

The Unsubscribe Request semantics are defined in Table 9.34 and the syntax shall be as defined in the schema file org.atsc.unsubscribe-request.json. Additional semantic definitions of parameters follow the table.

Table 9.34 Unsubscribe Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.unsubscribe"
msgType	1	array	A list of the notifications from which the Broadcaster Application is requesting to unsubscribe
items	0..N	examples	One of the msgTypes from Table 9.31 "msgType" Column

`msgType` – An array of notification msgTypes from the "msgType" column in Table 9.31 for which the Broadcaster Application is requesting to unsubscribe from. If empty, this request performs no operation. Use the "All" enum value to unsubscribe from all notifications.

The Unsubscribe Response semantics are defined in Table 9.35 and the syntax shall be as defined in the schema file `org.atsc.unsubscribe-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.35 Unsubscribe Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
msgType	1	array	Lists the notifications from which the Broadcaster Application is unsubscribed
items	0..N	examples	One of the msgTypes from Table 9.31 "msgType" Column
error	oneOf X		See Section 8.3.3

`msgType` – An array of notification msgTypes from the "msgType" column in Table 9.31 from which the Broadcaster Application is unsubscribed.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -24: The Broadcaster Application was not subscribed to any of the requested notifications

Note that unsubscribing from "All" notifications is expected to unsubscribe from any notification listed in Table 9.31. An error is expected to only occur if there are currently no outstanding subscriptions for any of the notifications.

For example, the Broadcaster Application wants to unsubscribe from all notifications:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.unsubscribe",
    "params": {
        "msgType": ["All"]
    },
    "id": 52
}
```

Upon success, the Receiver would respond with the list of msgTypes from which the Broadcaster Application successfully unsubscribed:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "msgType": ["alertingChange", "contentChange",
"contentRecoveryStateChange", "displayOverrideChange",
"recoveredComponentInfoChange", "rmpMediaTimeChange",
"rmpPlaybackStateChange", "rmpPlaybackRateChange"]
    },
    "id": 52
}
```

9.3.2 Content Advisory Rating Block Change Notification API

The Content Advisory Rating Block Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if there is a change to the content advisory rating blocking of the currently displayed service, either from unblocked to blocked or vice versa.

When the service is blocked, the Broadcaster Application may have restricted access to APIs, for example, the display. When the service is unblocked, the Broadcaster Application is expected to resume normal operations.

In addition to the blocked status, the Receiver might also provide the content advisory rating of the currently displayed service to allow the Broadcaster Application to determine why and perhaps inform the user why the content has been blocked. Note that this notification is not expected to be issued if there is no change to the blocked status even when the content advisory rating may change.

Note that Content Advisory Ratings, downloadable rating region table(s) and parental control may be addressed by law or regulation.

The Content Advisory Rating Block Change Notification semantics are defined in Table 9.36 and the syntax shall be as defined in the schema file `org.atsc.notify-ratingBlock.json`. Additional semantic definitions of parameters follow the table.

Table 9.36 Content Advisory Rating Block Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"ratingBlock"
`blocked`	1	boolean	Indicates whether content is blocked or not
`contentRating`	0..1	string	The new value of the content advisory rating provided by the current service signaling

`blocked` – A required Boolean value shall represent the state of blocking after the state changes. This could be a result of user action (e.g., a change to the parental control settings) or a change to the ratings of the current content.

`contentRating` – An optional string containing the content advisory rating of the currently displayed service as provided in the service signaling. The `contentRating` string shall conform to the encoding specified in A/331 [3], Section 7.3. Note that the content rating string is expected to contain all of the rating values including from multiple rating regions if appropriate. To specify content advisory information data for multiple rating regions, additional three-part strings (one for each region) shall be concatenated to create one string consisting of multiple concatenated three-part strings. In this case, the third part of each content advisory information string except the last shall be followed by a comma (","). Thus, the last character of the entire content advisory ratings string is a right curly brace ("}"). The referenced encoding from A/331 is that used by ROUTE DASH. MMT encodes ratings according to A/332 [4].

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

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "ratingBlock",
        "blocked": true,
        "contentRating": "1,'TV-PG-L', {0 'TV-PG'}{1 'L'}"
    }
}
```

9.3.3 Service Change Notification API

The Service Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application under conditions described in Application Lifecycle, Section 6.3.

The Service Change Notification semantics are defined in Table 9.37 and the syntax shall be as defined in the schema file `org.atsc.notify-serviceChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.37 Service Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"serviceChange"
`service`	1	string (uri)	Provides the `globalServiceID` of the newly acquired service
`requested`	0..1	boolean	Indicates that a new `service` is being requested by the user or Broadcaster Application

`service` – The required `service` property shall provide the `globalServiceID` associated with the newly acquired service as specified in the Query Service ID response API (See Section 9.2.3). Note that the Receiver is not expected to notify the Broadcaster Application for service types that do not require `globalServiceID` to be defined, such as DRM, ESG, and NRT services.

`requested` – The optional `requested` property, when set to "true", indicates that the `service` is in the process of being requested and is not yet acquired. When `requested` is set to "false" or is absent, a new service has been acquired (and implicitly indicates that the Broadcaster Application has a common `@appId` and `@appContextId` on the new service).

In the following example, the user has caused a service change to a service with a `globalServiceID` "`https://doi.org/10.5239/8A23-2B0B`":

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "serviceChange",
        "service": "https://doi.org/10.5239/8A23-2B0B"
    }
}
```

9.3.4 Caption State Change Notification API

The Caption State Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application when the state of the caption display has changed.

The Caption State Change Notification semantics are defined in Table 9.38 and the syntax shall be as defined in the schema file `org.atsc.notify-captionState.json`. Additional semantic definitions of parameters follow the table.

Table 9.38 Caption State Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"captionState"
`captionDisplay`	1	boolean	Indicates whether closed captioning is being displayed or not

`captionDisplay` – A required Boolean value representing the new state of closed caption display. A `"true"` value indicates captions are being displayed while a `"false"` value indicates they are not.

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

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

9.3.5 Language Preference Change Notification API

The Language Preference Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if the user changes the preferred language applicable

to either audio, user interfaces, subtitles/captions, or overall. Only the preferences that have changed are expected to be provided in this notification.

The Language Preference Change Notification semantics are defined in Table 9.39 and the syntax shall be as defined in the schema file org.atsc.notify-langPref.json. Additional semantic definitions of parameters follow the table.

Table 9.39 Language Preference Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"langPref"
preferredUILang	0..1	string	Provides the new preferred language of the Receiver User Interfaces
preferredAudioLang	0..1	string	Provides the new preferred language for the audio output
preferredCaptionSubtitleLang	0..1	string	Provides the new preferred language of the closed captions or subtitles

preferredUILang, preferredAudioLang, preferredCaptionSubtitleLang – Each of these optional strings shall conform to the semantics described in the Query Language Preferences response API (see Section 9.2.4).

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"}
    }
}
```

9.3.6 Caption Display Preferences Change Notification API

The Caption Display Preferences Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if there are changes in preferences for display of closed captioning.

The Caption Display Preferences Change Notification semantics are defined in Table 9.40 and the syntax shall be as defined in the schema file org.atsc.notify-captionDisplayPrefs.json. Note that the semantics of the caption display preferences, "cta708", are defined in Section 9.2.5.1 while the IMSC1 [48] attributes, "imsc1", (as defined in A/343 [7]) as specified in Section 9.2.5.2. The IMSC1 attributes are defined by text in Section 9.2.5.2 but are intended to be an integral part of the JSON schema in this section.

Table 9.40 Caption Display Preferences Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"captionDisplayPrefs"
cta708	0..1	object	See semantic definition in Section 9.2.5.1
imsc1	0..1	object	See semantic definition in Section 9.2.5.2

For example, the Receiver notifies the Broadcaster Application that the user has changed their caption display preferences to red text on gray background. All the available 708 parameters along with two IMSC1 parameters are included in this example:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
      "msgType": "captionDisplayPrefs",
      "cta708": {
        "characterColor": "#FF0000",
        "characterOpacity": 0.5,
        "characterSize": 100,
        "fontStyle": "MonospacedSerifs",
        "backgroundColor": "#808080",
        "backgroundOpacity": 0.25,
        "characterEdge": "Raised",
        "characterEdgeColor": "#000000",
        "windowColor": "#000000",
        "windowOpacity": 0
      },
      "imsc1": {
        "region_textAlign": "center",
        "content_fontWeight": "bold"
      }
    }
}
```

9.3.7 Audio Accessibility Preference Change Notification API

The Audio Accessibility Preference Change Notification is expected to 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 Audio Accessibility Preference Change Notification semantics are defined in Table 9.41 and the syntax shall be as defined in the schema file org.atsc.notify-audioAccessibilityPref.json. Additional semantic definitions of parameters follow the table.

Table 9.41 Audio Accessibility Preference Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"audioAccessibilityPref"
videoDescriptionService	0..1		

	enabled	0..1	boolean	Indicates whether or not a video description service is enabled
	language	0..1	string	The preferred language of the video description service
audioEIService		0..1		
	enabled	0..1	boolean	Indicates whether or not emergency information audio is enabled
	language	0..1	string	The preferred language of the emergency information audio

`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 BCP 47 [21]. 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 Receiver. If `videoDescriptionService.enabled` is equal to `true` and the preferred language of VDS rendering is not available, the Receivers shall not include the `videoDescriptionService.language` property.

`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 BCP 47 [21]. 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 Receiver. If `audioEIService.enabled` is equal to `true` and the preferred language is not available, the Receivers shall not include the `audioEIService.language` property.

For example, if the user has changed the video description service's accessibility preference to ON, the Receiver notifies the Broadcaster 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"
        }
    }
}
```

9.3.8 Alerting Change Notification API

The Alerting Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if there is a change to the version of the AEAT or OSN alerting data structure and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1. Note that the receipt of a new alerting object without a previous

receipt is considered a version change. The Alerting Change Notification may also be issued if alerting event filtering has been changed resulting in a change to the filtered events list.

The notification message contains a list of new or updated alerting fragments.

The Alerting Change Notification semantics are defined in Table 9.42 and the syntax shall be as defined in the schema file org.atsc.notify-alertingChange.json. Additional semantic definitions of parameters follow the table.

Table 9.42 Alerting Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"alertingChange"
alertList	1	array	A list containing the new or updated alerting fragments
items	0..N		
alertingType	1	enum	"AEAT" or "OSN"
alertingFragment	1	string (xml)	The XML fragment of the associated alerting type
receiveTime	0..1	string (date-time)	If alertingType = "OSN", the date and time when the fragment was received
filteredEventList	0..1	array	Provides an array of AEA IDs which have been filtered out by the Receiver
items	1..N	string	

`alertList` – A required array containing a list of new or updated alerting fragments. This notification is not expected to occur if the array is empty, that is, there is no change.

`alertingType` – A required parameter containing one of "AEAT" or "OSN". The corresponding `alertingFragment` shall contain the XML fragment corresponding to the `alertingType`.

`alertingFragment` – A required string shall contain the alerting XML fragment for the associated `alertingType`. The AEAT XML and OSN XML fragments are extracted from their respective LLS tables which are described in A/331 [1].

`receiveTime` – The date and time when the OSN fragment was received. This value shall be provided when the `alertingType` is "OSN" and is optional otherwise. (Note: The OSN table includes a Notification Duration field which is the duration of the `KeepScreenClear` message starting from the time the OSN was received. Thus, the time the OSN was received is necessary for the Broadcaster Application to fully utilize the OSN information.) The date-time JSON data type shall be formatted as defined in the JSON Schema specification [19].

`filteredEventList` – Provides a list of AEA events that have been filtered out by the Receiver. The Receiver may elect to filter out an event for a variety of reasons based on user preferences, location or some other criteria. If an `AEA` event is filtered out, the corresponding `AEAT.AEA@aeaId` shall appear in the `filteredEventList` property. If an `AEA` event is not filtered out, the corresponding `AEAT.AEA@aeaId` shall not appear in the list. An empty or absent `filteredEventList` indicates that no events have been filtered out by the Receiver. This property is only applicable when the `alertingType` is "AEAT". AEA events that have been "filtered out" are those that have been handled or processed by the Receiver and need not be processed by the Broadcaster Application.

Note that if the filtering criteria change due to, for example, the Receiver moving, or the user

changing preferences, previously filtered out events may become unfiltered and events that were previously unfiltered may now be filtered out. In this case, the Alerting Change Notification shall be issued by the Receiver with a new `filteredEventList`. The Broadcaster Application may need to take alternative actions based on the new list of events.

For example, the Receiver may indicate that a new AEAT has been received by issuing this JSON-RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "alertingChange",
        "alertList": [
          { "alertingType": "AEAT",
            "alertingFragment": "<AEAT>…</AEAT>" }
        ]
    }
}
```

As a further example, the Receiver may indicate that a new AEAT and OSN have been received by issuing this JSON-RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "alertingChange",
        "alertList": [
          { "alertingType": "AEAT",
            "alertingFragment": "<AEAT>…</AEAT>" },
          { "alertingType": "OSN",
            "alertingFragment": "<OSN>…</OSN>",
            "receiveTime": "2017-01-01T23:54:59.590Z" }
        ]
    }
}
```

9.3.9 Content Change Notification API

The Content Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if a new signed package or new version of a signed package has been received and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1. Note that a signed package is considered "received" if the contained files are available through the Receiver Web Server.

The notification message contains a list of URIs referencing the received packages.

The Content Change Notification semantics are defined in Table 9.43 and the syntax shall be as defined in the schema file `org.atsc.notify-contentChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.43 Content Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"contentChange"
`packageList`	1	array	A list containing newly received package URIs
items	0..N	string (uri)	The package URI of a specific package delivered over ROUTE whose files are now available in the Application Context Cache

`packageList` – An array of packages whose contents have been received, checked for signing and are now available for Broadcaster Application access. The package URI of a specific package delivered over ROUTE shall be obtained from the value signaled in the **`EFDT.FDT-Instance.File`**`@Content-Location` attribute according to A/331 [3]. Broadcaster Applications may use these package URIs to determine which collection of files have been received or updated.

For example, to notify the Broadcaster Application that new versions of various content files from a particular package have been received, the content signing has been verified and the files are now available through the Receiver Web Server, the Receiver may issue the following JSON-RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "contentChange",
        "packageList": ["http://192.168.1.10/items/zz/content"]
    }
}
```

The Broadcaster Application may elect to reload itself or any portion of itself when such a notification is received.

9.3.10 Service Guide Change Notification API

The Service Guide Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if there is a change to some portion of the service guide data structures and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1. Note that the receipt of a new service guide fragment without a previous receipt is considered a version change.

The notification message contains a list of URLs referencing the new or updated service guide XML fragments.

The Service Guide Change Notification semantics are defined in Table 9.44 and the syntax shall be as defined in the schema file `org.atsc.notify-serviceGuideChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.44 Service Guide Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"serviceGuideChange"
`urlList`	1	array	Lists the set of service guide URLs pointing to newly changed service guide fragments
items	0..N		
`sgType`	1	enum	`"Service"`, `"Schedule"` or `"Content"`
`sgUrl`	1	string (uri)	The XML fragment of the associated service guide type
`service`	1	string (uri)	The URI of the service related to the service guide type.
`content`	0..1	string (uri)	When the sgType = "Content", this parameter is used to provide the unique ID of the content, if available

`urlList` – A required array containing a list of URLs to new or updated service guide fragments.

This notification is not expected to occur if the array is empty, that is, there has been no change. The semantics of the properties `sgType`, `sgUrl`, `service` and `content` shall be as specified in Section 9.2.9 Query Service Guide URLs API response.

For example, the Receiver may indicate that a new schedule has been received by issuing this JSON-RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "serviceGuideChange",
        "urlList": [
          { "sgType": "Schedule",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Schedule.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B" }
        ]
    }
}
```

Note that the URLs provided are examples only. The actual URLs used, including the file names, are completely dependent on the Receiver implementation and how it chooses to make the ESG files available through its HTTP server.

As a further example, the Receiver may indicate that a new service information and associated schedule and content have been received by issuing this JSON RPC command:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "serviceGuideChange",
        "urlList": [
          { "sgType": "Service",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Service.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B" },
          { "sgType": "Schedule",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Schedule.xml",
            "service": "https://doi.org/10.5239/8A23-2B0B" },
          { "sgType": "Content",
            "sgUrl": "http://127.0.0.1:8080/wmbc.appctx/Content.xml",
            "servict": "https://doi.org/10.5239/8A23-2B0B",
            "content": "urn:eidr:10.5240:7791-8534-2C23-9030-8610-5" }
        ]
    }
}
```

The prefixes shown in these examples are informative only. The Broadcaster Application should make no assumptions regarding the path and should simply use it to access the fragment data directly.

The referenced service guide files, in this example, `Service.xml`, `Schedule.xml` and `Content.xml`, shall contain the service, schedule and content XML fragments as described in A/332 [4], respectively. The Receiver is expected to extract each XML fragment from the binary SGDU structure before making it available to the Broadcaster Application.

To associate ESG files with Broadcaster Applications, the corresponding Application Context Identifiers shall be provided in the Extended FDT (EFDT) element, **`FDT-Instance`**`@appContextIdList` defined when sending the ESG files in the LCT channel of the ESG Service ROUTE session. Descriptions of the FDT extensions can be found in A/331 [3] and the ESG Service in A/332 [4]. Application Context Identifiers need not be included in the EFDT if the ESG data is not needed by the Broadcaster Application.

9.3.11 Signaling Data Change Notification API

The Signaling Data Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if a new version of any LLS table or SLS fragment is received since either the Broadcaster Application subscribed to receive such update notifications for tables listed via the API specified in Section 9.3.1 or a version was previously notified. The Broadcaster Application may respond to the notification of a change to the signaling data by using the Query Signaling Data API specified in Section 9.2.10 to fetch a new copy.

Note that this notification is issued whenever any LLS change is detected, including the AEAT and OSN signaling. This signaling is independent of the Alerting Change Notification API (Section 9.3.8), that is, if the Broadcaster Application subscribes to both the Signaling Data Change and Alerting Change Notifications, then both notifications are expected to be issued when a new or changed AEAT or OSN fragment is detected.

The Signaling Data Change Notification semantics are defined in Table 9.45 and the syntax shall be as defined in the schema file `org.atsc.notify-signalingData.json`.

Table 9.45 Signaling Data Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"signalingData"
objectList	0..1	array	Lists the signaling tables that have changed resulting in this notification
items	0..N		
name	1	string or integer	See names definition in Section 9.2.10
version	1	integer	The version of the signaling element
group	0..1	integer	Required for LLS tables. Provides the LLS group ID.
table	1	string (XML or JSON or Base64)	The signaling table data
encoding	0..1	string	The content encoding if not UTF-8

objectList – The semantics of this optional property are defined in the property of the same name in the Query Signaling Data response API in Section 9.2.10. It is recommended that the Receiver include any tables that have changed as part of the notification objectList to avoid potential timing issues that could occur when the Broadcaster Application uses the Query Signaling Data API in response to the notification.

The following is an example notification:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "signalingData",
        "objectList": [
          { "name": 1,
            "version": 23,
            "group": 1,
            "table": "<SLT …… </SLT>" },
          { "name": "MPD",
            "version": 65,
            "table": "<MPD …… </MPD>" }
        ]
    }
}
```

9.3.12 Dialog Enhancement Preference Change Notification API

The Dialog Enhancement Preference Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if the user changes his or her preferences on dialog enhancement processing and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The Dialog Enhancement Preference Change Notification semantics are defined in Table 9.46 and the syntax shall be as defined in the schema file org.atsc.notify-dialogEnhancementPrefChange.json. Additional semantic definitions of parameters follow the table.

Table 9.46 Dialog Enhancement Preference Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"dialogEnhancementPrefChange"
dialogEnhancementPref	1	integer	The user's new dialog enhancement preference gain value in dB

`dialogEnhancementPref` – This required property shall conform to the semantics described for the property with the same name in the Query Dialog Enhancement Preference response API (see Section 9.2.11).

For example, if the user changes Dialog Enhancement processing to a gain value of 9 dB in his or her preference settings, the Receiver would send the following notification:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "dialogEnhancementPrefChange",
        "dialogEnhancementPref": 9
    }
}
```

9.3.13 Dialog Enhancement Limit Change Notification API

The Dialog Enhancement Limit Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if the limit for dialog enhancement processing is changed in the currently decoded audio stream and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The Dialog Enhancement Limit Change Notification semantics are defined in Table 9.47 and the syntax shall be as defined in the schema file `org.atsc.notify-dialogEnhancementLimitChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.47 Dialog Enhancement Limit Change Notification Semantics

Property Name		Use	Data Type	Short Description
jsonrpc		1	string	"2.0"
method		1	string	"org.atsc.notify"
msgType		1	enum	"dialogEnhancementLimitChange"
dialogEnhancementLimit		1		
	max	1	integer	The maximum allowed dialog enhancement gain value in dB
	min	1	integer	The minimum allowed dialog enhancement gain value in dB

`dialogEnhancementLimit` – The range of allowed gain value in dB for the Dialog Enhancement processing to be applied in the audio decoder.

`max` – The maximum allowed gain value in dB for the Dialog Enhancement processing to be applied in the audio decoder.

`min` – The minimum allowed gain value in dB for the Dialog Enhancement processing to be applied in the audio decoder.

For example, if the decoded audio stream restricts Dialog Enhancement processing to a range of 0 dB to 6 dB in audio streams metadata, the Receiver would provide the following notification:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "dialogEnhancementLimitChange",
        "dialogEnhancementLimit": {
            "max": 6,
            "min": 0
        }
    }
}
```

9.3.14 RF Signal Change Notification API

The RF Signal Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if any of the properties listed in Table 9.48 change in the currently tuned RF channel and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The RF Signal Change Notification semantics are defined in Table 9.48 and the syntax shall be as defined in the schema file `org.atsc.notify-rfSignalChange.json`. Additional semantic definitions of the parameters follow the table.

Upon subscription, the Receiver is expected to promptly issue an initial notification. There should not be notifications more than once per second. The calculation of the field values is Receiver dependent.

Table 9.48 RF Signal Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"rfSignalChange"
`rfChannel`	1	integer	The RF channel number
`frequency`	1	integer	The frequency in Hertz
`signalQuality`	1	enum	"lost", "weak" or "strong"
`signalStrength`	0..1	integer (0 ... 100)	Signal strength represented as a percentage 0 to 100
`gainLevel`	0..1	integer (0 ... 100)	Gain level represented as a percentage 0 to 100
`bootstrapLock`	0..1	boolean	Indicates successful bootstrap symbol acquisition
`alpLock`	0..1	boolean	Indicates successful ALP data acquisition

`rfChannel` – This required property shall contain the RF channel number of the currently tuned RF channel.

`frequency` – This required property shall contain the center frequency in Hz of the currently tuned RF channel.

`signalQuality` – This required property shall provide a simplified overall RF quality aligned with the Android Media TV `onSignalStrengthUpdated` API [52]. The available values are "lost", "weak" or "strong".

`signalStrength` – This optional property shall provide the received signal strength reported as a percentage. The available values are in the range zero to 100, inclusive.

`gainLevel` – This optional property shall provide the average signal gain level of the tuner reported as a percentage. The available values are in the range zero to 100, inclusive.

`bootstrapLock` – This optional Boolean property shall confirm, if `"true"`, the A/321 defined symbol acquisition [1]. A `"false"` value shall indicate that bootstrap symbol acquisition cannot be confirmed.

`alpLock` – This optional Boolean property shall confirm, if `"true"`, the A/330 ATSC Link-Layer Protocol (ALP) data acquisition [2]. A `"false"` value shall indicate that ALP data acquisition cannot be confirmed.

For example, the Receiver might provide the following notification:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "RFSignalChange",
        "rfChannel": 15,
        "frequency": 479000000,
        "signalQuality": "strong",
        "signalStrength": 75,
        "gainLevel": 50,
        "bootstrapLock": true,
        "alpLock": true
    }
}
```

9.4 Cache Request APIs

The Cache Request APIs may be used by the currently executing Broadcaster Application to request that the Receiver download one or more objects from a broadband server and place them into a specified location in the Application Context Cache. Files may be identified individually by URL, or a DASH MPD or Period may be specified, in which case all the media files referenced by the MPD or Period are requested.

9.4.1 Cache Request API

The Broadcaster Application can use the Cache Request API to request that the Receiver download one or more indicated files. The Broadcaster Application might request to download ad content via broadband before the time of an ad replacement to avoid playback problems that might occur due to network congestion if the ad were to be streamed in real time.

The Receiver's response to the Cache Request API indicates whether or not the indicated files are already present in the Application Context Cache. Thus, the API may also be used to check whether or not the one or more indicated files are present in the Application Context Cache. The status check function works for files that might have arrived by either the broadcast or the broadband delivery path.

As stated in Section 6.2, storage capability and management of the Application Context Cache are Receiver-specific, so that files requested via this API might or might not be stored, depending on the status of the Application Context Cache. However, the Broadcaster Application can use the Query Cache Usage API defined in Section 9.5 to check how much storage quota of Application Context Cache is assigned for the Application Context ID. The Mark Unused API defined in Section 9.8 can be used to indicate to the Application Context Cache system that cached file(s) are unused. If the currently executing Broadcaster Application is terminated, the Receiver may cancel all in-progress file retrieval processes and release all cached files requested by this API.

Note that the method name, `"org.atsc.CacheRequest"`, starts with a capital "C" inconsistent with the method naming in other parts of this standard. The reader is cautioned to use the method name verbatim to avoid issues.

The Cache Request Request semantics are defined in Table 9.49 and the syntax shall be as defined in the schema file `org.atsc.CacheRequest-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.49 Cache Request Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.CacheRequest"
`sourceUrl`	0..1	string (uri)	The base URL from which files are to be retrieved
`targetUrl`	0..1	string (uri-reference)	The target URL where files are to be placed in the Application Context Cache
`URLs`	1		
items	0..N	string (uri-reference)	Relative URLs of files to either be retrieved or to provide status for (see description)

`sourceURL` – When `sourceURL` is present, this API requests the Receiver to retrieve the files referenced in the provided `URLs` array, where `sourceURL` is the base URL of each file. When `sourceURL` is present, it shall include the https protocol identifier. When `sourceURL` is absent, the API shall indicate a request for the Receiver to return information about the presence or absence of the identified files within the Application Context Cache.

`targetURL` – This relative URL shall indicate the location within the Application Context Cache relative to its base where the files are to be placed. When `sourceURL` is not present, the `targetURL` shall indicate the location within the Application Context Cache relative to its base where the Receiver should look for the files given in the `URLs` array and reply with an indication of whether or not all the files are present. When `sourceURL` is present and `targetURL` is not present, the files shall be stored under each URL relative to the root of the Application Context Cache.

`URLs` – When `sourceURL` is present, the `URLs` array shall represent an array of one or more strings which contain relative URLs of files to be retrieved and stored in the Application Context Cache. Each URL string in `URLs` shall be a relative URL. The effective URL for retrieval of the file from the broadband server shall be a concatenation of the `sourceURL` and the URL string of the file. When `sourceURL` is not present, each URL string shall refer to a file that may be present in the Application Context Cache as the concatenation of the `targetURL` and the

URL of the file, and the response to the API shall indicate whether or not all referenced files are present in the cache.

The Cache Request Response semantics are defined in Table 9.50 and the syntax shall be as defined in the schema file `org.atsc.query.CacheRequest-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.50 Cache Request Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`cached`	1	boolean	Indicates whether or not the requested files have been cached
`error`	oneOf X		See Section 8.3.3

`cached` – This required Boolean result shall indicate, when "true" that all the files referenced in the URLs are present in the Application Context Cache and that none are expired. When "false", `cached` shall indicate that one or more files are not present or are expired.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – The requested content could not be found.
- -5 – No broadband connection is available to honor the request.
- -15 – The URL format specified in sourceURL or targetURL of the request is illegal.
- -16 – The URL format specified in one or more URLs of the request is illegal.

For example, the Broadcaster Application may wish to request the download of three PNG files from a broadband server at `https://foo.com/service1` and to store these files in the Application Context Cache in specified subdirectories at or below an `images/` subdirectory. The source and destinations for each of these three files are as follows:

1. File 1:
   - Source: `https://foo.com/service1/A/big-image1.png`
   - Target location in App Context Cache: `images/A/big-image1.png`
2. File 2:
   - Source: `https://foo.com/service1/B/big-image2.png`
   - Target location in App Context Cache: `images/B/big-image2.png`
3. File 3:
   - Source: `https://foo.com/service1/C/big-image3.png`
   - Target location in App Context Cache: `images/C/big-image3.png`

To accomplish the download, the Broadcaster Application could issue the following API:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.CacheRequest",
    "params": {
      "sourceURL": "https://foo.com/service1/",
      "targetURL": "images/",
      "URLs":["A/big-image1.png","B/big-image2.png","C/big-image3.png"]
    },
    "id": 37
  }
```

In this example, the first PNG file is fetched using the URL `https://foo.com/service1/A/big-image1.png` and placed into the Application Context Cache at a subdirectory `images/A/big-image1.png`.

Upon successfully beginning the retrieval process, if the files had not been retrieved previously, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"cached": false},
    "id": 37
  }
```

Note that `cached` is "false", indicating that these files are not already present. If all the files had already been present in the Application Context Cache and none had expired, `cached` would have returned "true"; otherwise, the Receiver would begin to re-download the files.

If the Broadcaster Application wishes later to check to see whether or not the first two of these files have been successfully downloaded, it could issue the following API to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.CacheRequest",
    "params" {
      "targetURL": "images/",
      "URLs":["A/big-image1.png", "B/big-image2.png"]
    },
    "id": 38
  }
```

If both of the indicated files are present and not expired, the Receiver may respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"cached": true},
    "id": 38
}
```

9.4.2 Cache Request DASH API

The Cache Request DASH API may be used by the currently executing Broadcaster Application to indicate to the Receiver that certain files should be retrieved via broadband and stored in the Application Context Cache. Instead of listing each URL individually, using this API, files are

specified either in an MPEG DASH `Period` XML fragment or in a complete DASH MPD. If a complete DASH MPD is specified, the MPD file and the MPEG DASH segments specified in the MPD file shall be retrieved via broadband and stored. The URL of each MPEG DASH segment file shall be generated according to the MPEG DASH specification [29]. In response to the XLink Resolution Notification API (9.15.1), the Broadcaster Application can provide the same DASH `Period` XML fragment.

The Cache Request DASH API may also be used to check whether or not the files indicated in the DASH `Period` or MPD are present in the Application Context Cache and not expired. The status check function works for files that might have arrived by either the broadcast or the broadband delivery path.

This API does not wait for resources to be cached. The Broadcaster Application is expected to repeatedly call this API until `cached = "true"`. When the Receiver determines that one or more resources cannot be cached (e.g., Internet lost, HTTPS returns 404, etc.) it returns error -4.

Note that the method name, `"org.atsc.CacheRequestDASH"`, starts with a capital "C" inconsistent with the method naming in other parts of this standard. The reader is cautioned to use the method name verbatim to avoid issues.

The Cache Request DASH Request semantics are defined in Table 9.51 and the syntax shall be as defined in the schema file `org.atsc.CacheRequestDASH-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.51 Cache Request DASH Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.CacheRequestDASH"
`object A`	oneOf X		Used to specify a Period request
`sourceUrl`	0..1	string (uri)	The base URL from which `Period`-referenced segment files will be retrieved
`targetUrl`	1	string (uri-reference)	The target URL where files are to be placed in the Application Context Cache
`Period`	1	string (xml)	A DASH `Period` XML fragment whose referenced segments are requested to be cached
`object B`	oneOf X		Used to specify an MPD request
`sourceUrl`	0..1	string (uri)	The base URL from which MPD-referenced segment files will be retrieved
`targetUrl`	0..1	string (uri-reference)	The target URL where files are to be placed in the Application Context Cache
`mpdFileName`	1	string	A DASH MPD file whose referenced segments are requested to be cached

When the API is used with a DASH `Period`:

`sourceURL` – When `sourceURL` is present, this API requests the Receiver to retrieve the media files referenced in the provided DASH `Period` XML fragment, where `sourceURL` is the base URL of the files specified in the URLs in the `Period`. When `sourceURL` is present, it shall include the https protocol identifier. When `sourceURL` is absent, the API shall indicate a request for the Receiver to return information about the presence or absence of the identified files within the Application Context Cache.

`targetURL` – This relative URL shall indicate the location within the Application Context Cache relative to its base where the files are to be placed. When `sourceURL` is not present, the `targetURL` shall indicate the location within the Application Context Cache relative to its base where the Receiver should look for the files referenced by the `Period` and reply with an indication of whether or not all the files are present and not expired. When `sourceURL` is present and `targetURL` is not present, the files shall be stored under each URL relative to the root of the Application Context Cache.

`Period` – The `Period` shall represent an XML fragnent defined as a `Period` of MPEG DASH compliant with A/331 [3]. Each Media Segment and Initialization Segment URL is constructed using the processing rules of MPEG DASH [29] subclause 5.6. The `Period` shall use only relative URL references. The **`Period`**`@duration` attribute shall be present. When `sourceURL` is included, the URLs in the `Period` shall resolve to media files present on the referenced broadband server.

When the API is used with a DASH MPD:

`sourceURL` – This API requests the Receiver to retrieve the media files referenced in the DASH MPD identified by `mpdFileName`, where `sourceURL` is the URL of the broadband server from which the MPD may be retrieved. When `sourceURL` is present, it shall include the `https` protocol identifier. When `sourceURL` is absent, the API shall indicate a request for the Receiver to return information about the availability of the files identified by the referenced MPD within the Application Context Cache. When `sourceURL` is absent, the response to the request shall indicate `"cached":false` if the MPD itself is not present in the Application Context Cache, or if any of the files it references are not present or are expired.

`targetURL` – When `sourceURL` is present, the API requests the Receiver to retrieve and place the files associated with the indicated MPD, and the MPD itself, into the Application Context Cache. In that case, the `targetURL` shall indicate the location within the Application Context Cache relative to its base where the files are to be placed. The Receiver is expected to also retrieve the MPD and place it at the location in the Application Context Cache given by `targetURL`. When `sourceURL` is not present, the `targetURL` shall indicate the location within the Application Context Cache relative to its base where the Receiver should look for the MPD and the files referenced by the MPD and reply with an indication of whether or not the MPD and all the files it references are present and not expired. When `sourceURL` is present and `targetURL` is not present, the files shall be stored under each URL relative to the root of the Application Context Cache.

`mpdFileName` – The required `mpdFileName` shall represent the filename of an MPEG DASH MPD that is compliant with A/331[3]. The URL of each Media Segment and Initialization Segment referenced in the indicated MPD is constructed using the processing rules of MPEG DASH [29] subclause 5.6. The referenced MPD shall include only relative URLs. When `sourceURL` is included, the URLs in the MPD shall resolve to media files present on the referenced broadband server, and the MPD itself shall be present at the server location indicated in `sourceURL` with the filename given in `mpdFileName`.

According to MPEG DASH [29] subclause 5.6.4, "URLs at each level of the MPD are resolved according to RFC 3986 with respect to the **`BaseURL`** element specified at that level of the document

or the level above in the case of resolving base URLs themselves (the document 'base URI' as defined in RFC 3986 [25] Section 5.1 is considered to be the level above the MPD level)." For this API, the `sourceURL` is the document "base URI" on the broadband server, and the `targetURL` is the document "base URI" in the Application Context Cache.

The Cache Request DASH Response semantics are defined in Table 9.52 and the syntax shall be as defined in the schema file `org.atsc.query.CacheRequestDASH-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.52 Cache Request DASH Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`cached`	1	boolean	Indicates whether or not the requested files have been cached
`error`	oneOf X		See Section 8.3.3

`cached` – This Boolean result shall indicate, when "true", that all the files referenced in the `Period` or MPD are present in the Application Context Cache at the indicated location and that none are expired. When "false", `cached` shall indicate that one or more files are expired or not present. When `sourceURL` is present in the request, a result of "true" shall be returned in the case that the indicated files are already present in the Application Context Cache, and none are expired.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4: Content not found.
- -5: No broadband connection is available.
- -11: The indicated MPD cannot be accessed.
- -15: The URL format specified in `sourceURL` or `targetURL` of the request is illegal.
- -17: The format of the MPEG DASH fragment specified in the Period is illegal.
- -18: The referenced MPD segment file cannot be found.

For example, if the Broadcaster Application wishes to request from a broadband server the fetching of MPEG DASH media segment files corresponding to one `Period` from a broadband server at `https://wxyz.com/svc4.4/content/`, and place them into the Application Context Cache at `advertising1/` it could issue the following API:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.CacheRequestDASH",
    "params": {
       "sourceURL": "https://wxyz.com/svc4.4/content/",
       "targetURL": "advertising1/",
       "Period": "<Period start='PT9H' duration='PT30S'>
         <AdaptationSet mimeType='video/mp4'/>
         <SegmentTemplate timescale='9000' media='video/xbc$Number$.mp4v'
         duration='90000' startNumber='32401' /><Representation id='v2'
         width='1920' height='1080'/></Period>"
    },
    "id": 38
  }
```

The resulting video Media Segment files would be retrieved and stored in the Application Context Cache in the `advertising1/video/` subdirectory. Upon successfully beginning the retrieval process, if the files had not been retrieved previously the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"cached": false},
    "id": 38
  }
```

The `cached` value of "false" in the response indicates that the requested files are not all already present in the cache. If all the files had already been present in the Application Context Cache and none were expired, `cached` would have returned "true", otherwise the Receiver would begin to re-download the files.

If the Broadcaster Application wishes later to check to see whether or not the files associated with the indicated DASH `Period` have been successfully downloaded, it could issue the following API to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.CacheRequestDASH",
    "params": {
       "targetURL": "advertising1/",
       "Period": "<Period start='PT9H' duration='PT30S'>
          <AdaptationSet mimeType='video/mp4'/>
          <SegmentTemplate timescale='9000' media='video/xbc$Number$.mp4v'
          duration='90000' startNumber='32401' />
          <Representation id='v2' width='1920' height='1080'/></Period>"
    },
    "id": 37
  }
```

If all of the indicated Media Segment files are present, including Initialization Segments, the Receiver may respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"cached": true},
    "id": 37
}
```

If any of the indicated Media Segment files or Initialization Segments are missing, the Receiver may respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"cached": false},
    "id": 37
}
```

9.5 Query Cache Usage API

If the Broadcaster Application wishes to know the total quota size of the cache assigned to the Application Context ID with which it is associated and the total current usage of the cache in the Application Context ID hierarchy, the Query Cache Usage API can be used.

The Query Cache Usage Request semantics are defined in Table 9.53 and the syntax shall be as defined in the schema file `org.atsc.query.cacheUsage-request.json`.

Table 9.53 Query Cache Usage Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.cacheUsage"

The Query Cache Usage Response semantics are defined in Table 9.54 and the syntax shall be as defined in the schema file `org.atsc.query.cacheUsage-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.54 Query Cache Usage Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`usageSize`	1	integer	The total number of bytes used in the current Application Context Cache
`quotaSize`	1	integer	The total number of bytes allocated to the current Application Context Cache
`error`	oneOf X		See Section 8.3.3

`usageSize` – The total usage byte size of the cache associated with the Application Context ID of the Broadcaster Application.

`quotaSize` – The total size in bytes of the quota allocated for the Application Context ID of the Broadcaster Application.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Broadcaster Application wishes to query the status of cache usage, it could do as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.cacheUsage",
     "id": 39
}
```

If the usage size of cache equals 8,475,337 bytes and the quota size available to the Application Context ID equals 209,715,200 bytes, the Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "usageSize": 8475337,
        "quotaSize": 209715200
    },
    "id": 39
}
```

9.6 Event Stream APIs

Events intended for Broadcast Applications can be encountered in broadcast media, either as Event Message (`'emsg'` or `'evti'`) Boxes in-band with the media, or as static `EventStream` elements at the `Period` level in a DASH MPD, obtained via broadband from a signaling server or content recovery server, or detected in decoded video content from a video watermark. These Events may initiate interactive actions on the part of a Broadcast Application, or they may indicate that new versions of files are being delivered, or various other things. Specification of the delivery of events for ATSC 3.0 applications and synchronization of these application events with underlying content can be found in the A/337 standard [6].

In the case of AMP media playback, parsing and processing of Events is expected to be performed by the Broadcaster Application. In the case of RMP media playback, 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

9.6.1 Event Stream Subscribe API

A Broadcaster Application that is currently subscribed to Event Stream notifications are expected to be notified when certain Event Stream events are encountered during RMP playback in the MPD or the Media Segments. For MPEG DASH, the Event Message Box (`'emsg'`) box contains in-band events, and the MPD may include static events in an `EventStream` element at the `Period` level. Events in MMT-based Services may be carried in `'evti'` boxes in MPUs [6]. 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 Request semantics are defined in Table 9.55 and the syntax shall be as defined in the schema file org.atsc.eventStream.subscribe-request.json. Additional semantic definitions of parameters follow the table.

Table 9.55 Event Stream Subscribe Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.eventStream.subscribe"
schemeIdUri	1	string (uri)	The event stream scheme ID that is requested to be sent to the Broadcaster Application
value	0..1	string	A specific event to be detected
dispatchMode	0..1	string	Sets the relative timing of the event notification

schemeIdUri – The schemeIdUri URI string associated with the Event Stream event of interest to the Broadcaster Application. The syntax of the schemeIdUri is expected to comply with the syntax of **AEI.EventStream**@schemeIdUri as defined in [6].

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

dispatchMode – An optional string specifying when an event is set. The values are as follows:

- onReceive – (default value). If set to this value, or if the dispatchMode property is not present, events shall be dispatched as soon as practical after they are received (and not when they are due to start).
- onStart – If set to this value, events shall be dispatched at the moment when the event is due to start.

For more details regarding this operation, see DASH-IF Events Reference Model in [42].The Event Stream Subscribe Response semantics are defined in Table 9.56 and the syntax shall be as defined in the schema file org.atsc.eventStream.subscribe-response.json. Additional semantic definitions of parameters follow the table.

Table 9.56 Event Stream Subscribe Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Empty object on successful subscription. The error structure is returned if unsuccessful.
error	oneOf X		See Section 8.3.3

result – On successful subscription, the result structure shall contain no elements. In JSON, this is represented as "result": {}.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -16 – Indicates that the schemeIdUri property contains a malformed URI.

For example, if the Broadcaster Application wishes to register for Event Stream events associated with schemeIdUri "urn:uuid:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE", it could subscribe as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.eventStream.subscribe",
    "params": {"schemeIdUri": "urn:uuid:1AD2F3EF-87C8-46B4-BD1D-
94C174C278EE"},
    "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:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE"` to the Broadcaster Application using the Event Stream Event API defined in Section 9.6.3 below.

If the Broadcaster 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:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE",
        "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:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE"` and `value` = "17" to the Broadcaster Application using the notification API defined in Section 9.6.3 below. The Broadcaster 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 Broadcaster Application may subscribe to multiple different Event Stream events (with different `schemeIdUri` values, or different `schemeIdUri`/`value` combinations).

Once subscribed, the Broadcaster Application may unsubscribe using the API described in Section 9.6.2.

9.6.2 Event Stream Unsubscribe API

If a Broadcaster Application has subscribed to an Event Stream using the Event Stream Subscribe API defined in Section 9.6.1, it can use the Event Stream Unsubscribe API defined here to request that the Receiver discontinue notifications pertaining to the identified event.

The Event Stream Unsubscribe Request semantics are defined in Table 9.57 and the syntax shall be as defined in the schema file `org.atsc.eventStream.unsubscribe-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.57 Event Stream Unsubscribe Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.eventStream.unsubscribe"
`schemeIdUri`	1	string (uri)	The event stream scheme ID that is requested to be sent to the Broadcaster Application
`value`	0..1	string	A specific event to be detected

`schemeIdUri` – The `schemeIdUri` URI string associated with the Event Stream event for which the Broadcaster Application would like to remove the subscription. The syntax of the `schemeIdUri` shall comply with the syntax of **`AEI.EventStream`**`@schemeIdUri` as defined in [6].

`value` – An optional string used to identify a particular Event Stream event from which to remove the subscription.

The Event Stream Unsubscribe Response semantics are defined in Table 9.58 and the syntax shall be as defined in the schema file `org.atsc.eventStream.unsubscribe-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.58 Event Stream Unsubscribe Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful subscription removal. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – On successful unsubscribe request, the `result` structure is expected to contain no elements. In JSON, this is represented as `"result": {}`.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -16 – Indicates that the `schemeIdUri` property contains a malformed URI.
- -24: The Broadcaster Application was not subscribed to any of the requested notifications

For example, if the Broadcaster Application wishes to unsubscribe to all Event Stream events associated with `schemeIdUri` "`urn:uuid:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE`", 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:1AD2F3EF-87C8-46B4-BD1D-
94C174C278EE"},
    "id": 26
}
```

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

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

If the Broadcaster 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:1AD2F3EF-87C8-46B4-BD1D-94C174C278EE",
        "value": "48"
    },
    "id": 29
}
```

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

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

9.6.3 Event Stream Event API

The Event Stream Event is expected to be issued by the Receiver to the currently executing Broadcaster Application during RMP playback 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 semantics are defined in Table 9.59 and the syntax shall be as defined in the schema file `org.atsc.eventStream.event-notification.json`. Additional semantic definitions of parameters follow the table.

Table 9.59 Event Stream Event Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.eventStream.event"
schemeIdUri	1	string (uri)	Identifies the source event stream from which this event originated
value	0..1	string	Defined by scheme
currentTime	0..1	number (>=0)	The current offset on the media timeline in floating point seconds
eventTime	1	number (>=0)	The media presentation time when this event occurred in floating point seconds
duration	0..1	number (>=0)	The duration of the event in floating point seconds
id	0..1	integer (0 … 4294967295)	The relative ID of this event
data	0..1	string	The data set from the event, optionally encoded with the contentEncoding method
contentEncoding	0..1	string	Identifies an encoding applied to the data property

schemeIdUri – A required string identifying the Event Stream with which the Event is associated. The syntax of the schemeIdUri shall comply with the syntax of **AEI.EventStream**@schemeIdUri as defined in [6].

value – An optional string with semantics as defined by the owners of the Event Stream scheme identified by schemeIdUri.

currentTime – An optional floating-point number representing the current time on the RMP media presentation timeline, expressed as an offset in seconds from the startDate (as specified in Query RMP Media Time API, Section 9.13.1).

eventTime – A required floating-point number representing the presentation time at which the event starts on the RMP media presentation timeline, expressed as an offset in seconds from the startDate (as specified in Query RMP Media Time API, Section 9.13.1).

duration – An optional floating-point number representing the duration of the event in seconds.

id – An optional number indicating the relative id of this event.

data – An optional string or object representing additional data associated with this event, with semantics as defined by the owners of the Event Stream scheme identified by schemeIdUri. The contentEncoding property indicates an encoding that has been applied to the data. When a content encoding is indicated, it is the responsibility of the Broadcaster Application to decode the data.

contentEncoding – An optional string specifying a content encoding that has been applied by the Receiver to the data. The only supported value for the contentEncoding property is "base64", indicating that the data parameter value is base64 encoded [27]. Absence of a specified encoding is indicated by absence of the contentEncoding property from the response.

The start time of the event on the RMP presentation timeline is when currentTime (as provided in Query RMP Media Time API, Section 9.13.1) is equal to eventTime of the Event.

For MPD Events [41], the Receiver shall populate the data property of the Event Stream Event response with the un-decoded value of the **Event**@messageData parameter of the MPD Event (i.e., without applying the decoding identified by **Event**@contentEncoding, if any) or, if no

Event@messageData attribute is present in the MPD Event, the un-decoded value of the Event element of the MPD Event. The Receiver shall populate the `contentEncoding` property of the response with the **Event@contentEncoding** parameter of the MPD Event, if present.

For AEI events [6], the Receiver shall populate the data property of the Event Stream Event response with the Event parameter of the AEI event and the `contentEncoding` property of the response shall be absent.

For events delivered via `'emsg'` box [6], `'evti'` box [6], or Dynamic Event Message [5] and where the received event data is not UTF-8-compliant, the Receiver shall apply base64 encoding [27] to the received event data, populate the data property of the Event Stream Event response with the encoded event data, and populate the `contentEncoding` property of the response with the value `"base64"`. Receivers shall not apply an encoding to received event data that is UTF-8 compliant. (Efficient methods for establishing the UTF-8 compliance of stream event data are widely available to Receiver manufacturers [51].)

Note that it is the responsibility of the Broadcaster Application to base64 decode the data property of the Event Stream Event response, regardless of anticipated binary or UTF-8 payload assumptions, if the Receiver indicates `"base64"` in the `contentEncoding` property.

An example Event Stream notification message that might occur if the Broadcaster 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",
        "currentTime": 1460.2,
        "eventTime": 1450.6,
        "id": 60,
        "data": "d8a0c98fs08-d9df0809s"
    }
}
```

Note in this example that if the Broadcaster 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 Broadcaster Application.

9.7 Request Receiver Actions

9.7.1 Acquire Service API

The current service may be changed by two entities, the Broadcaster Application via request to the Receiver, or the user via the Receiver directly. This may be within the same or different RF channel. Depending on the information sent in the Broadcaster Application signaling, the Receiver performs the actions described in Section 6.3, Broadcaster Application Lifecycle.

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. The Receiver processes the request and if it can, it changes the service selection.

The Acquire Service Request semantics are defined in Table 9.60 and the syntax shall be as defined in the schema file org.atsc.acquire.service-request.json. Additional semantic definitions of parameters follow the table.

Table 9.60 Acquire Service Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.acquire.service"
svcToAcquire	1	string (uri)	The globalServiceID of the service to be acquired

svcToAcquire – This required string shall correspond to the globalServiceID (as defined in SLT.Service@globalServiceID; see A/331 [3] Section 6.3) of the service to acquire.

The Acquire Service Response semantics are defined in Table 9.61 and the syntax shall be as defined in the schema file org.atsc.acquire.service-response.json. Additional semantic definitions of parameters follow the table.

Table 9.61 Acquire Service Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Empty object on successful acquisition. An error structure is returned if unsuccessful.
error	oneOf X		See Section 8.3.3

result – If the service acquisition is successful, the Receiver shall respond with a JSON-RPC response object with an empty, "{}", result object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -6 – Service not found
- -7 – Service not authorized

For example, if the Broadcaster Application requests access to a service represented by globalServiceID "https://doi.org/10.5239/8A23-2B0B", it can issue this request to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.acquire.service",
    "params": {"svcToAcquire": "https://doi.org/10.5239/8A23-2B0B"},
    "id": 59
}
```

The Receiver would respond, if acquisition were successful with:

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

If `globalServiceID` `"https://doi.org/10.5239/8A23-2B0B"` is unknown to the Receiver, the response would be:

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

9.7.2 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 Request semantics are defined in Table 9.62 and the syntax shall be as defined in the schema file `org.atsc.scale-position-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.62 Video Scaling and Positioning Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.scale-position"
`scaleFactor`	1	number (10.0 ... 100.0)	The percentage to scale the video from 10.0% to 100.0%
`xPos`	1	number (0.0 … 100.0-scaleFactor)	The X-axis location to position the video as a percentage of full screen width
`yPos`	1	number (0.0 ... 100.0-scaleFactor)	The Y-axis location to position the video as a percentage of full screen height

`scaleFactor` – This required number value in the range 10.0 to 100.0 shall represent the video scaling parameter, where 100.0 represents full-screen (no scaling).

`xPos` – This required number value in the range 0.0 to 100.0-`scaleFactor` 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 0.0 indicates the left side of the video window is aligned with the left side of the display window. A value of 50.0 indicates the left side of the video window is aligned with the vertical centerline of the display window, etc.

`yPos` – This required number value in the range 0.0 to 100.0-`scaleFactor` 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 0.0 indicates the top of the video window is aligned with the

top of the display window. A value of 50.0 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.

With a successful Video Scaling and Positioning Request, it is expected that scaling using the `scaleFactor` value is applied before positioning using the `xPos` and `yPos` values to perform the transform of the video window. With any additional successful Video Scaling and Positioning Request, it is expected that the transform is applied to a reset full-screen video window and not the already transformed video window from a previous successful Video Scaling and Positioning Request.

The Video Scaling and Positioning Response semantics are defined in Table 9.63 and the syntax shall be as defined in the schema file `org.atsc.scale-position-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.63 Video Scaling and Positioning Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful video scaling and positioning. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3 as extended below
`code`	1	integer	The error code indicating what problem occurred
`message`	1	string	A concise message describing the error
`data`	0..1	object	
`minimumScaleFactor`	0..1	number (10.0 ... 100.0)	Minimum scaleFactor supported by the Receiver.

`result` – If the video scaling and positioning request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -8 – Failed to scale/position the video.

In this case, the Receiver may optionally include the `minimumScaleFactor` as a property of the error data object.

`minimumScaleFactor` – Provides the minimum `scaleFactor` supported by the Receiver. See the `scaleFactor` semantics defined above.

For example, if the Broadcaster Application wished to scale the displayed video to 25% of full screen, and position the left edge of the display horizontally at 10% of the screen width and the top edge of the display vertically at 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": 25.0,
        "xPos": 10.0,
        "yPos": 15.0
    },
    "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",
        "data":{
            "mimimumScaleFactor": 30.0
        }
    },
    "id": 589
}
```

9.7.3 Set RMP URL API

The Broadcaster Application may choose to use the Receiver Media Player (RMP) to play video content originating 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 Broadcaster 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 content presented via the specified MPD is considered to be a part of the currently selected Service. The effects of changing this URL are temporary and if the Service is re-selected (e.g., by the Broadcaster Application via the Acquire Service API), then the RMP is expected to process the `endOperation` function. Note that the Broadcaster Application may use the stopRMP operation to stop the current RMP playback, but selecting a service overrides this operation.

The Broadcaster Application may request that the RMP synchronize the requested operations to a future time on the current presentation timeline. This explicit synchronization is needed in the

scenario where the currently playing content is being delivered by an alternate transport such as what might be found in redistribution. The RMP is not expected to be capable of queuing more than one such pending request at a time. Synchronization is indicated through use of the `rmpSyncTime` parameter as specified in the following paragraphs.

If the Set RMP URL API is called by a Broadcaster Application and a `startRMP` operation is specified, then:

- if no `rmpSyncTime` value is specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and immediately begin playback of the MPD given in the current request;
- if the RMP is currently playing the content specified in service-level signaling and if an `rmpSyncTime` value is specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and to begin the playback of the MPD given in the current request when the presentation time specified by `rmpSyncTime` is reached;
- if the RMP is currently playing an MPD specified in a prior Set RMP URL request and if `rmpSyncTime` has the value -1.0 specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and to begin the playback of the MPD given in the current request when the end of the presentation currently being played by the RMP is reached;
- Otherwise, the Receiver is expected to ignore the current request and continue to play the current content.

If the Set RMP URL API is called by a Broadcaster Application and a `stopRMP` operation is specified, then:

- if the RMP is currently playing the content specified in service-level signaling or an MPD specified in a prior Set RMP URL request and if no `rmpSyncTime` value is specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and immediately stop the presentation of the RMP;
- if the RMP is currently playing the content specified in service-level signaling or an MPD specified in a prior Set RMP URL request and if an `rmpSyncTime` value is specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and to continue playback of the current content until the presentation time indicated by `rmpSyncTime` is reached, at which time it is expected to stop the presentation of the RMP; and
- if the RMP playback is currently stopped, then the Receiver is expected to ignore the current request.

If the Set RMP URL API is called by a Broadcaster Application and a `resumeService` operation is specified, then:

- if the RMP is currently either playing an MPD specified in a prior Set RMP URL request or stopped by a `stopRmp` operation of a prior Set RMP URL request and if no `rmpSyncTime` is specified in the current request, then the Receiver is expected to cancel any pending Set RMP URL request and process the `endOperation` function;
- if the RMP is currently playing an MPD specified in a prior Set RMP URL request and if an `rmpSyncTime` is specified in the current request, then the RMP is expected to immediately cancel any pending Set RMP URL request and to continue playback of the

currently playing MPD until the presentation time indicated by `rmpSyncTime` is reached, at which time it is expected to process the `endOperation` function;

- Otherwise, the Receiver is expected to ignore the current request and continue to play the current content.

If the Receiver determines that it is unable to perform the action as requested, it is expected to return an error code and is not expected to perform the requested action.

At the time the RMP begins playback of the MPD given in a Set RMP URL request, the Broadcaster Application can receive a notification via the Signaling Data Change Notification API (Section 9.3.11). In any case, whenever an MPD change or update causes a discontinuity in the presentation timeline, the RMP is expected to cancel any pending Set RMP URL requests.

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 A/331 [3].

The URL may include an MPD Anchor identifying the entry point on the media presentation timeline (e.g., an offset from the start of the MPD, an offset from the start of a named period, a UTC time, or the "live edge") at which the RMP should begin playback. MPD Anchor shall be as defined in MPEG DASH [29]. This allows flexibility for many use cases including bookmarking. If the playback position indicated by a specified MPD Anchor is not available to the RMP, the RMP is not expected to play the MPD at the given URL and an error code is expected to be returned.

The Set RMP URL Request semantics are defined in Table 9.64 and the syntax shall be as defined in the schema file `org.atsc.setRMPURL-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.64 Set RMP URL Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.setRMPURL"
`operation`	0..1	enum	"startRmp", "stopRmp", "pauseRmp", "resumeRmp", "fastForwardRmp", "rewindRmp", "skipForwardRmp", "skipBackwardsRmp", "resumeService"
`rmpurl`	0..1	string (uri)	Provides the URI of the MPD to be played by the RMP if "operation" = "startRmp"
`rmpSyncTime`	0..1	number	Indicates the time offset when the operation specified should occur
`endOperation`	0..1	enum	"stopRmp", "pauseRmp", "resumeService"

`operation` – This optional string shall define the operation to be performed by the RMP. When `operation` is absent, the Receiver shall cancel any pending Set RMP URL API request. The meaning of the enumerated values shall be defined as follows:

`"startRmp"` indicates that the RMP shall start playing the URI provided by the `rmpurl` property as described in the property description below.

`"stopRmp"` indicates that the RMP shall cease playback. For this operation, the `rmpurl` property is not required and shall be ignored if present.

"pauseRmp" indicates that the RMP shall suspend playback, freeze-frame, and mark the current position in the content.

"resumeRmp" indicates that the RMP shall continue playing from a previous pauseRmp operation.

"fastForwardRmp" indicates that the RMP shall speed up playback. The initial speed and sequential calls for this operation are Receiver-dependent.

"rewindRmp" indicates that the RMP shall playback in reverse. The initial speed and sequential calls for this operation are Receiver-dependent.

"skipForwardRmp" indicates that the RMP shall skip forward and resume playback. The time skipped is Receiver-dependent.

"skipBackwardsRmp" indicates that the RMP shall skip backward and resume playback. The time skipped is Receiver-dependent.

"resumeService" indicates that the RMP shall resume normal playback of the current Service. For this operation, the "rmpurl" property is not required and shall be ignored if present.

rmpurl – When the operation value is set to startRmp, this string shall be specified and provide a fully qualified URI referencing an MPD to be played by the RMP, whether referencing an MPD over broadband or in the Application Context Cache. The URI shall be accessible to the Receiver. The appropriate error code (see below) shall be returned if the URI cannot be accessed. Note that for an MPD in the Application Context Cache, the full URI can be constructed using the Base URI provided using the Query Receiver Web Server URI API as described in Section 9.2.7.

rmpSyncTime – This optional floating-point number indicates a future time (i.e., later than currentTime) on the media presentation timeline of the presentation currently being played by the RMP (in seconds, relative to the media presentation time given by startDate as specified in the Query RMP Media Time API, Section 9.13.1) at which the action specified by operation should be performed. If rmpSyncTime is not specified, the action indicated by operation should begin playing immediately. If rmpSyncTime has the value -1.0, the action indicated by operation should be performed when the end of the presentation currently being played by the RMP is reached. (The end of the presentation is considered to occur when no further presentation description is indicated.)

endOperation – This optional string indicates what the RMP shall do when it reaches the end of the MPD presentation. The values shall be as defined for the parameter, operation. If this parameter is absent the default value shall be resumeService.

The Set RMP URL Response semantics are defined in Table 9.65 and the syntax shall be as defined in the schema file org.atsc.setRMPURL-response.json. Additional semantic definitions of parameters follow the table.

Table 9.65 Set RMP URL Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		An empty structure is returned on successful request otherwise the error structure is returned
error	oneOf X		See Section 8.3.3

`result` – If the set RMP URL is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -5 – No broadband connection is available.
- -11 – The indicated MPD cannot be accessed.
- -12 – The content cannot be played.
- -13 – The requested MPD Anchor cannot be reached.
- -15 – Indicates that the provided `rmpurl` property value is an illegal URL.
- -19 –The synchronization specified by `rmpSyncTime` cannot be achieved.
- -21 – Changing RMP playback from the current source is not supported.

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

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "startRmp",
               "rmpurl": "https://stream.wxyz.com/33/program.mpd"},
    "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
}
```

Furthering the example, the user interacts with a Broadcaster Application, which now wishes to display a full-screen video on demand selection screen with no video. The Broadcaster Application makes the following request to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "stopRmp"},
    "id": 113
}
```

Upon success, the RMP would cease displaying video and the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 113
}
```

Finally, the user exits from the video on demand scenario and now wants to return to watching broadcast services. The Broadcaster Application would make the following request:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "resumeService"},
    "id": 106
}
```

Upon success, the Receiver would resume normal playback operations and would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 106
}
```

As a fourth example, if the `currentTime` as provided by the Query RMP Media Time API is 1740 seconds past the `startDate`, no MPD given in a prior Set RMP URL request is being played, and the Broadcaster Application wants the RMP to begin the playback of the MPD located at `https://stream.wxyz.com/33/program.mpd` when the `currentTime` of the presentation currently being played by the RMP reaches 1800 seconds with an entry point into the specified MPD that is 5 minutes from its beginning, the application can issue a command to the Receiver as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "startRmp",
               "rmpurl": "https://stream.wxyz.com/33/program.mpd#t=5:00",
               "rmpSyncTime": 1800.00},
    "id": 107
}
```

Upon successfully scheduling the pending MPD playback, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 107
}
```

If the Receiver's RMP cannot accept the request for scheduled playback of the content (e.g., because the specified synchronization time has passed or is too soon for the RMP to prepare), the Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "error": { "code": -19,
    "message": "The synchronization specified by rmpSyncTime cannot be
achieved"},
    "id": 107
}
```

As a fifth example, if the RMP is playing an MPD per a prior Set RMP URL API request and the `currentTime` is 10 seconds past the `startDate` as provided by the Query RMP Media Time API, and the Broadcast Application wants the RMP to resume the playback of the content specified in service-level signaling when the `currentTime` of the presentation currently being played reaches 60 seconds, it can issue a request to the Receiver as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "resumeService",
               "rmpSyncTime": 60.00},
    "id": 108
}
```

Upon successfully scheduling the requested playback of the content specified in the service-level signaling, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 108
}
```

As a sixth example, if the RMP is playing content specified in the service-level signaling (or an MPD specified in a prior Set RMP URL API), the `currentTime` is 10 seconds past the `startDate` as provided by the Query RMP Media Time API, then the Broadcaster Application can request the Receiver to stop the current playback when the `currentTime` reaches 30 seconds with the following request:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setRMPURL",
    "params": {"operation": "stopRmp",
               "rmpSyncTime": 30.00},
    "id": 109
}
```

Upon successfully scheduling the requested stop of the current playback, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 109
}
```

9.7.4 Audio Volume API

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 HTML5 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 HTML5 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. Note that the API changes *only* the RMP volume ("V1"). The overall Receiver Volume Control is not manageable from the Broadcaster Application and control of this audio volume is not in the scope of the present document.

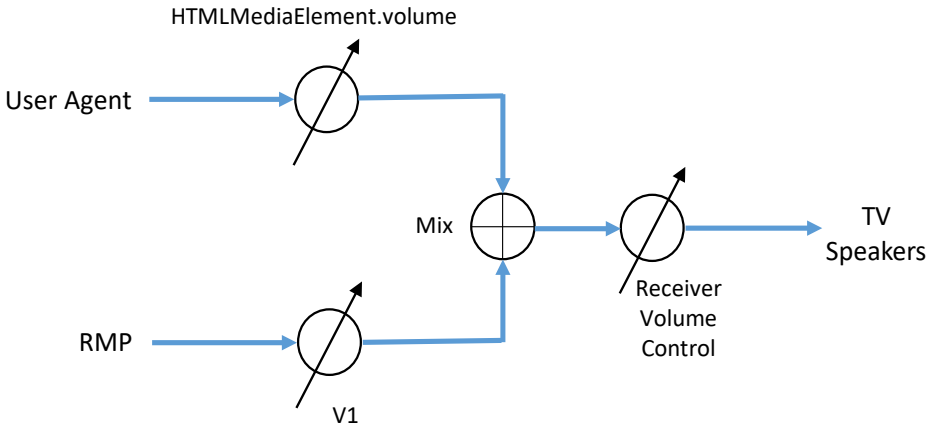


Figure 9.1 RMP audio volume.

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 Request semantics are defined in Table 9.66 and the syntax shall be as defined in the schema file `org.atsc.audioVolume-request.json`.

Table 9.66 Audio Volume Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.audioVolume"
`audioVolume`	0..1	number (0.0 … 1.0)	If present, the requested audio volume in the range 0.0 (muted) to 1.0 (full volume)

`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 number shall be from 0.0 (minimum or muted) to 1.0 (full volume). The encoding is the same as the `.volume` property of the HTML5 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.

The Audio Volume Response semantics are defined in Table 9.67 and the syntax shall be as defined in the schema file `org.atsc.audioVolume-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.67 Audio Volume Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`audioVolume`	1	number (0.0 … 1.0)	The current audio volume in the range 0.0 (muted) to 1.0 (full volume)
`error`	oneOf X		See Section 8.3.3

`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.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Broadcaster 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 Receiver might respond with:

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

9.7.5 Dialog Enhancement API

By default, the audio decoder in the Receiver's Receiver Media Player applies Dialog Enhancement processing as configured by the user in his or her preference settings. The Broadcaster Application may wish to provide an interface to get or set the amount of processing, or to release a setting previously made by a Broadcaster Application. The Dialog Enhancement API may be used for this case.

It is anticipated that once the user changes his or her desired Dialog Enhancement processing level in the Receiver preferences, these changes should be applied immediately. Therefore, the Broadcaster Application-initiated preference setting is expected to be superseded, and the Receiver is expected to use the gain value from the preference settings again. Simultaneously, the Broadcaster Application may get informed about this change through the Dialog Enhancement Preference Change Notification API and consequently can act accordingly upon this event.

The Receiver processes the request and if it can, changes the amount of processing. The settings in the user's preferences are not expected to be changed.

The Dialog Enhancement Request semantics defined in Table 9.68 and the syntax shall be as defined in the schema file `org.atsc.dialogEnhancement-request.json`.

Table 9.68 Dialog Enhancement Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.dialogEnhancement"
`dialogEnhancementGain`	0 or oneOf X	integer	If present, the requested dialog enhancement gain in dB
`dialogEnhancementReset`	0 or oneOf X	boolean	If present and "`true`", resets the dialog enhancement value set by the Broadcast Application

`dialogEnhancementGain` – This optional integer number specifies, when present, the gain value in dB of the Dialog Enhancement processing to be applied in the audio decoder. A value of 0 shall disable the Dialog Enhancement processing. A value of `dialogEnhancementGain` that is outside the allowed value range of Dialog Enhancement processing as indicated by metadata in the currently decoded audio stream shall be restricted by the Receiver.

If neither this value nor the `dialogEnhancementReset` is specified in the request, the amount of processing is not changed by this request. This can be used to determine the current amount and limits of processing applied by the audio decoder since these values are returned in the response.

`dialogEnhancementReset` – If set to `"true"`, this optional Boolean value shall release the Broadcaster Application-controlled dialog enhancement processing. The Receiver is expected to revert to dialog enhancement processing as configured by the user in the preference settings. If absent or set to `"false"`, the state and amount of dialog enhancement processing remains unchanged.
If neither this value nor the `dialogEnhancementGain` is specified in the request, the amount of processing is not changed by this request. This can be used to determine the current amount and limits of processing applied by the audio decoder since these values are returned in the response.

Note: The user's preferences may be obtained by using the Query Dialog Enhancement Preferences API specified in Section 9.2.11.

The Dialog Enhancement Response semantics are defined in Table 9.69 and the syntax shall be as defined in the schema file `org.atsc.dialogEnhancement-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.69 Dialog Enhancement Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`dialogEnhancementGain`	1	integer	The current dialog enhancement gain value in dB
`dialogEnhancementLimit`	1		Provides the current-audio-stream-signaled dialog enhancement limits
`max`	1	integer	Upper limit of the allowed Dialog Enhancement processing gain value in dB
`min`	1	integer	Lower limit of the allowed Dialog Enhancement processing gain value in dB
`error`	oneOf X		See Section 8.3.3

`dialogEnhancementGain` – This required integer number shall indicate the Dialog Enhancement gain value in dB as configured to be applied in the audio decoder. If the desired gain value is outside the range of allowed gain values, the currently applied gain value is clipped towards the nearest specified limit.

`dialogEnhancementLimit` – This required object supplies information on the signaled limits of Dialog Enhancement processing signaled in the currently decoded audio stream.

`max` – This required integer shall provide the currently signaled upper limit of the allowed Dialog Enhancement processing gain value in dB.

`min` – This required integer shall provide the currently signaled lower limit of the allowed Dialog Enhancement processing gain value in dB.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -22: Dialog Enhancement failed

For example, if the Broadcaster Application wishes for the Dialog Enhancer in the Receiver Media Player's audio decoder to apply processing at a gain value of 8 dB it would submit the following request:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.dialogEnhancement",
    "params": {"dialogEnhancementGain": 8},
    "id": 192
}
```

If the request is processed successfully, the Receiver would respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "dialogEnhancementGain": 8,
        "dialogEnhancementLimit": {
            "max": 12,
            "min": 0
        }
    },
    "id": 192
}
```

If the request is not successful, the Receiver may respond with error code -22:

```
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -22, "message": "Dialog Enhancement failed"},
    "id": 192
}
```

9.7.6 Launch Broadcaster Application API

This API enables the currently executing Broadcaster Application to start a new Broadcaster Application from the HELD.

The `@appId` string of the calling Broadcaster Application is included as a parameter on the entry point call. See Section 8.2.

The Launch Broadcaster Application Request semantics are defined in Table 9.70 and the syntax shall be as defined in the schema file `org.atsc.launchApp-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.70 Launch Broadcaster Application Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.launchApp"
`appId`	1	string (uri)	The `appId` of the Broadcaster Application to launch
`parameters`	0..1	string	Opaque text string from the calling Broadcaster Application to the launched Broadcaster application

`appId` – This required string is the `appId` as defined in A/331 [3] Section 7.1.8.

`parameters` – This optional string of text that is passed from the calling Broadcaster Application to the launched Broadcaster Application. The syntax and semantics are private between the two Broadcaster Applications.

Note that the `appId` and `parameters` strings are passed on via the query string defined in section 8.2.

The Launch Broadcaster Application Response semantics are defined in Table 9.71 and the syntax shall be as defined in the schema file `org.atsc.launchApp-response.json`. There is no return from this API if it is successful. If the request is unsuccessful, the error response defined below is returned.

Table 9.71 Launch Broadcaster Application Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`error`	1		See Section 8.3.3

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -23 – `appId` not found in the HELD
- -25 – `appId` found in the HELD but not available, or broadcast-only and not yet acquired
- -26 – `appId` found in the HELD, broadband-only, but no network connectivity
- -27 – Receiver does not support the required capabilities

In the following example, the Broadcaster Application launches another Broadcaster Application:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.launchApp",
    "params": {
        "appId": "pbs.org/kids/1"
    },
    "id": 42
}
```

Upon success, the Receiver is not expected to respond since the new Broadcaster Application will have been started. Upon failure, the Receiver is expected to respond with an error if possible.

9.7.7 Media Track Selection API for DASH

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 DASH Media Track Selection API may be used for these cases.

The Receiver processes the request and if it can, it changes the selection.

The DASH Media Track Selection Request semantics are defined in Table 9.72 and the syntax shall be as defined in the schema file `org.atsc.track.selection-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.72 DASH Media Track Selection Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.track.selection"
`selectionId`	1	integer	The track ID to be selected

`selectionId` – This required integer shall correspond to a value of `@id` attribute in either an `AdaptationSet` in the current Period, or alternatively, for complex audio presentations involving pre-selection, the DASH `Period.Preselection@id` value of the current Period. For unambiguous selection of one track or audio presentation, all id values within the Period should be unique.

The DASH Media Track Selection Response semantics are defined in Table 9.73 and the syntax shall be as defined in the schema file `org.atsc.track.selection-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.73 DASH Media Track Selection Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful track selection. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – If the media track selection request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -10 – The specified track cannot be selected.

For example, if the Broadcaster 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": 329
}
```

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

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

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

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

9.7.8 Graphics Display Regions API

The Broadcaster Application might need to provide the regions of the screen that would be occupied by the graphical layout that it plans to present. The Receiver might request to use such information and make adjustments to other display components that are being rendered for purposes such as mitigating potential display conflict. The graphics display regions layout and numbering for use with the Graphics Display Regions API are illustrated in Figure 9.2.

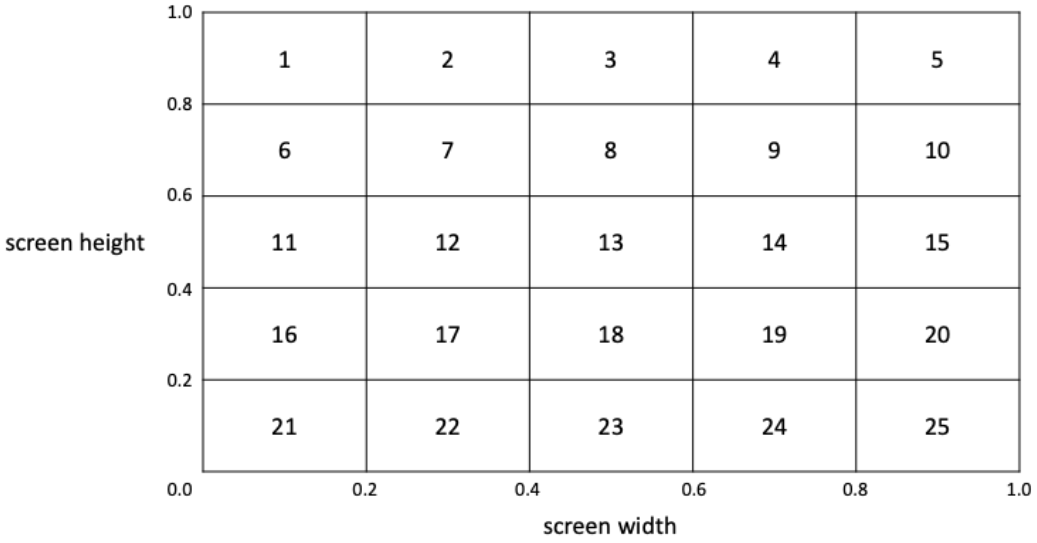


Figure 9.2 Graphics Display Regions Layout and Numbers.

The Graphics Display Regions Request semantics are defined in Table 9.74 and the syntax shall be as defined in the schema file `org.atsc.graphicsDisplayRegions-request.json`.

Table 9.74 Graphics Display Regions Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.graphicsDisplayRegions"
`occupiedRegions`	1	integer (0 … 33554431)	Provides each graphics display regions number that would be occupied by the broadcaster application graphical layout as indicated by the appropriate bit set in an integer. (Note: 33,554,431 = 2^25 – 1)

`occupiedRegions` – The bits set to '1' in this required integer indicates which display regions in the 5x5 display grid shown in Figure 9.2 above are occupied, even partially, by the graphical layout that the Broadcaster Application has prepared to present for display. The least significant bit of the integer set to '1' indicates that the Broadcaster Application plans to present a graphical layout that will occupy display region number 1 shown in Figure 9.2, the next significant bit set to '1' indicates display region number 2, and so on. The 5x5 display grid has 25 display regions so only the first 25 bits of the integer might be set to '1'. Therefore, the seven most significant bits shall be set to '0'.

The Graphics Display Regions Response semantics are defined in Table 9.75 and the syntax shall be as defined in schema file `org.atsc.graphicsDisplayRegions-response.json`.

Table 9.75 Graphics Display Regions Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
result	oneOf X		Returned on successful request otherwise the error structure is returned
error	oneOf X		

result – If the graphics display regions request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, "{}", result object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

9.7.9 Media Asset Selection API for MMT

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

The Request semantics are defined in Table 9.76 and the syntax is defined in the schema file org.atsc.asset.selection-request.json. Additional semantic definitions of parameters follow the table.

Table 9.76 MMT Media Asset Selection Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.asset.selection"
assetId	1	string	The asset ID to be selected

assetId – This required string is expected to correspond to a value of the asset ID in the MP Table [30]. Asset ID may have a UUID (Universally Unique Identifier), or a URI (Uniform Resource Identifier) scheme and the type is the byte array with a length of 32 bits.

The MMT Media Asset Selection Response semantics are defined in Table 9.77 and the syntax shall be as defined in the schema file org.atsc.asset.selection-response.json. Additional semantic definitions of parameters follow the table.

Table 9.77 MMT Media Asset Selection Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Empty object on successful asset selection. The error structure is returned if unsuccessful.
error	oneOf X		See Section 8.3.3

`result` – If the MMT media asset selection request is successful, the Receiver is expected to respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -33 – The specified asset cannot be selected.

9.8 Mark Unused API

The Mark Unused API may be used by the currently executing Broadcaster Application to indicate to the Application Context Cache 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 Request semantics are defined in Table 9.78 and the syntax shall be as defined in the schema file `org.atsc.cache.markUnused-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.78 Mark Unused Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.cache.markUnused"
`elementUri`	1	string (uri-reference)	The relative path within the Application Context Cache of the element to be marked unused

`elementUri` – This required URI shall be the path of an element within the Broadcaster Application's Application Context Cache that is to be marked unused.

The Mark Unused Response semantics are defined in Table 9.79 and the syntax shall be as defined in the schema file `org.atsc.markUnused-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.79 Mark Unused Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful request. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – If the mark unused request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – Content not found

For example, the Broadcaster Application may wish to indicate that a particular replacement ad was not needed anymore after it had been used. The files comprising the replacement ad, that is the `Period` XML fragment and the associated audio and video segments, would be sent in a particular directory hierarchy labeled "ads/16" for this example. The Broadcaster Application would

mark the `Period` XML fragment file as unused, and all segments referenced by the `Period` XML fragment file as unused as well. Note that the Broadcaster Application would be responsible for marking all of the resources of the ad as unused, the Receiver is not responsible for processing the `Period` XML fragment file to discover referenced resources. Alternatively, the Broadcaster Application could mark the entire directory, "ads/16", unused if the directory only contained the replacement ad `Period` XML fragment file and its associated segments.

The RPC request to mark the "ads/16" directory unused would be formatted as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.cache.markUnused",
    "params": {
        "elementUri": "ads/16"
    },
    "id": 42
}
```

The Receiver might respond with the following on success:

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

Similarly, to mark a single file unused, the Broadcaster Application could make the following request:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.cache.markUnused",
    "params": {
        "elementUri": "news/storyImages/photo12.png"
    },
    "id": 42
}
```

The Receiver might respond with the following on success:

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

Standard HTTP failure codes are expected to 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 have indeterminate results in that some Receivers may not have made the element unavailable and thus respond positively to the request while others may immediately respond with an error status.

9.9 Content Recovery APIs

9.9.1 Query Content Recovery State API

A Broadcaster Application may wish to know whether it is being managed using content recovery via watermarking and/or fingerprinting as specified in A/336 [5]. This allows the Broadcaster Application to offer different functionality in content recovery scenarios than may be offered when broadcast signaling is present and, in content recovery scenarios, it allows the application to identify the presence of modifications that may be introduced by an upstream device (such as a Set-Top Box (STB)) as discussed in Annex A of A/336 [5] on an ongoing basis during its execution and alter its behavior accordingly.

The Query Content Recovery State Request semantics are defined in Table 9.80 and the syntax shall be as defined in the schema file `org.atsc.query.contentRecoveryState-request.json`.

Table 9.80 Query Content Recovery State Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.contentRecoveryState"

The Query Content Recovery State Response semantics are defined in Table 9.81 and the syntax shall be as defined in the schema file `org.atsc.query.contentRecoveryState-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.81 Query Content Recovery State Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`audioWatermark`	0..1	integer (0 … 2)	Indicates the audio watermark detection state
`videoWatermark`	0..1	integer (0 … 2)	Indicates the video watermark detection state
`audioFingerprint`	0..1	integer (0 … 2)	Indicates the audio fingerprint detection state
`videoFingerprint`	0..1	integer (0 … 2)	Indicates the video fingerprint detection state
`error`	oneOf X		See Section 8.3.3

`audioWatermark` – This integer value shall indicate one of the following states of audio watermark detection:

0: if the Receiver is not employing both audio watermark detection and application signaling recovered as specified in A/336 for application management;

1: if the Receiver is employing both audio watermark detection and application signaling recovered as specified in A/336 for application management and the Receiver is not currently detecting a VP1 Audio Watermark Segment as defined in A/336;

2: if the Receiver is employing both audio watermark detection and application signaling recovered as specified in A/336 for application management and the Receiver is currently detecting a VP1 Audio Watermark Segment as defined in A/336.

`videoWatermark` – This integer value shall indicate one of the following states of video watermark detection:

0: if the Receiver is not employing both video watermark detection and application signaling recovered as specified in A/336 for application management;

1: if the Receiver is employing both video watermark detection and application signaling recovered as specified in A/336 for application management and the Receiver is not currently detecting a VP1 Video Watermark Segment or any other video watermark message as defined in A/336;

2: if the Receiver is employing both video watermark detection and application signaling recovered as specified in A/336 for application management and the Receiver is currently detecting a VP1 Video Watermark Segment or any other video watermark message as defined in A/336.

`audioFingerprint` – This integer value shall indicate one of the following states of audio fingerprint recognition:

0: if the Receiver is not employing both audio fingerprint recognition and application signaling recovered as specified in A/336 for application management;

1: if the Receiver is employing both audio fingerprint recognition and application signaling recovered as specified in A/336 and the Receiver is not currently recognizing an audio fingerprint;

2: if the Receiver is employing both audio fingerprint recognition and application signaling recovered as specified in A/336 and the Receiver is currently recognizing an audio fingerprint.

`videoFingerprint` – This integer value shall indicate one of the following states of video fingerprint recognition:

0: if the Receiver is not employing both video fingerprint recognition and application signaling recovered as specified in A/336;

1: if the Receiver is employing both video fingerprint recognition and application signaling recovered as specified in A/336 and the Receiver is not currently recognizing a video fingerprint;

2: if the Receiver is employing both video fingerprint recognition and application signaling recovered as specified in A/336 and the Receiver is currently recognizing a video fingerprint.

If a key/value pair is absent in the result, it indicates that the value of the key/value pair is 0 (i.e., the associated capability is not supported by the Receiver).

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.contentRecoveryState",
    "id": 122
}
```

If the Receiver supports application management using application signaling recovered from both audio and video watermarks as specified in A/336 and both are currently being detected, the Receiver is expected to respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "audioWatermark": 2,
        "videoWatermark": 2
    },
    "id": 122
}
```

9.9.2 Query Display Override API

A Broadcaster Application may wish to know if the Receiver is receiving "display override" signaling obtained via watermarking (as defined in A/336 [5]) indicating that modification of the video and audio presentation should not be performed, and whether the Receiver is actively enforcing that signaling by suppressing access by the Broadcaster Application to presentation resources ("resource blocking").

This information may be employed by the Broadcaster Application, for example, to:

- Ensure efficient utilization of Receiver and network resources (e.g., it may choose to not request resources from a broadband server when those resources cannot be presented to the user);
- Preserve an accurate representation of the user experience (e.g., to accurately report the viewability of a dynamically inserted advertisement as may be required by an ad viewability standard); or
- Comply with the requirements of the display override state, for example by halting any audio or video modification, in the event that the Receiver is not performing resource blocking.

The Query Display Override Request semantics are defined in Table 9.82 and the syntax shall be as defined in the schema file `org.atsc.query.displayOverride-request.json`.

Table 9.82 Query Display Override Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.displayOverride"

The Query Display Override Response semantics are defined in Table 9.83 and the syntax shall be as defined in the schema file `org.atsc.query.displayOverride-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.83 Query Display Override Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`resourceBlocking`	0..1	boolean	"true" indicates that the Receiver is blocking the output of the Broadcaster Application
`displayOverride`	0..1	boolean	"true" indicates that a display override condition is in effect
`error`	oneOf X		See Section 8.3.3

`resourceBlocking` – This optional Boolean value indicates if the Receiver is blocking the Broadcast Application from presenting video and audio pursuant to an active display override state as defined in A/336 [5].

`displayOverride` – This optional Boolean value shall be true if a display override condition is currently in effect per a video watermark Display Override Message as specified in Section 5.1.9 of A/336 [5] or per an audio watermark display override indication as specified in Sections 5.2.4 and 5.4.2 of A/336 [5]. Otherwise, the value shall be false.

If either or both key/value pairs are absent in the result, this shall indicate that the value of the absent key/value pair is false.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.displayOverride",
    "id": 62
}
```

If the display override condition is currently indicated via audio watermark as specified in Section 5.2.4 of A/336 [5] and the Receiver is blocking the Broadcaster Application from presenting video and audio, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
        "resourceBlocking": true,
        "displayOverride": true
    },
    "id": 62
}
```

9.9.3 Query Recovered Component Info API

When content recovery via watermarking or fingerprinting is employed, it is useful for the Broadcaster Application to be able to determine which video or audio components of a service are being received by the Receiver (e.g., as a result of selection by the user on an upstream device).

This can enable the Broadcaster Application to modify the on-screen placement or the language of overlaid graphics or audio to conform to the characteristics of the received component.

During content recovery via watermarking or fingerprinting, the Receiver receives component descriptors in a recovery file specified in Section 5.4.2 of A/336 [5]. This API provides a means for the Broadcaster Application to access descriptors that were recovered for components that were identified using watermarks or fingerprints.

The Query Recovered Component Info Request semantics are defined in Table 9.84 and the syntax shall be as defined in the schema file `org.atsc.query.recoveredComponentInfo-request.json`.

Table 9.84 Query Recovered Component Info Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.recoveredComponentInfo"

The Query Recovered Component Info Response semantics are defined in Table 9.85 and the syntax shall be as defined in the schema file `org.atsc.query.recoveredComponentInfo-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.85 Query Recovered Component Info Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`component`	1	array	An array of recovered media components
`items`	1..N		
`mediaType`	1	examples	"audio", "video", "both"
`componentID`	0..1	string	
`descriptor`	0..1	string	
`error`	oneOf X		See Section 8.3.3

`mediaType`, `componentID` and `descriptor` are the data values associated with the media components received by the Receiver, as given in the fields of the same name in a `componentDescription` element of a recovery file as specified in Table 5.29 of A/336 [5].

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.recoveredComponentInfo",
    "id": 39
}
```

If a `componentDescription` element of the recovery file lists an audio component with the `componentID` value "1" and the `descriptor` value "component descriptor string 1", and a video component with the `componentID` value "2" and the `descriptor` value "component descriptor string 2" associated with the components received by the Receiver, the Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"component": [
        {
            "mediaType": "audio",
            "componentID": "1",
            "descriptor": "component descriptor string 1"
        },
        {
            "mediaType": "video",
            "componentID": "2",
            "descriptor": "component descriptor string 2"
        }
    ]},
    "id": 39
}
```

9.9.4 Content Recovery State Change Notification API

The Content Recovery State Change Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application if the content recovery state as defined in Query Content Recovery State API in Section 9.9.1 changes from one state to another different state in which at least one property value changes and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The Content Recovery State Change Notification semantics are defined in Table 9.86 and the syntax shall be as defined in the schema file `org.atsc.notify-contentRecoveryStateChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.86 Content Recovery State Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"contentRecoveryStateChange"
`audioWatermark`	0..1	integer (0 … 2)	Indicates the audio watermark detection state
`videoWatermark`	0..1	integer (0 … 2)	Indicates the video watermark detection state
`audioFingerprint`	0..1	integer (0 … 2)	Indicates the audio fingerprint detection state
`videoFingerprint`	0..1	integer (0 … 2)	Indicates the video fingerprint detection state

`audioWatermark`, `videoWatermark`, `audioFingerprint` and `videoFingerprint` are defined in the Query Content Recovery State API in Section 9.9.1.

If a key/value pair is absent in the `params`, it indicates that the value of the key/value pair is 0.

For example, if the user changes from a non-watermarked service to a new service marked with both audio and video watermarks and both audio and video watermarks are detected and being used for content recovery in the Receiver, the Receiver notifies the Broadcaster Application the content recovery state change as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "contentRecoveryStateChange",
        "audioWatermark": 2,
        "videoWatermark": 2
    }
}
```

9.9.5 Display Override Change Notification API

The Display Override Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the display override state or resource blocking state as defined in Query Display Override API in Section 9.9.2 changes from one state to another different state and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The Display Override Change Notification semantics are defined in Table 9.87 and the syntax shall be as defined in the schema file `org.atsc.notify-displayOverrideChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.87 Display Override Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"displayOverrideChange"
`resourceBlocking`	0..1	boolean	"true" indicates that the Receiver is blocking the output of the Broadcaster Application
`displayOverride`	0..1	boolean	"true" indicates that a display override condition is in effect

`resourceBlocking` and `displayOverride` are defined in Query Display Override API in Section 9.9.2.

If a key/value pair is absent in the `params`, it indicates that the value of the key/value pair is false.

For example, if the display override state changes from inactive to active and the Receiver is blocking the currently executing Broadcaster Application from presenting video and audio, the Receiver notifies the Broadcaster Application the Display Override change as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "displayOverrideChange",
        "resourceBlocking": true,
        "displayOverride": true
    }
}
```

9.9.6 Recovered Component Info Change Notification API

The Recovered Component Info Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the video or audio components of a service being received by the Receiver change (e.g., as a result of selection by the user on an upstream device) and the Broadcaster Application has subscribed to receive such notifications via the API specified in Section 9.3.1.

The Recovered Component Info Change Notification semantics are defined in Table 9.88 and the syntax shall be as defined in the schema file `org.atsc.notify-recoveredComponentInfoChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.88 Recovered Component Info Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"recoveredComponentInfoChange"
`mediaType`	1	examples	"audio", "video", "both"
`componentID`	0..1	string	
`descriptor`	0..1	string	

`mediaType`, `componentID` and `descriptor` are defined in the Query Recovered Component Info API in Section 9.9.3.

For example, if the user at an upstream device of the Receiver changed from Spanish to English audio track described by the `componentID` value "1" and `descriptor` value "`component description string 3`", the Receiver notifies the Broadcaster Application of the recovered component changed as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "recoveredComponentInfoChange",
        "mediaType": "audio",
        "componentID": "1",
        "descriptor": "component description string 3"
    }
}
```

9.10 Filter Codes APIs

The Receiver may use Filter Codes to selectively download NRT data files by comparing the stored Filter Codes with the Filter Codes associated with the NRT data files in the EFDT. Refer to Section 6.5.3 for a complete description of filter code processing.

9.10.1 Set Filter Code Instances API

The Set Filter Code Instances API can be issued by a Broadcaster Application to notify the Receiver to store the specified Filter Code Instances.

The Set Filter Code Instances Request semantics are defined in Table 9.89 and the syntax shall be as defined in the schema file `org.atsc.setFilterCodes-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.89 Set Filter Code Instances Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.setFilterCodes"
`filters`	1	array	A list of Filter Code Instance definitions
`items`	1..N		Each item object describes a Filter Code Instance
`filterCode`	1	integer	The Filter Code value of the Filter Code Instance
`expires`	0..1	string (date-time)	Indicates the date and time when the Filter Code Instance expires

`filters` – A required array of Filter Code Instance definitions.

`filterCode` – An unsigned integer associated with personalization categories as determined by the broadcaster. This attribute sets the value portion of the Filter Code Instance. It is the broadcaster's responsibility to maintain a scope of uniqueness of values between Filter Code Instances within an `AppContextID`.

`expires` – This string shall be represented by the date-time JSON data type as defined in the JSON Schema specification [19] to indicate the expiry of a Filter Code Instance. Filter Code Instances shall not be used after expiry. If the `expires` value is omitted, then no expiration is indicated, and the Filter Code Instance shall expire when the Broadcaster Application terminates.

Note that populating this field using XML strings formatted as `"xs:dateTime"` should be avoided since `"xs:dateTime"` has legal instances that are not compliant with JSON `"date-time"` format.

The Set Filter Code Instances Response semantics are defined in Table 9.90 and the syntax shall be as defined in the schema file `org.atsc.setFilterCodes-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.90 Set Filter Code Instances Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful set operation. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – If the set Filter Code Instances request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

In the following example, the Broadcaster Application sets two Filter Code Instances for the Receiver to use:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.setFilterCodes",
    "params": {
        "filters": [
          {"filterCode": 101, "expires": "2016-07-17T09:30:47Z"},
          {"filterCode": 102}]
    },
    "id": 57
}
```

Upon success, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 57
}
```

9.10.2 Clear Filter Code Instances API

The Clear Filter Code Instances API can be issued by a Broadcaster Application to notify the Receiver to clear either all the defined Filter Code Instances or a set of specified Filter Code Instances.

The Clear Filter Code Instances Request semantics are defined in Table 9.91 and the syntax shall be as defined in the schema file `org.atsc.clearFilterCodes-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.91 Clear Filter Code Instances Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.clearFilterCodes"
`filters`	0..1	array	A list of Filter Code Instances to be cleared
items	0..N	integer	A Filter Code identifying a Filter Code Instance to be cleared

`filters` – An optional list of unsigned integer Filter Codes that the Receiver shall use to remove matching Filter Code Instances from processing. If the `filters` property is absent or empty, all active Filter Code Instances shall be cleared. Note that Filter Code Instances cleared in this way shall no longer be included in NRT file processing (see Section 6.5.3) regardless of any defined expiration time.

The Clear Filter Code Instances Response semantics are defined in Table 9.92 and the syntax shall be as defined in the schema file `org.atsc.clearFilterCodes-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.92 Clear Filter Code Instances Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful clear operation. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – If the clear Filter Code Instances request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -28 – One or more of the provided filter codes are unknown. All other requested filter codes are expected to be cleared.

In the following example, the Broadcaster Application provides two Filter Codes for the Receiver to clear:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.clearFilterCodes",
    "params": {
        "filters": [101, 102]
    },
    "id": 62
}
```

Upon success, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 62
}
```

9.11 Keys APIs

The APIs in this section allow the Broadcaster Application, with the Receiver's permission, to access certain specified remote-control keys. The Query Device Info API (Section 9.12) requires a set of input keys to be defined, including the navigation keys, `Select` key and `Back` key, that may be requested by the Broadcaster Application. A key designated as the `BAAppear` key is also defined by the Query Device Info API which provides a way for the Receiver to notify the Broadcaster Application that the user wishes to interact. The expected use of the `BAAppear` key is described below. The Query Device Info API may also provide additional keys that the Broadcaster Application may request. The APIs that allow the Broadcaster Application to manage the input keys that are available are defined in this section.

It is necessary that the Broadcaster Application request any input keys from the Receiver that it expects to handle as input from the user. Broadcaster Applications should only request input keys applicable to the current user interaction and should avoid requesting input keys that are not needed for the current interactive operations. This allows re-purposing of keys by the Receiver. If keys beyond the `BAAppear` key have not been requested, then any key listed by the Query Device Info API response `deviceInput` object (see Section 9.12) and not in use by the Receiver may result in a `BAAppear` key being issued to the Broadcaster Application.

For example, at startup, a Broadcaster Application should use the Query Device Info API to determine what keys are available on the Receiver and map them to the keys necessary to provide its intended user experience. In this example, the Broadcaster Application offers a variety of other functions the user may be interested in such as weather, news or broadband-based VOD content. To notify the user that the Broadcaster Application is present and available, the Broadcaster Application should display the `BAAppear` text and image provided by the Query Device Info API response.

Unless a user is already interacting with the Broadcaster Application, the Broadcaster Application would request the `BAAppear` input key. This is referred to as the launch screen. The launch screen may become invisible after this depending on the Broadcaster Application design. However, when invisible, the expectation is that any key passed to the Broadcaster Application should bring visibility back if interactivity is available. The Broadcaster Application design may choose to directly launch interactive features upon receipt of the `BAAppear` input key. The Broadcaster Application may choose to make the launch page visible at any time to let the user know that interactivity is available.

On receipt of the `BAAppear` input key, the Broadcaster Application would request the appropriate input keys from the Receiver via the Request Keys API and begin showing dialogs for the user to interact with the Broadcaster Application. As the user clicks keys on the remote control, the Broadcaster Application is expected to respond appropriately, displaying new pages and dialogs. Note that the Broadcaster Application may choose to request or relinquish keys as enumerated by the Receiver in response to the Broadcaster Application's Query Device Info API call. Receivers are not expected to re-purpose keys given to a Broadcaster Application via a successful Request Keys API call. Similar to the launch screen, the Broadcaster Application in this interactive context is expected to be visible when any key is provided to it by the Receiver. This provides users the opportunity to navigate to the launch page and allows Receivers to re-purpose relinquished keys as needed.

In some situations, the Receiver may display a dialog or other prompt for user input "in front of" or "over" the Broadcaster Application taking focus away from the BA. This would happen if the user requested a dialog to change Receiver settings, for example. Input from the user would then be processed by the Receiver dialog even though the Broadcaster Application had requested the same input keys. The Broadcaster Application can detect whether or not it currently is in focus by using the W3C "`onblur`" event. See [UI Events] in CTA-5000-G [9].

Once the user has completed interacting with the Broadcaster Application, the Broadcaster Application should dismiss all of its dialogs and visible pages and relinquish all of the requested keys. As a reminder of the applicable `BAAppear` key, the Broadcaster Application may display the `BAAppear` text and image.

It is expected that the Broadcaster Application provides a path using the `Back` key, or otherwise, to the launch page. It is expected that further `Back` key presses at a launch page should make the Broadcaster Application invisible.

When at the launch page, Receivers may re-purpose keys except the `BAAppear` key if needed.

9.11.1 Keycode Consistency

It is expected that the keycodes provided by the APIs in this section and Section 9.12 are consistent across physical, virtual, and alternative methods for enabling user input. For example, keys from a physical keyboard and a virtual keyboard on the display are expected to return the same codes for the same keys.

9.11.2 Request Keys API

A Broadcaster Application can request to receive optional key presses that are typically used and processed by Receivers. Note that "key press" in this context represents a user action of pressing a key on a keypad or remote control and should not be confused with any W3C event mechanisms. For example, numeric key presses on the remote control are typically used by the underlying Receiver to tune directly to a specific channel. However, the Broadcaster Application may wish to present a data entry UI to accept numeric data from the user in order to perform a specific action or to solicit input from the viewer. In this case, the Broadcaster Application can request the Receiver to temporarily re-route numeric key presses to itself. Based on the Receiver manufacturer, the Receiver may reject this request, in which case, the Broadcaster Application may choose to display a soft keyboard on the TV screen or resort to using other types of device input.

The Request Keys Request semantics are defined in Table 9.93 and the syntax shall be as defined in the schema file `org.atsc.request.keys-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.93 Request Keys Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.request.keys"
`keys`	1	array	A list of requested keys to be associated with the Broadcaster Application
`items`	1..N	string	A requested key name

`keys` – This required parameter shall be an array of strings, each representing a particular remote-control key or type of key the Broadcaster Application would like the Receiver to forward. The available key strings are defined as follows:

- "Numeric" – Indicates the numeric keys 0-9.
- "ArrowUp" – Indicates the "ArrowUp" input key.
- "ArrowDown" – Indicates the "ArrowDown" input key.
- "ArrowRight" – Indicates the "ArrowRight" input key.
- "ArrowLeft" – Indicates the "ArrowLeft" input key.
- "Back" – Indicates the "Back" input key.

- "BAAppear" – Indicates that the input key dedicated to the Broadcaster Application. See Section 9.12.
- <other> – Indicates any of the strings in W3C "UI Events KeyboardEvent key Values," Section 3 [32].

Requested keys that are not known to the Receiver are expected to be ignored.

The Request Keys Response semantics are defined in Table 9.94 and the syntax shall be as defined in the schema file `org.atsc.request.keys-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.94 Request Keys Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`accepted`	1		A list of accepted keys
`items`	0..N	string	
`error`	oneOf X		See Section 8.3.3

`accepted` – The Receiver shall respond with a JSON-RPC response object including a `"result"` object. The result object includes a string array indicating the keys for which the request was successful. The strings supplied shall correspond to the strings allowed in the `"keys"` parameter of the request operation above. It can be assumed that any requested keys that are not included in the `"accepted"` array were not accepted.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Broadcaster Application requests receipt of numeric keys and Channel Up and Channel Down arrows, it can issue this request to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.request.keys",
    "params": {"keys": ["Numeric", "ChannelUp", "ChannelDown"]},
    "id": 43
}
```

If the Receiver grants the numeric key presses but not the Channel Up and Channel Down key presses, the Receiver could respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"accepted": ["Numeric"]},
    "id": 43
}
```

9.11.3 Relinquish Keys API

A Broadcaster Application can relinquish previous requests for key presses. This would be used after a request made via the Request Keys API (Section 9.11.1) to return the handling of key presses to the Receiver.

The Relinquish Keys Request semantics are defined in Table 9.95 and the syntax shall be as defined in the schema file `org.atsc.relinquish.keys-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.95 Relinquish Keys Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.relinquish.keys"
`keys`	1	array	A list of keys to be relinquished from the Broadcaster Application
items	0..N	string	A key name to be relinquished

`keys` – This required parameter is an array of strings, each representing a particular remote-control key or type of key the Broadcaster Application no longer wishes to process and is relinquishing to the Receiver. If the `keys` parameter is not provided, is equal to `"All"` or is an empty array, then all keys previously requested for forwarding by the Broadcaster Application shall be relinquished. Any specified key that was not previously requested or is not known to the Receiver shall be ignored. Available key strings are defined in the `keys` property semantic definition of the Request Keys API in Section 9.11.1.

The Relinquish Keys Response semantics are defined in Table 9.96 and the syntax shall be as defined in the schema file `org.atsc.relinquish.keys-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.96 Relinquish Keys Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object if the keys were successfully relinquished. The error structure is returned if unsuccessful.
`error`	oneOf X		See Section 8.3.3

`result` – If the relinquish keys request is successful, the Receiver shall respond with a JSON-RPC response object with an empty, `"{}"`, `result` object.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application can issue this request to the Receiver:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.relinquish.keys",
    "params": {
       "keys": ["ChannelUp", "ChannelDown"]
   },
   "id": 44
}
```

The Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 44
}
```

9.11.4 Request Keys Timeout

To help avoid application misbehavior, the Receiver may force key requests to be relinquished by a Broadcaster Application after a certain amount of time, defined by each Receiver manufacturer. This is referred to as a request key timeout. Prior to a request key timeout, the Receiver sends a warning notification to the Broadcaster Application to provide the application time to respond to the timeout. The Broadcaster Application may, at this point, choose to issue another Request Keys API call or it may allow the key request(s) to time out. If another Request Keys API call is issued, it may or may not be accepted by the Receiver.

The Request Key Timeout Notification semantics are defined in Table 9.97 and the syntax shall be as defined in the schema file `org.atsc.notify-requestKeyTimeout.json`. Additional semantic definitions of parameters follow the table.

Table 9.97 Request Key Timeout Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"requestKeyTimeout"
`timeout`	1	array	Provides a list of keys about to be removed from Broadcaster Application input
`items`	1..N		
`key`	1	string	The name of the key
`time`	1	integer	The time in seconds when the key will no longer be available to the Broadcaster Application

`timeout` – A required array that identifies each key that is nearing timeout along with the number of seconds until timeout occurs.

`key` – This parameter is a string representing a particular remote-control key or type of key which the Receiver wishes to take back from the Broadcaster Application. Available key strings are defined in the keys property semantic definition of the Request Keys API in Section 9.11.1.

`time` – An integer number of seconds indicating the timeout for the given key or type of key. The timeout value shall be a minimum of 3 seconds.

For example, the Receiver may send the following notification to indicate that Channel Up and Channel Down key requests are expected to time out in 3 seconds:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "requestKeyTimeout",
        "timeout": [
            {"key": "ChannelUp", "time": 3},
            {"key": "ChannelDown", "time": 3}
        ]
    }
}
```

9.12 Query Device Info API

The Query Device Info API provides an interface between a Broadcaster Application and the Receiver to retrieve device-specific information. It is a generic conduit between the Receiver and the Broadcaster Application to provide basic device information including make and model of device, along with optional additional key/value pair information about the device. The format and definition of the optional additional key/value pairs are manufacturer-specific and not specified here. Specific parameters may be defined as part of a business relationship between a broadcaster and a device manufacturer.

The two unique ID parameters (`deviceId`, `advertisingId`) in the response below are expected to be initialized once by the Receiver to afford long-term persistence across all Services and Receiver power cycles and to be provided by the Receiver depending on authorization granted by the user. The ability to authorize the disclosure of either the unique `deviceId` or `advertisingId` parameters, either per broadcaster, per Broadcaster Application or as a whole, affords increased privacy to the user.

The Query Device Info API request `params` object is optional. If `params` is omitted (or if `deviceInfoProperties` is omitted or is an empty array), the Receiver is expected to respond with only the device make and model and the `deviceInput` object. The Broadcaster Application can then use the device make and model to determine which additional properties to query. The `deviceInfoProperties` is an array of desired properties, and the Receiver provides the values of these properties in the response.

The Query Device Info Request semantics are defined in Table 9.98 and the syntax shall be as defined in the schema file `org.atsc.query.deviceInfo-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.98 Query Device Info Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.deviceInfo"
`deviceInfoProperties`	0..1	array	List of device properties to be returned
`items`	0..N	string	The name of a particular device property of interest to the Broadcaster Application

`deviceInfoProperties` – This parameter is an array of strings, each representing a particular aspect of the device about which the Broadcaster Application is interested.

The Query Device Info Response semantics are defined in Table 9.99 and the syntax shall be as defined in the schema file `org.atsc.query.deviceInfo-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.99 Query Device Info Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
deviceMake	1	string	A string containing the make of the device
deviceModel	1	string	A string containing the model of the device
deviceInput	1	object	A mapping of key names to values
ArrowUp	1	integer	Provides the key code value for the ArrowUp key
ArrowDown	1	integer	Provides the key code value for the ArrowDown key
ArrowRight	1	integer	Provides the key code value for the ArrowRight key
ArrowLeft	1	integer	Provides the key code value for the ArrowLeft key
Select	1	integer	Provides the key code value for the Select key
Back	1	integer	Provides the key code value for the Back key
BAAppear	1	object	Defines the Broadcaster Application "Launch" key
label	1	string	Provides a recognizable name for the key, e.g., "NextGen App"
keycode	1	integer	Provides the key code value for the "BAAppear" key
img	0..1	string	Provides an inline encoded image of the key or label
<other>	0..N	integer	<other> is any of the strings in W3C "UI Events KeyboardEvent key Values", Section 3 [32] whose value provides the associated key code
deviceInfo	0..1	object	A collection of key/value pairs defining specific attributes of the device
deviceId	0..1	string	A persistent, globally unique UUID associated with the device
advertisingId	0..1	string	A persistent, globally unique UUID associated with advertising on the device
deviceCapabilities	0..1	string	
deviceSupportedWebSocketAPIs	0..1	array	The set of A/344 API methods supported
items	0..N		*items may be either strings or objects as described below*
	anyOf	string	A/344 Methods supported, identified as fully qualified method names (name only)
	anyOf	object	Object describing method and revision supported
method	1	string	A/344 Methods supported, identified as fully qualified method names

rev	0..1	string	Specific revision date of method supported
deviceSupportedDRMs	0..1	array	The set of DRM system id URNs supported
items	0..N	string	
error	oneOf X		See Section 8.3.3

`deviceMake` – This required string indicates the manufacturer of the Receiver.

`deviceModel` – This required string indicates the model of the Receiver.

`deviceInput` – This required object defines the available user input key name and codes of the Receiver user interface as well as a required "launch key" for the Broadcaster Application. The `deviceInput` object contains a collection of input key/value pairs where the key is the user input name, and the value is the associated integer code. The Receiver is expected to provide all input keys and their associated values that the Broadcaster Application may request. The Broadcaster Application may use the Key APIs defined in Section 9.11 to request any or all of the keys defined in the `deviceInput` object. It is expected that the keycodes provided are consistent across all input methods (see Section 9.11.1).

In addition to the collection of input keys, the `deviceInput` object shall contain a `BAAppear` object as described below.

`ArrowUp` – This key event is sent when the user triggers the up arrow directional key on the remote control or through some equivalent action.

`ArrowDown` – This key event is sent when the user triggers the down arrow directional key on the remote control or through some equivalent action.

`ArrowRight` – This key event is sent when the user triggers the right arrow directional key on the remote control or through some equivalent action.

`ArrowLeft` – This key event is sent when the user triggers the left arrow key on the remote control or through some equivalent action.

`Select` – This key event is sent when the user triggers a remote-control key signaling selection. Examples of common remote-control keys that may provide a selection input include 'OK', 'Enter', 'Select' and 'Return' keys. The Select event indicates that the user wishes the Broadcaster Application to take an action based on the current focus. For example, if an icon is highlighted, the Select key code indicates that the user wishes to engage with the associated function.

`Back` – This key event is sent when the user triggers a remote-control key signaling the desire to move back a step, depending on the current Broadcaster Application state. For example, the Back event may indicate to the Broadcaster Application that activities on the current display are completed and the user wishes to return to a previous display.

`BAAppear` – This required object defines the Broadcaster Application "launch key" label, keycode, and an optional image encoding.

`label` – This required string contains text that is intended to be presented to the user that assists in defining the "launch key" on the remote-control device.

`keycode` – This required integer is the keycode of the "launch key".

`img` – This optional string contains an encoding of an image to assist the user in locating the "launch key" which may be virtual (i.e., not a key on the remote). The string shall conform to the data URL scheme defined in IETF RFC 2397 [28] and the "base64" encoding defined in RFC 4648

[27]. The Receiver is required to support PNG, JPEG and GIF as provided in CTA-5000-G, Section 3.6 [9].

`<other>` – These keys provide the mapping of the various key strings as described in in W3C "UI Events KeyboardEvent key Values", Section 3 [32] to the corresponding key code. Note that any keys explicitly specified in Table 9.99 (e.g., `"Select"`) shall not be included again. Any input key included in this mapping shall be available for request as described in Section 9.11.

`deviceInfo` – This optional object includes key/value pairs. The `deviceInfo` is included in the response if the request includes one or more `deviceInfoProperties` strings corresponding to information the Receiver can supply.

`deviceId` – This optional string returns a globally unique UUID as defined in RFC 4122 [26] using the urn:uuid syntax when authorized for the particular Broadcaster Application. When absent, it is not supported by the Receiver. When present and all zeros (`"urn:uuid:00000000-0000-0000-0000-000000000000"`), the value of the user setting to provide this identifier is disabled for the given service indicating that the user has explicitly not authorized provision of the value.

`advertisingId` – This optional string returns a globally unique UUID as defined in RFC 4122 [26] using the urn:uuid syntax when authorized for the particular Broadcaster Application. When absent, it is not supported by the Receiver. When present and all zeros (`"urn:uuid:00000000-0000-0000-0000-000000000000"`), the value of the user setting to provide this identifier is disabled for the given service indicating that the user has explicitly not authorized provision of the value.

`deviceCapabilities` – This optional string describes the capabilities, or features, of the Receiver and shall conform to A/332, Section 5.2.2.3.3.2 "Device Capabilities Syntax and Semantics" [4]. This is, in essence, a standardized version of the `deviceInfo` string.

`deviceSupportedWebSocketAPIs` – This optional array of objects identifies the A/344 WebSocket API methods that the Receiver supports as described in this standard, using fully-qualified method names (e.g., `"org.atsc.query.deviceInfo"`). The objects in the array shall be strings, `"method"` / `"rev"` objects, or a combination of strings and `"method"` / `"rev"` objects. When it is a string, each is a fully qualified method name, in which case the method supported is as described in the revision of this standard as described in Section 8.2. The `"method"` / `"rev"` objects in the array shall consist of "method" and optionally "rev" described below.

`method` – A method that the Receiver supports as described in this standard (or another, see `"rev"` below), using a fully-qualified method name.

`rev` – This optional field identifies that the method is supported per a specific revision of A/344, formatted as described in Section 8.2 Interface Bindings, the `"rev="` structure. If this field is not present, the identified version is as described in Section 8.2, in the `"rev="` parameter.

`deviceSupportedDRMs` – This optional array of strings identifies the DRM systems that the Receiver supports. Each string shall be a DRM system id URN in the form of `"urn:uuid:<uuid>"`, such as described in "Protection System-Specific Identifiers" at [43].

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, a query from the Broadcaster Application may look like this:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.deviceInfo",
    "id": 92
}
```

A Receiver manufactured by the "Acme" company with model number "A300" might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
       "deviceMake": "Acme",
       "deviceModel": "A300",
       "deviceInput": {
          "ArrowUp": 38,
          "ArrowDown": 40,
          "ArrowRight": 39,
          "ArrowLeft": 37,
          "Select": 13,
          "Back": 461,
          "BAAppear": {
              "label": "NextGen App",
              "keycode": 500,
              "img": ""
          }
       },
       "deviceSupportedWebSocketAPIs": [
          "org.atsc.query.ratingLevel",
          {"method": "org.atsc.query.service", "rev": "20190502"},
          {"method": "org.atsc.scale-position"}
       ],
       "deviceSupportedDRMs": [
          "urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed",
          "urn:uuid:e2719d58-a985-b3c9-781a-b030af78d30e"
       ]
    },
    "id": 92
}
```

If the Broadcaster Application recognizes this make and model, it might know that additional information about this Receiver may be retrieved by indicating specific device properties in the request. Note that not all Receivers would be expected to recognize the `deviceInfoProperties` strings given in this example:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.deviceInfo",
    "params": {"deviceInfoProperties": ["numberOfTuners", "yearOfMfr"]},
    "id": 93
}
```

This Receiver might respond with:

```
<-- {
    "jsonrpc": "2.0",
    "result": {
      "deviceMake": "Acme",
      "deviceModel": "A300",
      "deviceInput": {
        "ArrowUp": 38,
        "ArrowDown": 40,
        "ArrowRight": 39,
        "ArrowLeft": 37,
        "Select": 13,
        "Back": 461,
        "BAAppear": {
          "label": "NextGen App",
          "keycode": 500,
          "img": ""
        }
      },
      "deviceInfo": {
        "numberOfTuners": 1,
        "yearOfMfr": 2017
      },
      "deviceSupportedWebSocketAPIs": [
        "org.atsc.query.ratingLevel",
        {"method": "org.atsc.query.service", "rev": "20190502"},
        {"method": "org.atsc.scale-position"}
      ],
      "deviceSupportedDRMs": [
        "urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed",
        "urn:uuid:e2719d58-a985-b3c9-781a-b030af78d30e"
      ]
    },
    "id": 93
}
```

9.13 RMP Content Synchronization APIs

The RMP Content Synchronization APIs defined in this section provide similar functionality to the APIs available for the W3C `HTMLMediaElement` as provided in [9]. Readers are encouraged to review the `HTMLMediaElement` APIs to obtain background information on the semantics behind the RMP APIs described here.

The times used in the APIs within this section are essential to the proper operation of controlling the RMP. The relationships between the `startDate` and `currentTime` parameters used in the APIs and the media time are shown in Figure 9.3. See the detailed definitions of the `startDate` and `currentTime` parameters in Section 9.13.1 Query RMP Media Time API.

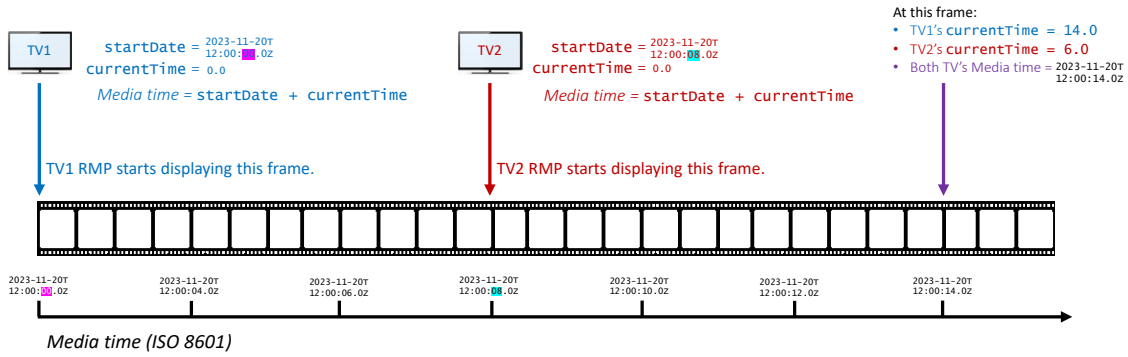


Figure 9.3 RMP Media Time Representation

In this representation, the wall-clock time equals the media time plus the broadcast chain delay plus the distribution delay plus the Receiver internal delay. Each TV may be presenting a given media time at a different wall-clock time.

9.13.1 Query RMP Media Time API

A Broadcaster Application may wish to know the current presentation time of the broadcast content being presented by the Receiver. This enables the Broadcaster Application to present supplemental content that is synchronized to the broadcast content. For example, a Broadcaster Application may display a graphical overlay at a particular moment during the presentation.

The Query RMP Media Time Request semantics are defined in Table 9.100 and the syntax shall be as defined in the schema file `org.atsc.query.rmpMediaTime-request.json`.

Table 9.100 Query RMP Media Time Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.rmpMediaTime"

The Query RMP Media Time Response semantics are defined in Table 9.101 and the syntax shall be as defined in the schema file `org.atsc.query.rmpMediaTime-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.101 Query RMP Media Time Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned
currentTime	1	number (>=0.0)	The current presentation time of the RMP
startDate	0..1	string (date-time)	The start time of the media timeline
error	oneOf X		See Section 8.3.3

`currentTime` – This required floating-point number value shall represent the current presentation time of the content being presented by the RMP, expressed as an offset, in seconds, to `startDate`. It shall have the same meaning as the `@currentTime` attribute of the `HTMLMediaElement` given in [9] that represents the official playback position of the content.

`startDate` – This optional date-time value represents an absolute time reference for the start (i.e., the zero time) of the media timeline of the content being presented by the RMP. It has the same meaning as "timeline offset" in [9] (i.e., the value provided from the `getStartDate()` method of an `HTMLMediaElement`). The date-time JSON data type shall be formatted as defined in the JSON Schema specification [19].

When the RMP is presenting content compliant with [41], the following requirements apply to the reported values of `startDate` and `currentTime`:

- The value of `startDate` represents the sum of `MPD@availablilityStartTime` in the MPD that was in use by the RMP when it began playing or recording the presentation and the time offset on the DASH Media Presentation timeline at which the RMP began playing or recording the presentation. When content delivered via broadband allows the RMP to seek to a position in the presentation earlier than the time at which RMP began playing or recording the content (e.g., live time-shift), the time offset on the DASH Media Presentation timeline shall be the earliest seek-able time offset in the content. Note that the media format of the recorded content is Receiver specific.
- When recorded content is being presented, both the `startDate` and the `currentTime` values shall have the same respective values as the `startDate` and the `currentTime` values applied during presentation of the live version of the recorded content.

If no explicit date and time is available for the content being present (e.g., the downloaded content), `startDate` shall be absent in the response. Otherwise, `startDate` shall be present in the response.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.rmpMediaTime",
    "id": 61
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"currentTime": 3600.033,
               "startDate": "2019-01-01T23:59:59.590Z"
    },
    "id": 61
}
```

9.13.2 Query RMP UTC Time API DEPRECATED

This API has been deprecated. Broadcaster Applications needing the current time of day should use the W3C Date.now() method available in the User Agent.

A Broadcaster Application may wish to know the UTC time being used by the RMP. This time should be used when working with the UTC presentation timeline instead of the JavaScript API, e.g., Date.now() of the User Agent.

The Query RMP UTC Time Request semantics are defined in Table 9.102 and the syntax shall be as defined in the schema file `org.atsc.query.rmpUTCTime-response.json`.

Table 9.102 Query RMP UTC Time Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.rmpUTCTime"

The Query RMP UTC Time Response semantics are defined in Table 9.103 and the syntax shall be as defined in the schema file `org.atsc.query.rmpUTCTime-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.103 Query RMP UTC Time Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`utcTime`	1	string (date-time)	The current UTC time
`error`	oneOf X		See Section 8.3.3

`utcTime` – This required date-time value shall be the current UTC time, which is the PTP time signaled in the broadcast PLP `L1D_time` fields converted to UTC (adjusted by the LLS

SystemTime fields). The date-time JSON data type shall be formatted as defined in the JSON Schema specification [19].

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.rmpUTCTime",
    "id": 62
}
```

The Receiver might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"utcTime": "2019-01-01T23:59:56.320Z"},
    "id": 62
}
```

9.13.3 Query RMP Playback State API

A Broadcaster Application may wish to know the playback state of the content being presented or prepared for presentation by the RMP. This allows the application to make adjustments in presenting supplemental content based on the playback state of the content. For example, the application may suspend presentation of supplemental content if playback of the presentation is paused due to content buffer underflow ("buffering") or user input or stopped due to reaching the end of a VOD content stream.

The Query RMP Playback State Request semantics are defined in Table 9.104 and the syntax shall be as defined in the schema file org.atsc.query.rmpPlaybackState-request.json.

Table 9.104 Query RMP Playback State Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.query.rmpPlaybackState"

The Query RMP Playback State Response semantics are defined in Table 9.105 and the syntax shall be as defined in the schema file org.atsc.query.rmpPlaybackState-response.json. Additional semantic definitions of parameters follow the table.

Table 9.105 Query RMP Playback State Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful request otherwise the error structure is returned

	playbackState	1	integer (-1 … 3)	The current RMP playback state
error		oneOf X		See Section 8.3.3

playbackState – This integer value shall indicate one of the following playback states of the RMP:

- -1 if the content is initializing, connecting and the state cannot be determined, for example there is a time window between changing the channel, accessing the SLT, MPD and initializing the channel where the state is unclear;
- 0 if the content is actively playing, and if encrypted, there are necessarily also valid DRM licenses and the CDM is decrypting the content;
- 1 if the playback is paused for any reason and has not ended (e.g., seeking or stalled, paused for user interaction, waiting for user input, stopped due to errors);
- 2 if the playback has ended (e.g., the end of the content is reached).
- 3 if the content is encrypted and not viewable (i.e., there are no valid keys and the CDM is not decrypting the content).

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.rmpPlaybackState",
    "id": 63
}
```

And if the RMP is playing back content, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"playbackState": 0},
    "id": 63
}
```

9.13.4 Query RMP Playback Rate API

A Broadcaster Application may wish to know the speed at which the content is being presented by the RMP. This allows the application to make adjustments in presenting supplemental content based on the playback speed of the content. For example, the application may choose to suspend presentation of supplemental content during trick-play mode.

The Query RMP Playback Rate Request semantics are defined in Table 9.106 and the syntax shall be as defined in the schema file org.atsc.query.rmpPlaybackRate-request.json.

Table 9.106 Query RMP Playback Rate Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.query.rmpPlaybackRate"

The Query RMP Playback Rate Response semantics are defined in Table 9.107 and the syntax shall be as defined in the schema file `org.atsc.query.rmpPlaybackRate-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.107 Query RMP Playback Rate Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request otherwise the error structure is returned
`playbackRate`	1	number	The current RMP playback rate
`error`	oneOf X		See Section 8.3.3

`playbackRate` – This required floating-point value indicates the playback speed of the content currently being presented by the RMP with the same meaning as the attribute `playbackRate` of `HTMLMediaElement` as given in [9].

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, the Broadcaster Application makes a query:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.query.rmpPlaybackRate",
    "id": 65
}
```

And if the RMP is playing back content at the normal speed, the Receiver would respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"playbackRate": 1.0},
    "id": 65
}
```

9.13.5 RMP Media Time Change Notification API

The RMP Media Time Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the current playback position of the content being presented by the RMP changed. This API has the same meaning functionally as the `timeupdate` event of `HTMLMediaElement` as given in [9]. The Receiver notifies the Broadcaster Application

using this API when the official playback position of the RMP changes as part of normal or trick playback.

The RMP Media Time Change Notification semantics are defined in Table 9.108 and the syntax shall be as defined in the schema file `org.atsc.notify-rmpMediaTimeChange.json`. Additional semantic definitions of parameters follow the table.

Note that although data is included in a single notification that is needed for modifying or replacing a Period, there is still the risk of a race condition on use of the state of these or related fields.

Table 9.108 RMP Media Time Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"rmpMediaTimeChange"
currentTime	1	number (>= 0.0)	The current presentation time of the RMP
startDate	0..1	string (xs:timeDate)	The start time of the media timeline
playbackState	0..1	number	As defined in Section 9.13.3.
refClock	0..1	string (xs:timeDate)	Current reference time derived from the L1D preamble and modified by LLS SystemTime table.
adaptationId_video	0..1	string	Space-separated list of MPD.Period.AdaptationSet.AdaptationSet@id
adaptationId_audio	0..1	string	Space-separated list of MPD.Period.AdaptationSet.AdaptationSet@id
adaptationId_text	0..1	string	MPD.Period.AdaptationSet.AdaptationSet@id
periodId	0..1	string	MPD.Period@id
periodStart	0..1	string (xs:timeDate)	Start time of the current Period.
duration	0..1	number (>0)	Duration in seconds of the current Period.
source	0..1	array	id, uriType and sourceType of the media playback for each adaptationId.
items	0..N		
id	1	string	Adaptation set id of the element
uriType	1	enum	"GSID", "XLinkURN" or "RMPURL"
sourceType	1	enum	As defined in Table 9.109

`currentTime` and `startDate` are defined in Query RMP Media Time API in Section 9.13.1. If the `startDate` key/value is absent in the `params`, it indicates that the value of the key/value pair is unchanged.

`playbackState` – Shall be as defined in Section 9.13.3.

`refClock` – Shall be the current reference time derived from the L1D preamble and modified by LLS SystemTime table [3]. If `playbackState` is present, this parameter shall be present.

`adaptationId_video` – Shall be the space-separated list of actively rendering `MPD.Period.AdaptationSet@id` of video. If `playbackState` is present, this parameter shall be present.

`adaptationId_audio` – Shall be the space-separated list of actively rendering `MPD.Period.AdaptationSet@id` of the audio. If `playbackState` is present, this parameter shall be present.

`adaptationId_text` – Shall be the actively rendering `MPD.Period.AdaptationSet@id` of the text. If `playbackState` is present, this parameter shall be present.

`periodId` – Shall be the `MPD.Period@id` of the current Period. If `playbackState` is present, this parameter shall be present.

`periodStart` – Shall be the `MPD.Period@start` of the current Period. If `playbackState` is present, this parameter shall be present.

`duration` – Shall be the duration of the current Period. If `playbackState` is present and the duration is known, this parameter shall be present.

`source` – Shall be an array of objects comprised of `id`, `uriType` and `sourceType` parameters defining the current playing media. If `playbackState` is present this array shall be present and contain elements describing each of the adaptation sets present in the current playing media.

`id` – Shall contain the adaptation set id.

`uriType` – Shall be one of: "`GSID`", "`XLinkURN`" or "`RMPURL`" according to `setRMPUrl` in Section 9.7.3. "`GSID`" shall be a `globalServiceID` associated with the service as given in the SLT in `SLT.Service@globalServiceID`. See A/331 [3] Section 6.3.

`sourceType` – Shall provide a `sourceType` according to Table 9.109 using the first `mediaType` that applies from top to bottom in the table. If `playbackState` is present, this shall be present.

Table 9.109 `sourceType` Definition

sourceType	Type of RMP Source
"broadcast"	If the RMP is playing broadcast stream and the current Period has not been replaced by an XLink resolution.
"broadband"	If the RMP is playing a broadband stream and the current Period has not been replaced by an XLink resolution.
"setrmpurl"	If the RMP is playing a stream set using `setRMPURL`.
"xlink"	If the RMP is playing any stream type and the current Period has been replaced via XLink resolution.
"other"	If the RMP is playing out from a source not known in advance or not been configured to playout any stream.

For a basic example, the Receiver may notify the Broadcaster Application of the media time change every 250 to 500 msec during the normal playback of the current service. A notification provided shortly after the previous notification which contained a `currentTime` value 3600.033 might be:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params":{
       "msgType": "rmpMediaTimeChange",
       "currentTime": 3600.283
     }
}
```

A more complex example when `playbackState` is present and the service contains two audio tracks (one of which is delivered over broadband) is as follows:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgtype": "rmpMediaTimeChange",
        "currentTime": 350.424,
        "startDate": "2019-01-01T23:59:59.590Z",
        "playbackState": 0,
        "refClock": "2019-01-01T23:59:53.590Z",
        "adaptationId_video": "2",
        "adaptationId_audio": "1 4",
        "adaptationId_text": "3",
        "periodId": "P1",
        "periodStart": "2019-01-01T23:50:30.000Z",
        "duration": 600,
        "source": [
            {
                "id": "2",
                "uriType": "GSID",
                "sourceType": "broadcast"
            },
            {
                "id": "1",
                "uriType": "GSID",
                "sourceType": "broadcast"
            },
            {
                "id": "4",
                "uriType": "GSID",
                "sourceType": "broadband"
            },
            {
                "id": "3",
                "uriType": "GSID",
                "sourceType": "broadcast"
            }
        ]
    },
    "id": 913
}
```

9.13.6 RMP Playback State Change Notification API

The RMP Playback State Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the playback state of the RMP as defined in Query RMP Playback State API in Section 9.13.3 changes from one value to another different value.

The RMP Playback State Change Notification semantics are defined in Table 9.110 and the syntax shall be as defined in the schema file `org.atsc.notify-rmpPlaybackStateChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.110 RMP Playback State Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"rmpPlaybackStateChange"
`playbackState`	1	integer (-1 … 3)	The new RMP playback state

`playbackState` – This integer value shall indicate the new playback state that is one of the playback states defined in the Query Playback State API in Section 9.13.3.

For example, if the user at the Receiver pauses the playback of the time-shift broadcast content, the Receiver notifies the Broadcaster Application of the playback state as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "rmpPlaybackStateChange",
        "playbackState": 1
    }
}
```

9.13.7 RMP Playback Rate Change Notification API

The RMP Playback Rate Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the playback speed as defined in the Query RMP Playback Rate API in Section 9.13.4 changes from one value to another different value. This API has the same meaning functionally as the `ratechange` event of `HTMLMediaElement` as given in [9].

The RMP Playback Rate Change Notification semantics are defined in Table 9.111 and the syntax shall be as defined in the schema file `org.atsc.notify-rmpPlaybackRateChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.111 RMP Playback Rate Change Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"rmpPlaybackStateChange"
`playbackRate`	1	number	The new RMP playback rate

`playbackRate` shall be as defined in the Query RMP Playback Rate API in Section 9.13.4.

For example, if the user at the Receiver performs fast-forward playback of the time-shift content at 2 times normal playback speed, the Receiver notifies the Broadcaster Application of the playback speed changes as shown below:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "rmpPlaybackRateChange",
        "playbackRate": 2
    }
}
```

9.13.8 RMP Media Asset Change Notification API

The RMP Media Asset Change Notification API is expected to be issued by the Receiver to the currently executing Broadcaster Application if the current playback position of the Asset being presented by the RMP changed.

The RMP Media Asset Change Notification semantics are defined in Table 9.112 and the syntax shall be as defined in the schema file `org.atsc.notify-rmpMediaAssetChange.json`. Additional semantic definitions of parameters follow the table.

Table 9.112 RMP Media Asset Change Notification Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
method	1	string	"org.atsc.notify"
msgType	1	enum	"rmpMediaAssetChange"
currentTime	1	number (>= 0.0)	The current presentation time of the RMP
startDate	0..1	string (xs:timeDate)	The start time of the media timeline
`assets`	1..N	array	List of Assets
`items`			
`assetId`	1	string	MP Table – Identifier_mapping() [30], 10.6.3 Identifier mapping
`assetType`	0..1	string	The type of the Asset
`presentationTime`	1	string (date-time)	Presentation time of the first MFU of the MPU
`duration`	1	number (>0)	Duration in seconds of the current Asset
`sourceType`	0..1	string	broadcast or broadband
`packageId`	1	integer	a unique identifier of the Package [30], 6.2 Package

`currentTime` and `startDate` are defined in Query RMP Media Time API in Section 9.13.1. If the `startDate` key/value is absent in the `params`, it indicates that the value of the key/value pair is unchanged.

`assetId` – This required integer shall correspond to the value of the asset ID in the MP Table. The `assetId` may have a UUID (Universally Unique Identifier), or URI (Uniform Resource Identifier) scheme and the type is the byte array with a length of 32 bits.

`assetType` – This value, if provided, shall be the four-character code ("4CC") type registered in MP4REG (http://www.mp4ra.org) [30], 10.3.9 MP table.

`presentationTime` – Shall be the presentation time of the first AU in the designated MPU [30], 10.5.2 MPU timestamp descriptor represented by the date-time JSON data type as defined in the JSON Schema specification [19].

`duration` – Shall be the Asset duration.

`sourceType` – Shall provide a `sourceType` according to Table 9.109

`packageId` – Shall be the package id of the package including the assets.

9.14 DRM APIs

The APIs in this section can be used by the Broadcaster Application to support the RMP or AMP handling of encrypted content. Two generic APIs are defined. A "notification" API is used by the Receiver to pass a message associated with an identified DRM System to the Broadcaster Application. An "operation" API is used by the Broadcaster Application to pass a message associated with an identified DRM System to the Receiver. These APIs support the needs of both the AMP and the RMP.

Refer to the ATSC A/362 Recommended Practice on Digital Rights Management (DRM) [39] for details on DRM operations within an ATSC 3.0 Receiver environment.

9.14.1 DRM Notification API

The DRM Notification API may be issued by the Receiver to the Broadcaster Application in order to deliver a DRM-related notification. A Broadcaster Application which receives this notification can use the DRM Operation API, defined in Section 9.14.2, to exchange a message with the Receiver's underlying content protection system, ultimately resulting in the delivery of the license/key required by the RMP or AMP for decryption of protected content.

The DRM Notification semantics are defined in Table 9.113 and the syntax shall be as defined in the schema file `org.atsc.notify-DRM.json`. Additional semantic definitions of parameters follow the table.

Table 9.113 DRM Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"DRM"
`systemId`	1	string	The identifier of the DRM system sending the notification
`message`	1	array	An array of DRM-system-proprietary JSON objects that are dependent on the systemId
`items`	0..N	object	

`msgType` – This required string shall be set to "DRM" to identify this notification API.

`systemId` – This string shall be set to a DRM system identifier, `@schemeIdUri`, as defined in the DASH-IF IOP, Section 7.6 [41]. For example, the UUID string value "`1077efec-c0b2-4d02-ace3-3c1e52e2fb4b`" corresponds to the common system ID of W3C EME. In the case of DASH, the value corresponds to the value of `@schemeIdUri` of the ContentProtection descriptor of the MPD. In the case of MMT, the value corresponds to the value of the `system_UUID` of the `security_property_descriptor` of the MPU specified in A/331 Section 7.2.4 [3].

`message` – This shall be the message passed from the content protection system formatted as an array of JSON objects.

The DRM Notification API may be used by the Receiver to notify the Broadcaster Application that a particular DRM license object that had been requested via the DRM Operation API (Section 9.14.2) has been retrieved from the broadcast.

It is out of scope of the present specification how a Broadcaster Application interacts with a license server and the exact format of messages passed from the content protection system to the Broadcaster Application using this DRM Notification API. The format of messages passed across the interface defined in this API depends on the content protection system used by the broadcaster.

9.14.2 DRM Operation API

The DRM Operation API can be issued by a Broadcaster Application to pass a message to the Receiver in order to play protected content as defined by Section 5.7 of [8]. This API can be used along with the DRM Notification API as defined in Section 9.14.1 which is issued by the Receiver to notify a message to a Broadcaster application in order to inform information about content protection.

Similar to W3C EME [31] which is an extension of the HTML5 media element, a Broadcaster Application can communicate with a license server and pass messages for license/key exchange to the underlying content protection system via this API. These APIs are simpler than W3C EME since the APIs only provide a message exchange mechanism and the content of the messages conveyed in the API, including any control such as installing/updating/removing licenses, are specific to an underlying content protection system and thus not specified here. Note that a Broadcaster Application needs to know the sequence of interactions with a broadcaster's web server and a license server and also the procedure for exchange of messages with the underlying content protection system of the Receiver.

The DRM Operation Request semantics are defined in Table 9.114 and the syntax shall be as defined in the schema file `org.atsc.drmOperation-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.114 DRM Operation Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.drmOperation"
`systemId`	1	string	The identifier of the DRM system for which the message is intended
`service`	1	string (uri)	Specifies the `globalServiceID` of the currently selected service
`message`	1	array	An array of DRM-system-proprietary JSON objects that are dependent on the systemId
`items`	0..N	object	

`systemId` – This string shall be set to a DRM system identifier, `@schemeIdUri`, as defined in the DASH-IF, Section 7.6 [39].

`service` – This required string shall indicate the `globalServiceID` associated with the currently selected service as given in the SLT in `SLT.Service@globalServiceID` to which this DRM message is associated.

`message` – This shall be a message specific to the content protection system in use formatted as an array of JSON objects.

The DRM Operation Response semantics are defined in Table 9.115 and the syntax shall be as defined in the schema file `org.atsc.drmOperation-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.115 DRM Operation Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful operation otherwise the error structure is returned
`message`	0..1	array	A list of content protection system messages
`items`	0..N	object	A system proprietary JSON object that is dependent on the systemId
`error`	oneOf X		See Section 8.3.3

`message` – This shall be the response to the command formatted as an array of JSON objects specific to the content protection system in use.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -6 – The `globalServiceID` supplied by the `service` property cannot be not found.
- -14 – The DRM system identifier provided is not supported by or is not known to the Receiver.

In the case of the AMP, when the DRM Operation API is used by the Broadcaster Application to ask the Receiver to fetch a particular DRM license file from the broadcast Entitlement Management service, the message object may include key/value pairs to specify the `Content-Location` of the file.

9.15 XLink APIs

The APIs in this section provide a mechanism allowing the Broadcaster Application to replace a DASH element marked with an XLink. Note that more than one element can be returned (e.g. replacing a Period with more than one Period). There are two APIs defined:

- A XLink Resolution Notification API allowing the Receiver to notify the Broadcaster Application that the RMP has detected an MPD element with an `xlink:href` attribute. Notifications only occur if the Broadcaster Application has subscribed to the notification (see Section 9.3.1.1). The Broadcaster Application may then provide an alternate element using the XLink Resolved API.
- The XLink Resolved API is used by the Broadcaster Application to provide a replacement MPD URL or inline text to be used by the RMP as an alternative to the element within which the `xlink:href` was detected. It is possible for the Receiver to ignore the XLink Resolved request if the resource is not found, or it is too late to make the substitution.

9.15.1 XLink Resolution Notification API

An XLink Resolution Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application when the Receiver Media Player (RMP) encounters a

`@xlink:href` attribute in an element of the MPD. The Receiver may resolve any `xlink:href` with URL scheme, "https:", without notifying the Broadcaster Application. For all other URIs, the Receiver is expected to notify the Broadcaster Application to resolve the `xlink:href`, if the Broadcaster Application is subscribed to the XLink Resolution Notification (see Section 9.3.1.1). The Broadcaster Application is expected to respond by using the XLink Resolved API (see Section 9.15.2) to replace the element in which the `xlink:href` appeared. There can be more than one `xlink:href` in an MPD and if so, the Receiver is expected to call XLink Resolution Notification multiple times.

The timing of `xlink:href` resolution by the Receiver is controlled by `xlink:actuate` which should be set to "`onLoad`" to cause the Receiver to resolve the XLinks on receipt of the MPD. Note that this is not the default. Therefore, `xlink:actuate="onLoad"`, should normally be present to give the Broadcaster Application the maximum time to resolve the link.

The XLink Resolution Notification semantics are defined in Table 9.116 and the syntax shall be as defined in the schema file `org.atsc.notify-xlinkResolution.json`. Additional semantic definitions of parameters follow the table.

Table 9.116 XLink Resolution Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"xlinkResolution"
`xlink`	1	string (uri)	The `xlink:href` URI detected in a DASH element
`elementType`	0..1	string	One of "MPD", "Period" or any other MPD element name on which xlink is permitted
`elementId`	1	string	The value of the element `@id`.

`xlink` – The `xlink:href` string detected in a DASH element.

elementType – The name of the element type in the MPD, e.g., "MPD", "Period", etc. If this property is omitted the default is "Period".

elementId – The value of the element's `@id` associated with this XLink.

Once an XLink Resolution Notification is sent, the Broadcaster Application may decide to provide alternate content using its own criteria or it may choose not to.

For example, the broadcast stream may contain advertisements that could be targeted at various viewers based on a variety of criteria. The time span of the default advertisement and the associated content would be described in a particular DASH element structure that would contain an `xlink:href` attribute. To accommodate the advertisement replacement, the Broadcaster Application would subscribe to the XLink Resolution Notification API.

On detection of an element containing the `xlink:href`, the Receiver is expected to issue the following message:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
         "msgType": "xlinkResolution",
         "xlink": "urn:rid:xbc4399FB77-3939EA47",
         "elementType": "Period",
         "elementId": "2"
    }
}
```

9.15.2 XLink Resolved API

After the XLink Resolution Notification (Section 9.15.1) is received, the Broadcaster Application determines which alternate content to replace the default content with and makes a request to the Receiver using the XLink Resolved API. The Broadcaster Application provides either a URI of the XML fragment of the alternate content or the actual element text in the request. The Receiver is expected to process the replacement XML fragment URI or element text, as appropriate, to replace the element with one or more alternate elements. If the Receiver is not able to replace the default content, the returned disposition object is expected to contain the appropriate code and description indicating the reason for the failure.

The XLink Resolved Request semantics are defined in Table 9.117 and the syntax shall be as defined in the schema file `org.atsc.xlinkResolution-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.117 XLink Resolved Request Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	
`method`	1	string	"org.atsc.xlinkResolution"
`xlink`	1	string(uri)	The XLink value from the `xlink:href` attribute in the MPD element
`mpdURL`	oneOf X	string (uri)	The URI of a replacement MPD fragment containing the XML element or elements to replace the element Period containing the `xlink:href` attribute
`element`	oneOf X	string	The inline text to replace the element containing the `xlink:href` attribute
`status`	0..1	integer	The status of the Broadcaster Application's action
`elementType`	0..1	string	The type of XML element containing the `xlink:href`
`elementId`	0..1	string	The `@id` of the element being replaced

`xlink` – This required string shall be the value of the `xlink:href` attribute on some element. This string corresponds to the value received in the XLink Resolution Notification parameters (see Section 9.15.1).

`mpdURL` – If provided as an alternative to `element`, this string shall provide the URI of an XML fragment to replace the element containing the `xlink:href` attribute. One of either the `mpdURL` or `element` properties shall be provided. If this property is provided, it shall be a fully qualified URI, whether referencing the XML element over broadband or in the Application Context

Cache. Note that for an XML element in the Application Context Cache, the full URI can be constructed using the Base URI provided using the Query Receiver Web Server URI API as described in Section 9.2.7.

`element` – If provided as an alternative to `mpdURL`, this string shall contain the actual element text to replace the element containing the `xlink:href` attribute. One of either the `mpdURL` or `element` properties shall be provided.

status – The status of the Broadcaster Application reaction as follows:

0 - Element not replaced and Receiver is free to act on it

1 – Element replaced. Receiver is to attempt to replace the element

-1 – Element not replaced. Receiver is forbidden acting on it

elementType – The name of the element being replaced, e.g., "MPD", "Period", etc. The default is "Period".

elementId – The `@id` of the element being replaced.

After receiving the XLink Resolved request, the Receiver may respond indicating that the original default element was successfully replaced with either the requested XML element URI or the supplied element. If the Receiver cannot successfully complete the request, an error code is expected to be returned. The Receiver may choose to delay the response until after the element was successfully replaced or may choose to respond with an error condition immediately. Broadcaster Applications are expected to consider the difference in timing when handling the XLink Resolved Response. The XLink Resolved Response semantics are defined in Table 9.118 and the syntax shall be as defined in the schema file `org.atsc.xlinkResolution-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.118 XLink Resolved Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Returned on successful request submission otherwise the error structure is returned
`elementType`	1	string	The name of the (MPD) element, e.g., "Period"
`elementId`	1	string	The `@Id` of the `elementType`
`error`	oneOf X		See Section 8.3.3

elementType – The name of the element being replaced, e.g., "MPD", "Period", etc.

elementId – The @id of the element being replaced.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – Requested content cannot be found. For example, invalid URL.
- -30 – The request was not received in time to replace the default content.
- -31 – The provided `elementType` and/or `elementId` not found.
- -32 – The provided (MPD) element content is not valid.

As an example, the Broadcaster Application submits the following request, presuming that the Base URI is `"http://192.168.32.117:8182/s02gPkwZx14iO"`:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.xlinkResolution",
    "params": {
         "xlink": "urn:rid:xbc4399FB77-3939EA47",
         "mpdURL": "http://192.168.32.117:8182/s02gPkwZx14iO",
         "status": 1
    },
    "id": 104
}
```

If the Receiver can replace the default content with the alternate XML element URI, it responds and a disposition indicating successful replacement with the alternate element used as shown below.

```
<-- {
    "jsonrpc": "2.0"
    "result": {
        "elementType": "Period",
        "elementId": "3"
    },
    "id": 104
}
```

Alternatively, if the Receiver could not use the replacement XML element URI, as requested by the Broadcaster Application, it returns a disposition indicating the reason for the failed request.

```
<-- {
    "jsonrpc": "2.0",
    "error": {
        "code": -30,
        "message": "Too late"
    },
    "id": 104
}
```

As a further example, the Broadcaster Application submits the following request containing the actual alternate period text instead of an `mpdURL`:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.xlinkResolution",
    "params": {
         "xlink": "urn:xbc4399FB77-3939EA47",
         "element": "<Period start='PT0S'>
              <AdaptationSet mimeType='video/mp4' … >
                <SegmentTemplate timescale='90000' …
                  media='alt-$Number$.mp4v' duration='90000'
                  startNumber='32401' />
                <Representation id='v2' width='1920' height='1080' … />
              </AdaptationSet>
            </Period>",
         "status": 1
    },
    "id": 105
}
```

If the Receiver can replace the default content with the alternate period, it responds with a disposition indicating successful replacement with the alternate period used as shown below.

```
<-- {
    "jsonrpc": "2.0"
    "result": {
        "elementType": "Period",
        "elementId": "3"
    },
    "id": 105
}
```

9.16 Prepare for Service Change API

The API in this section provides a mechanism for the Receiver to request that the Broadcaster Application prepare for a service change by releasing resources and performing other cleanup preparing for the situation where the Broadcaster Application might or might not be torn down during the service change.

The Prepare for Service Change API allows the Receiver to

- indicate to the Broadcaster Application that a service change is initiating,
- inform the Broadcaster Application to release resources generally, and
- optionally instruct the Broadcaster Application to release specific resources.

A Prepare for Service Change request, if implemented, is expected to be issued by the Receiver to the currently executing Broadcaster Application when a service change is in process, and the Receiver wishes the Broadcaster Application to release allocated resources to prepare for the possibility that the Broadcaster Application could be terminated and torn down during the service change. Optionally, the Receiver may enumerate specific resources to be released.

The Prepare for Service Change request semantics are defined in Table 9.119, and the syntax shall be as defined in the schema file `org.atsc.prepSvcChange-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.119 Prepare for Service Change Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.prepSvcChange"
fromService	1	uri	Specifies the globalServiceID of the current service.
toService	0..1	uri	Specifies the globalServiceID of the future service.
resources	0..1	array	An array of tokens describing specific resources to be released.
items	1..N	string	A requested resource token

`fromService` – This required URI shall indicate the `globalServiceID` associated with the currently selected service as given in the SLT in `SLT.Service@globalServiceID`.

`toService` – This optional URI, if present, shall indicate the `globalServiceID` associated with the service to be acquired as given in the SLT in `SLT.Service@globalServiceID`.

`resources` – This optional array, if present, shall include tokens that indicate specific resources to be released by the Broadcaster Application. Notwithstanding the presence or absence of any particular token in this field, the Broadcaster Application is expected to release as many resources as practicable. Values in this array shall be from Table 9.120.

Table 9.120 Service Change Resource Tokens

Token	Description
AMP	Any Application Media Player resources.
(other values)	Permitted for user-private or extension

The Prepare for Service Change Response semantics are defined in Table 9.121 and the syntax shall be as defined in the schema file `org.atsc.prepSvcChange-response.json`. Additional semantic definitions of parameters follow the table.

Table 9.121 Prepare for Service Change Response Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	Matches the request id value
result	oneOf X		Returned on successful operation otherwise the error structure is returned
resources	0..1	array	An array of tokens describing specific resources that were released
items	1..N	string	A requested resource token
error	oneOf X		See Section 8.3.3

`resources` – This optional array, if present, shall include tokens that indicate specific resources that were released by the Broadcaster Application. Values in this array shall be from Table 9.120. For resources that the Broadcaster Application has not allocated, but were requested by the Receiver to be released, the Broadcaster Application should include those resources in this

response. Note that if a Broadcaster Application fails to release (or to identify as released) specific resources identified in the request, the Receiver might tear down the Broadcaster Application without respect to any Broadcaster Applications associated with the `toService`.

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- None – There are no errors specific to this API.

For example, if the Receiver is in the process of a service change, it would request release of resources as follows:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.prepSvcChange",
    "params": {
      "fromService": "https://tv.atsc/OldService",
      "toService": "https://tv.atsc/NewService",
      "resources": ["AMP"]
    },
    "id": 22
}
```

The Broadcaster Application might respond:

```
<-- {
    "jsonrpc": "2.0",
    "result": {"resources": ["AMP"]},
    "id": 22
}
```

9.17 MMT AssetLink APIs

The APIs in this section provide a mechanism allowing the Broadcaster Application to replace Assets in MMT whose `asset_location` is a URI as is the case when the `location_type` value in `MMT_general_location_info` of the MPT, Section 7.2.3 is marked as "0x05" [30]. Figure 9.4 provides a diagram of the relationship between the MMT signaling structures.

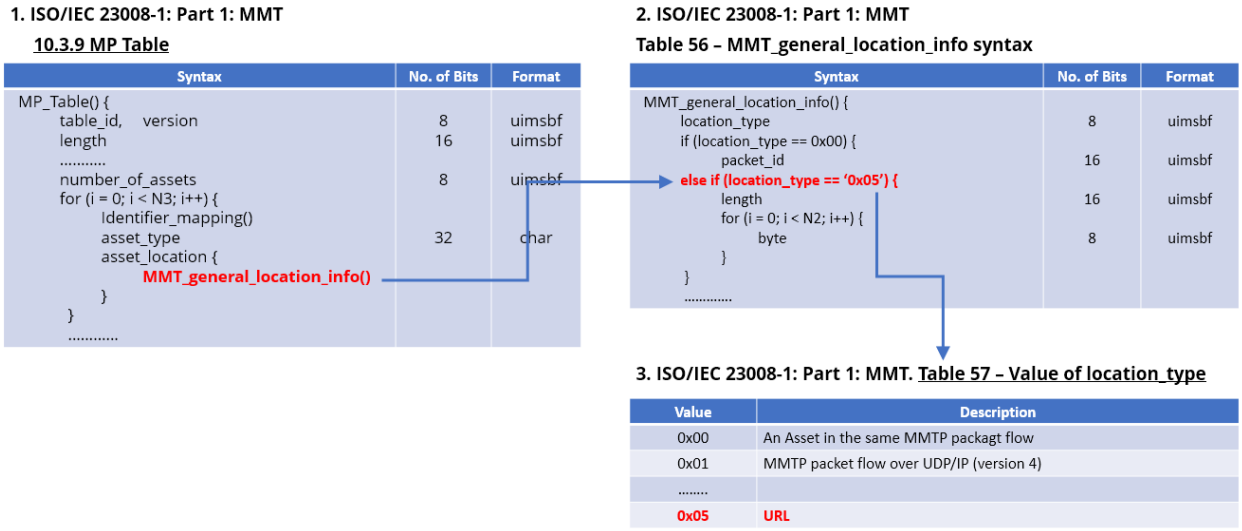


Figure 9.4 Relationship of MMT signaling tables.

There are two APIs defined:

- An AssetLink Resolution Notification API allows the Receiver to notify the Broadcaster Application that the RMP has detected an MMT Asset has an `asset_location` that is a URI. This Asset is termed the "target Asset" in this section. Notifications only occur if the Broadcaster Application has subscribed to the notification (see Section 9.3.1). The Broadcaster Application may then provide an alternate MMT Asset using the AssetLink Resolved API. An MMT Asset is delivered in the external or internal URL not in the same MMTP Over-the-Air packet flow and a single file containing the MMT Signaling Message and MPU Media Data.
- The AssetLink Resolved API is used by the Broadcaster Application to provide a replacement MMT Asset URL or inline text to be used by the RMP as an alternative to the target Asset. It is possible for the Receiver to ignore the AssetLink Resolved request if the resource is not found, or it is too late to make the substitution.

9.17.1 AssetLink Resolution Notification API

An AssetLink Resolution Notification is expected to be issued by the Receiver to the currently executing Broadcaster Application when the Receiver Media Player (RMP) encounters an Asset of the MPT with an `asset_location` defined as a URI [30]. The Receiver may resolve any `asset_location` with a URL scheme, "https:", without notifying the Broadcaster Application. For all other URIs, the Receiver is expected to notify the Broadcaster Application of an available target Asset, if the Broadcaster Application is subscribed to the AssetLink Resolution Notification (see Section 9.3.1). The Broadcaster Application is expected to respond by using the AssetLink Resolved API (see Section 9.17.2) to inform the Receiver what action it will take regarding the target Asset.

The AssetLink Resolution Notification semantics are defined in Table 9.122 and the syntax shall be as defined in the schema file org.atsc.notify-assetLinkResolution.json. Additional semantic definitions of parameters follow the table.

Table 9.122 AssetLink Resolution Notification Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`method`	1	string	"org.atsc.notify"
`msgType`	1	enum	"assetLinkResolution"
`assetLink`	1	string (uri)	The `asset_location` from the target Asset
`assetType`	0..1	string	The type of the target Asset.
`assetId`	1	string	The ID value of the target Asset.

`assetLink` – The `asset_location` provided in the target Asset.

`assetType` – This is described in a four-character code ("4CC") type registered in MP4REG (http://www.mp4ra.org) [30], 10.3.9 MP table.

`assetId` – The value of the Asset associated with this `assetLink`.

Once an AssetLink Resolution Notification is sent, the Broadcaster Application may decide to provide alternate content using its own criteria or it may choose not to. The Broadcaster Application is expected to inform the Receiver of its decision using the AssetLink Resolved API described in Section 9.17.2.

For example, on detection of a target Asset, the Receiver could issue the following message, presuming that the target Asset `asset_location` URI is `"http://192.168.32.117:8182/s02gPkwZx14iO"`:

```
<-- {
    "jsonrpc": "2.0",
    "method": "org.atsc.notify",
    "params": {
        "msgType": "assetLinkResolution",
        "assetLink": "http://192.168.32.117:8182/s02gPkwZx14iO",
        "assetType": "hevc",
        "assetId": "550e8400-e29b-41d4-a716-446655440000"
    }
}
```

9.17.2 AssetLink Resolved API

After the AssetLink Resolution Notification (Section 9.17.1) is received, the Broadcaster Application determines what action to take regarding the target Asset and whether to replace the content or do nothing. If it decides that the content can be replaced, it determines which alternate content to replace the default content with and makes a request to the Receiver using the AssetLink Resolved API. The Broadcaster Application provides either a URI of the alternate content or the actual alterative content in the request. The Receiver is expected to process the replacement URI or alternative content text, as appropriate, to replace the target Asset. If the Receiver is not able to replace the default content, the API response is expected to contain the appropriate code and description indicating the reason for the failure.

The AssetLink Resolved Request semantics are defined in Table 9.123 and the syntax shall be defined as in the schema file `org.atsc.assetLinkResolution-request.json`. Additional semantic definitions of parameters follow the table.

Table 9.123 AssetLink Resolved Request Semantics

Property Name	Use	Data Type	Short Description
jsonrpc	1	string	"2.0"
id	1	integer	
method	1	string	"org.atsc.assetLinkResolution"
assetLink	1	string (uri)	The asset_location from the target Asset
assetType	0..1	string	The type of the target Asset.
assetId	1	string	The ID value of the target Asset.
status	1	integer	The status of the Broadcaster Application's action
replacementURL	oneOf X	string (uri)	The URI of a replacement MMT Asset File
replacementText	oneOf X	string	The inline MMT Asset File encoded as Base64 to replace the target Asset

assetLink – The asset_location provided in the target Asset. This value is provided by the assetLink parameter in the AssetLink Resolution Notification described in Section 9.17.1.

assetType – This is described in a four-character code ("4CC") type registered in MP4REG (http://www.mp4ra.org) [30].

assetId – The value of the target Asset associated with the assetLink.

status – The status of the Broadcaster Application reaction to the notification as follows:

0 – Element not replaced, and Receiver is free to act on it

1 – Element replaced. Receiver is to attempt to replace the element

-1 – Element not replaced. Receiver is forbidden from acting on it

replacementURL – If provided as an alternative to the target Asset, this string is expected to provide the URI referencing an MMT Asset File to replace the target Asset. If this property is provided, it is expected to be a fully qualified URI, whether referencing the replacement MMT Asset File over broadband or from the Application Context Cache. For the Application Context Cache, the full URI can be constructed using the Base URI provided using the Query Receiver Web Server URI API as described in Section 9.2.7.

replacementText – Provides the replacement MMT Asset File encoded as a Base64 string.

After receiving the AssetLink Resolved request, the Receiver may respond indicating that the original target Asset was successfully replaced with either the requested assetURL URI or the supplied element. If the Receiver cannot successfully complete the request, an error code is expected to be returned. The Receiver may choose to delay the response until after the element was successfully replaced or may choose to respond with an error condition immediately. Broadcaster Applications are expected to consider the difference in timing when handling the AssetLink Resolved Response. The AssetLink Resolved Response semantics are defined in Table 9.124 and the syntax shall be as defined in the schema file org.atsc.assetLinkResolution-response.json. Additional semantic definitions of parameters follow the table.

Table 9.124 AssetLink Resolved Response Semantics

Property Name	Use	Data Type	Short Description
`jsonrpc`	1	string	"2.0"
`id`	1	integer	Matches the request id value
`result`	oneOf X		Empty object on successful replacement. The error structure is returned if the replacement was unsuccessful.
`error`	oneOf X		See Section 8.3.3

In addition to the errors in Annex B Section 5.1, the following errors from Table 8.5 may be returned:

- -4 – Requested content cannot be found. For example, invalid URL.
- -30 – The request was not received in time to replace the default content.
- -34 – The provided `assetLink`, `assetType` and/or `assetId` not found.
- -35 – The provided `element` content is not valid.

As an example, the Broadcaster Application submits the following request, presuming that the target Asset URI is `"http://192.168.32.117:8182/s02gPkwZx14iO"`:

```
--> {
    "jsonrpc": "2.0",
    "method": "org.atsc.assetLinkResolution",
    "params": {
         "assetLink": "http://192.168.32.117:8182/s02gPkwZx14iO",
         "assetType": "hevc",
         "assetId": "550e8400-e29b-41d4-a716-446655440000",
         "status": 1,
         "replacementURL": "/replacement-ad.mpt",
    },
    "id": 104
}
```

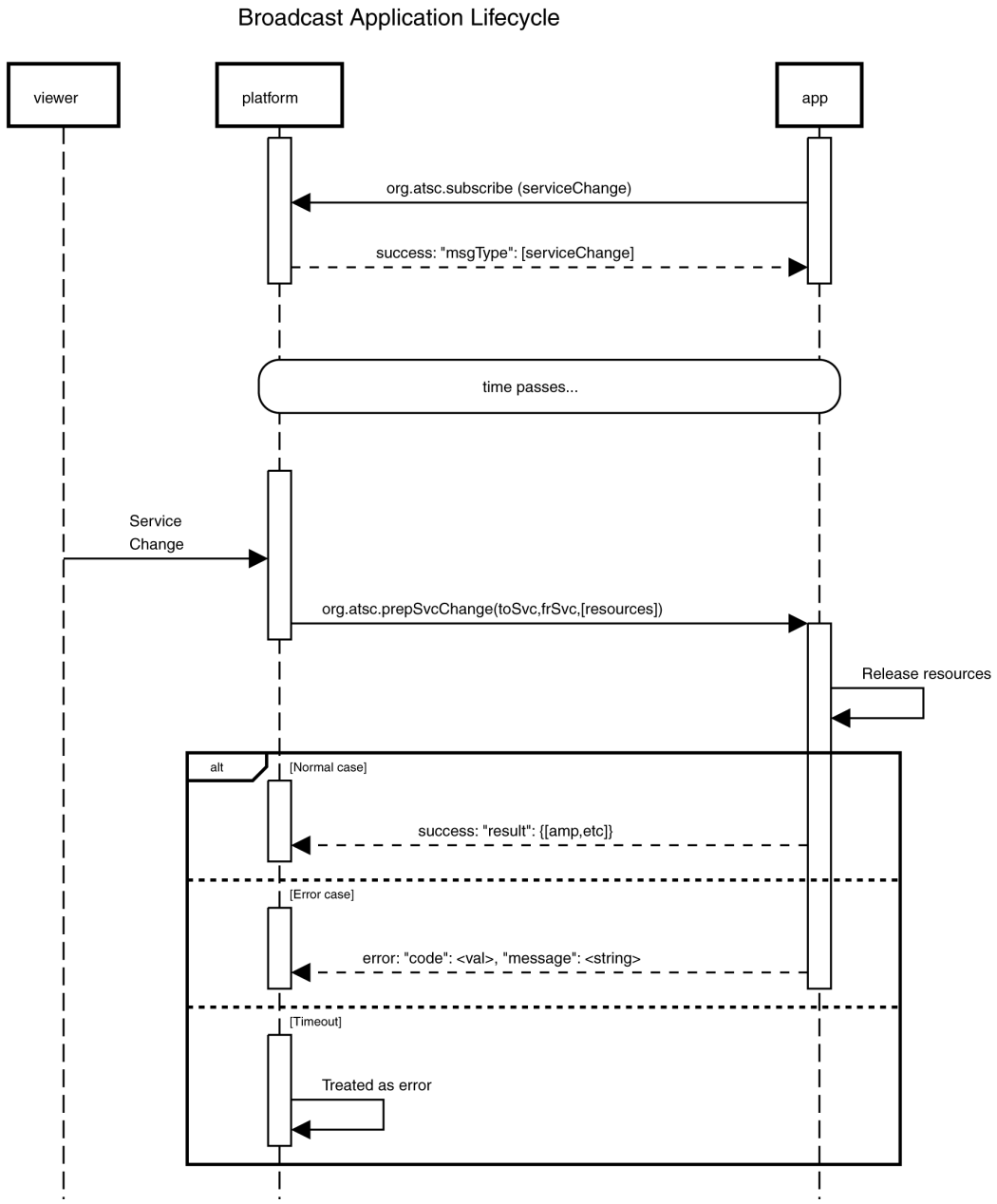
If the Receiver can replace the target Asset with the alternate MMT Asset File referred to by the URI, it responds with a successful result as shown below:

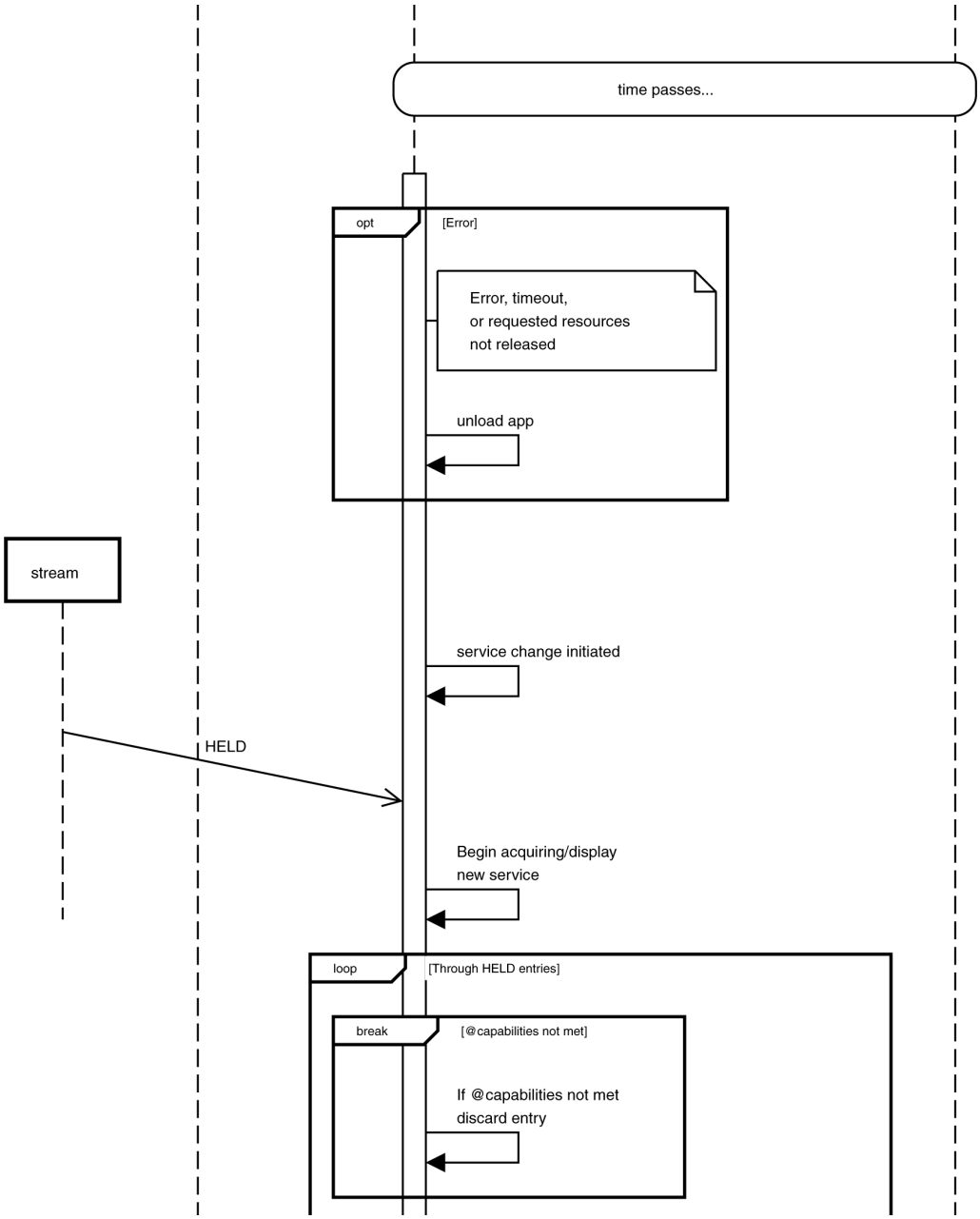
```
<-- {
    "jsonrpc": "2.0",
    "result": {},
    "id": 104
}
```

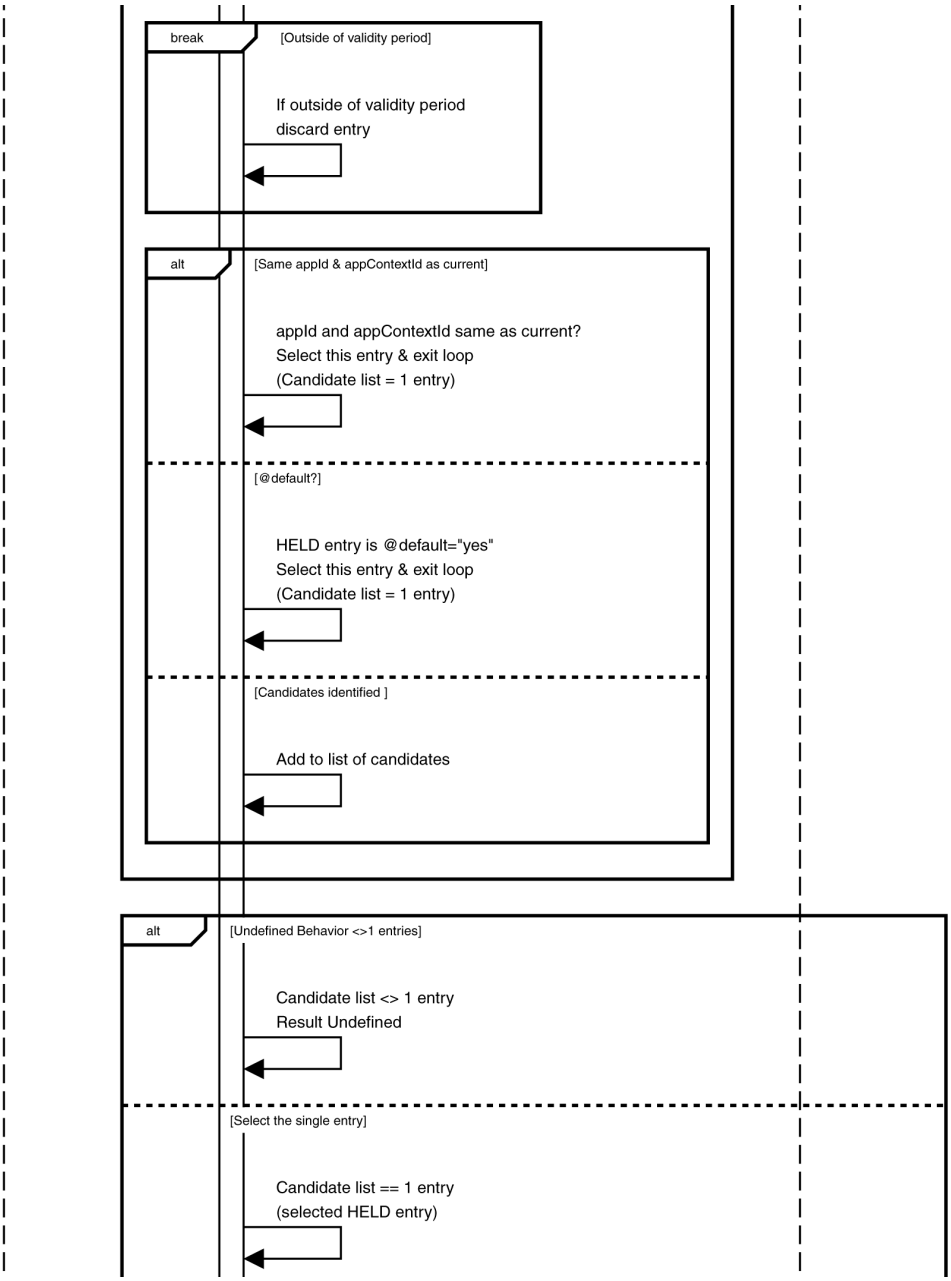
Alternatively, if the Receiver could not use the replacement MMT Asset File referenced by the URI, as requested by the Broadcaster Application, it returns a disposition indicating the reason for the failed request.

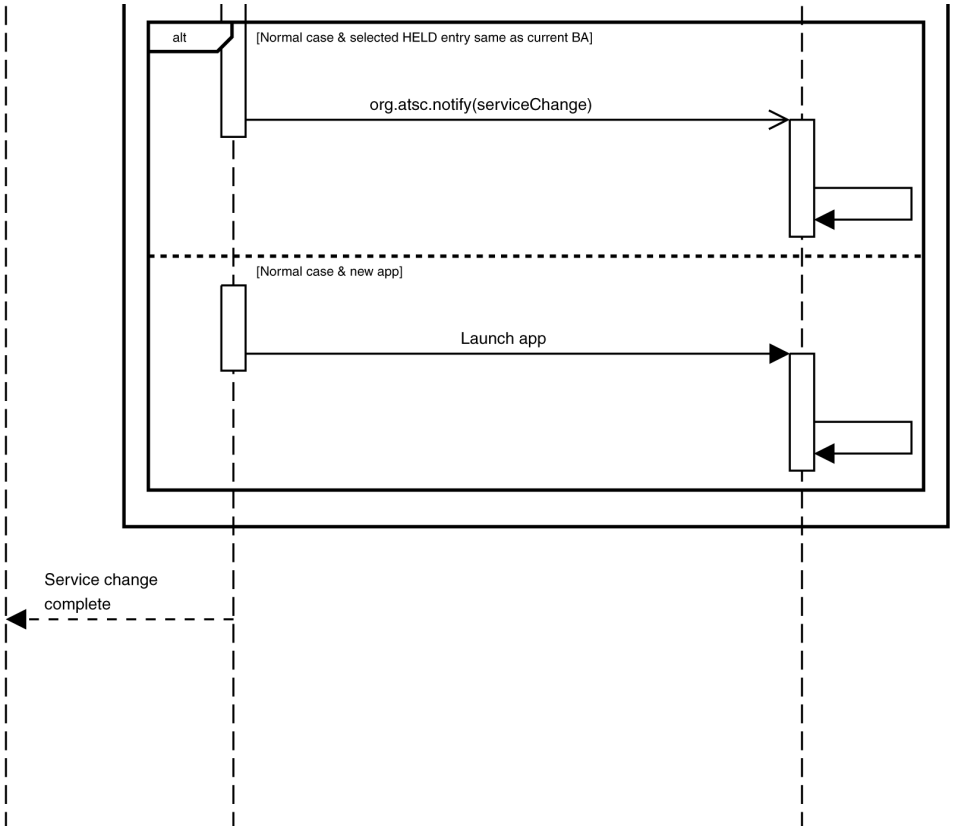
```
<-- {
    "jsonrpc": "2.0",
    "error": {
        "code": -30,
        "message": "Too late"
    },
    "id": 104
}
```

Annex A: Application Lifecycle Sequence Diagram




time passes...
opt
[Error]
Error, timeout,
or requested resources
not released
unload app
stream
service change initiated
HELD
Begin acquiring/display
new service
loop
[Through HELD entries]
break
[@capabilities not met]
If @capabilities not met
discard entry


break
[Outside of validity period]
If outside of validity period
discard entry
alt
[Same appId & appContextId as current]
appId and appContextId same as current?
Select this entry & exit loop
(Candidate list = 1 entry)
[@default?]
HELD entry is @default="yes"
Select this entry & exit loop
(Candidate list = 1 entry)
[Candidates identified]
Add to list of candidates
alt
[Undefined Behavior <>1 entries]
Candidate list <> 1 entry
Result Undefined
[Select the single entry]
Candidate list == 1 entry
(selected HELD entry)


alt
[Normal case & selected HELD entry same as current BA]
org.atsc.notify(serviceChange)
[Normal case & new app]
Launch app
Service change
complete

Annex B: JSON-RPC 2.0 Specification

The contents of this Annex were copied verbatim from http://www.jsonrpc.org/specification [46] on September 28, 2017 and cannot be modified.

Origin Date:
2010-03-26 (based on the 2009-05-24 version)
Updated:
2013-01-04
Author:
JSON-RPC Working Group <json-rpc@googlegroups.com>

1 Overview

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

It is designed to be simple!

2 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Since JSON-RPC utilizes JSON, it has the same type system (see http://www.json.org or RFC 4627). JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays). The term "Primitive" in this specification references any of those four primitive JSON types. The term "Structured" references either of the structured JSON types. Whenever this document refers to any JSON type, the first letter is always capitalized: Object, Array, String, Number, Boolean, Null. True and False are also capitalized.

All member names exchanged between the Client and the Server that are considered for matching of any kind should be considered to be case-sensitive. The terms function, method, and procedure can be assumed to be interchangeable.

The Client is defined as the origin of Request objects and the handler of Response objects. The Server is defined as the origin of Response objects and the handler of Request objects.

One implementation of this specification could easily fill both of those roles, even at the same time, to other different clients or the same client. This specification does not address that layer of complexity.

3 Compatibility

JSON-RPC 2.0 Request objects and Response objects may not work with existing JSON-RPC 1.0 clients or servers. However, it is easy to distinguish between the two versions as 2.0 always has a member named "jsonrpc" with a String value of "2.0" whereas 1.0 does not. Most 2.0 implementations should consider trying to handle 1.0 objects, even if not the peer-to-peer and class hinting aspects of 1.0.

4 Request object

A rpc call is represented by sending a Request object to a Server. The Request object has the following members:

jsonrpc

A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

method

A String containing the name of the method to be invoked. Method names that begin with the word rpc followed by a period character (U+002E or ASCII 46) are reserved for rpc-internal methods and extensions and MUST NOT be used for anything else.

params

A Structured value that holds the parameter values to be used during the invocation of the method. This member MAY be omitted.

id

An identifier established by the Client that MUST contain a String, Number, or NULL value if included. If it is not included it is assumed to be a notification. The value SHOULD normally not be Null[1] and Numbers SHOULD NOT contain fractional parts[2]

The Server MUST reply with the same value in the Response object if included. This member is used to correlate the context between the two objects.

4.1 Notification

A Notification is a Request object without an "id" member. A Request object that is a Notification signifies the Client's lack of interest in the corresponding Response object, and as such no Response object needs to be returned to the client. The Server MUST NOT reply to a Notification, including those that are within a batch request.

Notifications are not confirmable by definition, since they do not have a Response object to be returned. As such, the Client would not be aware of any errors (like e.g., "Invalid params","Internal error").

4.2 Parameter Structures

[1] The use of Null as a value for the id member in a Request object is discouraged, because this specification uses a value of Null for Responses with an unknown id. Also, because JSON-RPC 1.0 uses an id value of Null for Notifications this could cause confusion in handling.

[2] Fractional parts may be problematic, since many decimal fractions cannot be represented exactly as binary fractions.

If present, parameters for the rpc call MUST be provided as a Structured value. Either by-position through an Array or by-name through an Object.

- by-position: params MUST be an Array, containing the values in the Server expected order.
- by-name: params MUST be an Object, with member names that match the Server expected parameter names. The absence of expected names MAY result in an error being generated. The names MUST match exactly, including case, to the method's expected parameters.

5 Response object

When a rpc call is made, the Server MUST reply with a Response, except for in the case of Notifications. The Response is expressed as a single JSON Object, with the following members:

jsonrpc

A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

result

This member is REQUIRED on success. This member MUST NOT exist if there was an error invoking the method. The value of this member is determined by the method invoked on the Server.

error

This member is REQUIRED on error. This member MUST NOT exist if there was no error triggered during invocation. The value for this member MUST be an Object as defined in section 5.1.

id

This member is REQUIRED. It MUST be the same as the value of the id member in the Request Object. If there was an error in detecting the id in the Request object (e.g. Parse error/Invalid Request), MUST be Null. Either the result member or error member MUST be included, but both members MUST NOT be included.

5.1 Error object

When a rpc call encounters an error, the Response Object MUST contain the error member with a value that is a Object with the following members:

code

A Number that indicates the error type that occurred. This MUST be an integer.

message

A String providing a short description of the error. The message SHOULD be limited to a concise single sentence.

data

A Primitive or Structured value that contains additional information about the error. This may be omitted. The value of this member is defined by the Server (e.g. detailed error information, nested errors etc.).

The error codes from and including -32768 to -32000 are reserved for pre-defined errors. Any code within this range, but not defined explicitly below is reserved for future use. The error codes

are nearly the same as those suggested for XML-RPC at the following url: http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php

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.

The remainder of the space is available for application defined errors.

6 Batch

To send several Request objects at the same time, the Client MAY send an Array filled with Request objects.

The Server should respond with an Array containing the corresponding Response objects, after all of the batch Request objects have been processed. A Response object SHOULD exist for each Request object, except that there SHOULD NOT be any Response objects for notifications. The Server MAY process a batch rpc call as a set of concurrent tasks, processing them in any order and with any width of parallelism.

The Response objects being returned from a batch call MAY be returned in any order within the Array. The Client SHOULD match contexts between the set of Request objects and the resulting set of Response objects based on the id member within each Object.

If the batch rpc call itself fails to be recognized as an valid JSON or as an Array with at least one value, the response from the Server MUST be a single Response object. If there are no Response objects contained within the Response array as it is to be sent to the client, the server MUST NOT return an empty Array and should return nothing at all.

7 Examples

Syntax:

```
--> data sent to Server
<-- data sent to Client
```

rpc call with positional parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
```

```
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}
```

rpc call with named parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend":
42}, "id": 3}

<-- {"jsonrpc": "2.0", "result": 19, "id": 3}



--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend":
23}, "id": 4}

<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```

a Notification:

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

rpc call of non-existent method:

```
--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id":
"1"}
```

rpc call with invalid JSON:

```
--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id":
null}
```

rpc call with invalid Request object:

```
--> {"jsonrpc": "2.0", "method": 1, "params": "bar"}
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null}
```

rpc call Batch, invalid JSON:

```
--> [

  {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},

  {"jsonrpc": "2.0", "method"

]

<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"}, "id":
null}
```

rpc call with an empty Array:

```
--> []

<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null}
```

rpc call with an invalid Batch (but not empty):

```
--> [1]
<-- [
```

```
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null}

]
```

rpc call with invalid Batch:

```
--> [1,2,3]
<-- [

  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null},

  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null},

  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"}, "id":
null}

]
```

rpc call Batch:

```
--> [
        {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]},
        {"jsonrpc": "2.0", "method": "subtract", "params": [42,23], "id": "2"},
        {"foo": "boo"},
        {"jsonrpc": "2.0", "method": "foo.get", "params": {"name": "myself"}, "id":
"5"},
        {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
    ]
<-- [
        {"jsonrpc": "2.0", "result": 7, "id": "1"},
        {"jsonrpc": "2.0", "result": 19, "id": "2"},
        {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request"},
"id": null},
        {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"},
"id": "5"},
        {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
    ]
```

rpc call Batch (all notifications):

```
--> [
        {"jsonrpc": "2.0", "method": "notify_sum", "params": [1,2,4]},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]}
    ]
<-- //Nothing is returned for all notification batches
```

8 Extensions

Method names that begin with rpc. are reserved for system extensions, and MUST NOT be used for anything else. Each system extension is defined in a related specification. All system extensions are OPTIONAL.

— End of Document —