OIPF

Release 2 Specification

Volume 5 - Declarative Application

[V2.0] – [2010-09-07]

Open IPTV Forum

Open IPTV Forum

Postal address

Open IPTV Forum support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 43 83
Fax: +33 4 92 38 52 90

Internet
http://www.oipf.tv

Disclaimer

The Open IPTV Forum members accept no liability whatsoever for any use of this document.

This specification provides multiple options for some features. The Open IPTV Forum Profiles specification complements the Release 2 specifications by defining the Open IPTV Forum implementation and deployment profiles. Any implementation based on Open IPTV Forum specifications that does not follow the Profiles specification cannot claim Open IPTV Forum compliance.

Copyright Notification

Contents

Figures

Tables

Foreword

This Technical Specification (TS) has been produced by Open IPTV Forum.

This specification provides multiple options for some features. The Open IPTV Forum Profiles specification complements the Release 2 specifications by defining the Open IPTV Forum implementation and deployment profiles. Any implementation based on Open IPTV Forum specifications that does not follow the Profiles specification cannot claim Open IPTV Forum compliance.

Introduction

The Open IPTV Forum Release 2 Specification consists of nine Volumes:

- Volume 1 - Overview,
- Volume 2 - Media Formats,
- Volume 2a – HTTP Adaptive Streaming
- Volume 3 - Content Metadata,
- Volume 4 - Protocols,
- Volume 4a – Examples of IPTV Protocol Sequences,
- Volume 5 - Declarative Application Environment,
- Volume 6 - Procedural Application Environment, and
- Volume 7 - Authentication, Content Protection and Service Protection.

The present document, the Declarative Application Environment Specification (Volume 5), specifies the DAE functionality of the Open IPTV Forum Release 2 Solution.

1 Scope

The Open IPTV Forum has developed an end-to-end solution to allow any consumer end-device, compliant to the Open IPTV Forum specifications, to access enriched and personalized IPTV services either in a managed or a non-managed network.

Its functional architecture specification [OIPF_ARCH2] defines a block called OITF which resides inside the residential network. The OITF includes the functionality required to access IPTV services for both the unmanaged and the managed network.

Part of these functionalities is the **Declarative Application Environment (DAE)**: a declarative language based environment (browser) based on CEA-2014 [CEA-2014-A] for presentation of user interfaces and including scripting support for interaction with network server-side applications and access to the APIs of the other OITF functions.

The DAE is the focus of this specification.

The requirements for specifying this functionality are derived from the following sources:

- Open IPTV Service and Platform Requirement for R2 [OIPF_REQS2];
- Open IPTV Functional Architecture for R2 [OIPF_ARCH2].

2 References

2.1 Normative references

[3GPP TS 24.229]	3GPP, TS 24.229, “IP Multimedia Call Control Protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Stage 3 (Release 8)”
[CEA-2014-A]	CEA, CEA-2014-A, (Including the August 2008 Errata) “Web-based Protocol Framework for Remote User Interface on UPnP Networks and the Internet (Web4CE)”,
[TS 102 539]	ETSI TS 102 539, “Digital Video Broadcasting (DVB); Carriage of Broadband Content Guide (BCG) information over Internet Protocol (IP)
[TS 102 809]	ETSI TS 102 809 “Digital Video Broadcasting (DVB); Signalling and carriage of interactive applications and services in Hybrid broadcast/broadband environments”
[TS 102 851]	ETSI TS 102 851, “Digital Video Broadcasting (DVB); Uniform Resource Identifiers (URI) for DVB Systems”
[DVB-IPTV]	ETSI TS 102 034.V1.4.1, “DVB-IPTV 1.3: Transport of MPEG-2 TS Based DVB Services over IP Based Networks (and associated XML)”
[EN 300 468]	ETSI EN 300 469, “Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB Systems”
[TISPAN]	ETSI TS 183 063, “Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN);IMS-based IPTV stage 3 specification”
[IEC62455]	IEC, IEC 62455, “Internet protocol (IP) and transport stream (TS) based service access”
[RFC1321]	IETF, RFC 1321, “The MD5 Message-Digest Algorithm”, April 1992.
[RFC2109]	IETF, RFC 2109, “HTTP State Management Mechanism”, February 1997.
[RFC2119]	IETF, RFC 2119, “Key words for use in RFCs to Indicate Requirement Levels”, March 1997.
[RFC2326]	IETF, RFC 2326, “Real Time Streaming Protocol (RTSP)”, April 1998.
[RFC2616]	IETF, RFC 2616, “Hypertext Transfer Protocol -- HTTP/1.1”, June 1999.
[RFC3550]	IETF, RFC 3550, “RTP: A Transport Protocol for Real-Time Applications”, July 2003.
[RFC3840]	IETF, RFC 3840, “Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)”, August 2004.
[RFC3841]	IETF, RFC 3841, “Caller Preferences for the Session Initiation Protocol (SIP)”, August 2004.
[MPEG-7]	ISO/IEC 15938-5, “Multimedia Content Description Interface - Part 5:Multimedia description schemes”, , May 2003”
[JFIF]	JPEG File Interchange Format, Version 1.02, Eric Hamilton, C-Cube Microsystems, September 1, 1992
[PRES]	OMA, OMA-TS-Presence_SIMPLE_XDM-V1_1-20080627-A, “Presence XDM Specification”
[IM]	OMA, OMA-TS-SIMPLE_IM-V1_0-20080820-D, “Instant Messaging using SIMPLE”.
[CSS3 UI]	W3C, “CSS3 Basic User Interface Module”, May 2004.
[CSS3 BG]	W3C, “CSS Backgrounds and Borders Module Level 3”, Working Draft 10 September 2008.
[DOM 2 Core]	W3C, “Document Object Model (DOM) Level 2 Core Specification - Version 1.0”, November 2000
[DOM 2 Events]	W3C, “Document Object Model (DOM) Level 2 Events Specification - Version 1.0”, November 2000
[DOM 2 HTML]	W3C, “Document Object Model (DOM) Level 2 HTML Specification - Version 1.0”, January 2003
[DOM 2 Views]	W3C, “Document Object Model (DOM) Level 2 Views Specification - Version 1.0”, November 2000
[DOM 3 Events]	W3C, “Document Object Model (DOM) Level 3 Events Specification - Version 1.0”, December 2007
[HTML5]	W3C, “HTML5 - A vocabulary and associated APIs for HTML and XHTML, Working Draft 25 August 2009”
[SVG Tiny 1.2]	W3C, “Scalable Vector Graphics (SVG) Tiny 1.2 Specification”, August 2006
[Web-Storage]	W3C, “Web-Storage”, Last Call Working Draft 22 December 2009

[Widgets-Access]	W3C, “Widgets 1.0: Access Requests Policy”, Last Call Working Draft, 8 December 2009
[Widgets-APIs]	W3C, “Widgets 1.0: Widget Interface”, Candidate Recommendation, 22 December 2009
[Widgets-DigSig]	W3C, “Widgets 1.0: Digital Signature”, Candidate Recommendation, 25 June 2009
[Widgets-Packaging]	W3C, “Widgets 1.0: Packaging and Configuration”, Candidate Recommendation, 1 December 2009
[Window Object]	W3C, “Window Object 1.0”, April 2006
[XHR]	W3C, "The XMLHttpRequest Object", April 2008
[DLNA]	DLNA Networked Device Interoperability Guidelines, August 2009

2.2 Open IPTV Forum references

[OIPF_SERV2]	Open IPTV Forum, “Services and Functions for Release 2”, V1.0, October 2008.
[OIPF_REQS2]	Open IPTV Forum, “Open IPTV Forum Service and Platform Requirements”, V2.0, December 2008.
[OIPF_ARCH2]	Open IPTV Forum, “Open IPTV Forum, Functional Architecture – V2.0”, September 2009.
[OIPF_MEDIA2]	Open IPTV Forum, “Release 2 Specification, Volume 2 - Media Formats”, V2.0, September 2010.
[OIPF_HAS2]	Open IPTV Forum, “Release 2 Specification, Volume 2a – HTTP Adaptive Streaming”, V2.0, September 2010.
[OIPF_META2]	Open IPTV Forum, “Release 2 Specification, Volume 3 – Content Metadata”, V2.0, September 2010.
[OIPF_PROT2]	Open IPTV Forum, “Release 2 Specification, Volume 4 – Protocols”, V2.0, September 2010.
[OIPF_PROT2_EX]	Open IPTV Forum, “Release 2 Specification, Volume 4a – Examples of IPTV Protocol Sequences”, V2.0, September 2010.
[OIPF_PAE2]	Open IPTV Forum, “Release 2 Specification, Volume 6 - Procedural Application Environment”, V2.0, September 2010.
[OIPF_CSP2]	Open IPTV Forum, “Release 2 Specification, Volume 7 - Authentication, Content Protection and Service Protection”, V2.0, September 2010.

3 Terminology and conventions

3.1 Conventions

All sections and annexes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

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 [RFC2119].

In sections of the present document whose presence is indicated by one of the capabilities defined in Section 9.3, use of the [RFC2119] terms “MUST”, “SHALL” or “REQUIRED” applies only when the capability is made available to DAE applications. They do not have the effect of making that section mandatory.

In this document, “application” means “declarative application” (browser based application) throughout the DAE specification, as opposed to the “procedural applications” (Java based applications) defined in the PAE platform specification.

In the documented APIs ECMAScript attributes are read-write unless otherwise specified.

The type “`Integer`” is not a valid Javascript type as is. It is used as a short hand notation for a subset of type “`Number`” which includes only the numbers that can be written without a fractional or decimal component.

3.2 Definitions

Term	***Definition***
Audio from memory	Audible notifications and audio clips intended to be played from memory.
Broadcast related application	Interactive application associated with a television or radio channel, with part of a television channel (e.g. a particular program or show) or other television content. Often referred to as “red button” applications in the industry, regardless of how they are actually started by the end user.
Broadcast independent application	Interactive application not related to any TV channel or TV content or to the currently selected service provider.
Control UI	The Remote UI that controls DAE applications in the OITF, sent from an IPTV Applications server via the OITF or pre-stored in the OITF, and rendered in the DLNA RUIC on the Remote Control Device.
DLNA RUIC	A DLNA device with the role of finding and loading remote UI content exposed by a DLNA RUIS capability and rendering and interacting with the UI content. Note: This terminology references the DLNA RUI specification.
DLNA RUIS	A DLNA Function in the OITF with the role of exposing and sourcing UI content. Note: This terminology references the DLNA RUI specification.
Embedded object	A software module that extends the capabilities of the OITF browser. Features provided by an embedded object are made available to DAE applications through the methods and properties of a specific javascript object.
HTML document	An XHTML document and associated style and script files conforming to the restrictions and extensions defined in the present document.
Key Event	Event sent to a DAE application in response to input from the end-user. This input is typically generated in response to the end-user pressing a button on a conventional remote control. It may also be generated by some other mechanism on alternative input devices such as game controllers, touch screens, wands or drastically reduced remote controls.
Mandatory	The feature is an absolute requirement of the specification (a “MUST” as defined by RFC 2119).
Non-visual embedded object	A non-visual embedded object is an embedded object that has no visible representation and cannot get input focus
Optional	The feature is truly optional (a “MAY” as defined by RFC 2119).
Remote Control Device	A mobile or portable device which has the functionality of the DLNA RUIC.

Remote UI	The display of a UI from one device on a second (remote) device across a network.
Service provider related application	Interactive application related to the service provider selected through the service provider selection process.
Trick Mode	Facility to allow the User to control the playback of Content, such as pause, fast and slow playback, reverse playback, instant access, replay, forward and reverse skipping.

3.3 Abbreviations

In addition to the Abbreviations provided in Volume 1, the following abbreviations are used in this volume.

Abbreviation	*Definition*
AJAX	Asynchronous JavaScript and XML
CRID	Content Reference Identifier
CSS	Cascading style sheets
DOM	Document object model
GIF	Graphics Interchange Format
HE-AAC	High Efficiency AAC
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PSI	Public Service Identifier
RCF	Remote Control Function
SVG	Scalable Vector Graphics
TLS	Transport Layer Security
WAVE	Waveform audio format

4 DAE overview

This specification builds on the capability model defined in CEA-2014 [CEA-2014-A] in order to expose to an IPTV service provider the capabilities of any particular OITF.

In addition to what is defined in CEA-2014, other terminal capabilities are defined in Section 9.3 covering most of the features defined in this specification. This document does not define whether these capabilities are mandatory or not. Other documents or specifications need to address that. A small minimum set of capabilities are defined in Section 9.2.

Section 3.1 of this document defines how to interpret [RFC2119] terms like "SHALL" in sections of this document included in a capability. In sections of this document which are not covered by capabilities, terms like "SHALL" apply as used in each section.

4.1 Architecture of DAE

This section will introduce the basic concepts in the architecture of the DAE specification and their relationships. [CEA-2014-A] is the baseline technology for the DAE. In particular the following requirements hold:

- The OITF SHALL support the i-Box model as defined in [CEA-2014-A] with the changes described in Annex B of this document, in particular all requirements for an i-Box remote UI client as defined in Section 5.1.2, Sections 5.2 through 5.8 and Section 5.10 of CEA-2014-A (i.e. all Remote UI client requirements inside the subsections that are marked as either “Mandatory for every RUIC” or “Mandatory for i-Box” except where modified by Annex B of this document). This also includes (through reference) Annexes C, F, G, H, I of [CEA-2014-A]. The OITF SHALL also support the following features which are not mandatory for the i-box model.
  - 5.6.1 Multicast notifications
  - 5.7.1 Streamed A/V Content
  - 5.7.3 Full-screen video
- The OITF MAY support the 2-box and/or 3-box models defined in [CEA-2014-A]. Note that by default the interface with the AG and IG deviates from CEA-2014’s 2-box model and 3-box model. An overview of these differences is given in Section 4.1.1.
- A mandatory requirement in CEA-2014-A remains mandatory for the OITF, and recommended and optional requirements in CEA-2014-A remain recommended and optional for the OITF, unless explicitly specified differently inside this DAE specification. A detailed description of these differences can be found in Annex B.
- In case of a conflict between a CEA-2014 requirement and a normative statement in the DAE specification, the normative statement in the DAE specification SHALL have priority.

4.1.1 Remote UI and box models (Informative)

The architecture overview from Section 4.1 of [CEA-2014-A] defines various box models. Next to the i-Box model for accessing IPTV service providers or 3rd party internet services, it defines a 2-Box and 3-box model for in-home remote UI. Box Models are divided by not only where the server resides but also where the UI control point reside to perform discovery and setup of a remote UI connection. In case of the 2-Box and 3-box model the UI control point is a UPnP control point that discovers in-home servers. In case of the 2-box model, there is a UPnP Remote UI control point inside the OITF. If the UPnP remote UI control point resides in an external device (e.g. web pad, remote controller), whereby the external device lists the Remote UI servers and sets up a UI connection between the OITF and Remote UI Server this is called the 3-box model. An OITF that supports the 3-box model must be discoverable through UPnP itself, and expose the profile information of a Remote UI client to the home network.

For the OITF, only the CEA-2014-A i-Box model is mandatory. The 2-box and 3-box models are optional. The default interaction with the Application Gateway (AG), the IMS Gateway (IG) and the CSP gateway (CSPG) deviate in the following manner. However, it is not precluded for an AG, IG, CSPG or other devices in the home network to expose themselves as a regular UPnP Remote UI server that is compliant with CEA-2014, for example to serve a Remote UI of its configuration screen to the OITF.

- The AG is similar to a level 1 remote UI server as defined in Section 5.1.1.2 of [CEA-2014-A], with the difference that [Req. 5.1.1.2.d] is replaced with a different device description. The device description of the AG is defined in Section 10.1.1.2 of [OIPF_PROT2]. The requirements [Req. 5.1.1.2.b] and [Req. 5.1.1.2.c] are now optional: a URL to the XML UI Listing is provided by element <agUIServerURL> of the AG Description XML document. Note that

the UPnP Device description of the AG MAY offer a CEA-2014-A compatible level 1 or level 2 remote UI server in its UPnP device hierarchy that point to the same XML UI listing.

- The IG enables the discovery of IPTV services through the HNI-IGI interface as defined in [OIPF_PROT2]. This is quite different from a level 1 or level 2 remote UI server. The details of the device discovery of the IG are defined in Section 10.1.1.1 of [OIPF_PROT2].

Irrespective of the box models, and the discovery mechanism used, the OITF performs the following general steps to set up a connection to any internet or in-home service:

1) Setup & Connect phase:

   a) The OITF connects to a URL of a DAE application offered by a server over an HTTP connection. The OITF's capability profile is conveyed to the server, using the "User-Agent" HTTP header, to enable the server to adjust the contents to the DAE capabilities of the OITF. An OITF that supports additional content formats (e.g. Flash) can also convey these extensions to the server.

   b) After setting up the connection, the XHTML and/or SVG contents that constitute the DAE application are downloaded to the OITF.

   c) This connection can also be set up by a separate UI Control Point in case of an OITF that supports a 3-box model.

2) Presenting web content:

   a) After downloading the XHTML and/or SVG contents, the DAE application may become active and display a user interface as defined by the XHTML and/or SVG contents.

3) Controlling the UI:

   a) Remote control, keyboard and mouse events can be handled within scripts.

   b) Native control for web forms and spatial navigation must be supported.

   c) Client-side scripting control for the playback of A/V content must be supported.

4) Dynamic UI Updates:

   a) User interfaces can be dynamically updated by the server using a persistent TCP connection (NotifSocket) or through XML updates over an HTTP connection (AJAX).

5) 3rd Party Notifications:

   a) Notification messages linked to UI content can arrive on the OITF outside of an active UI interaction between the OITF and the server.

4.1.1.1 i-Box model

The i-Box Model supports the remote presentation and control of UIs that reside on a server on the Internet (WAN). The client (OITF) resides within the home domain, and is either non-discoverable and has a built-in "Connection setup and control" to perform connection management related operations, or is discoverable by an external so called UI Control Point within the home domain that allow the connection management related operations to be controlled by another device. This configuration is depicted in the diagram below.

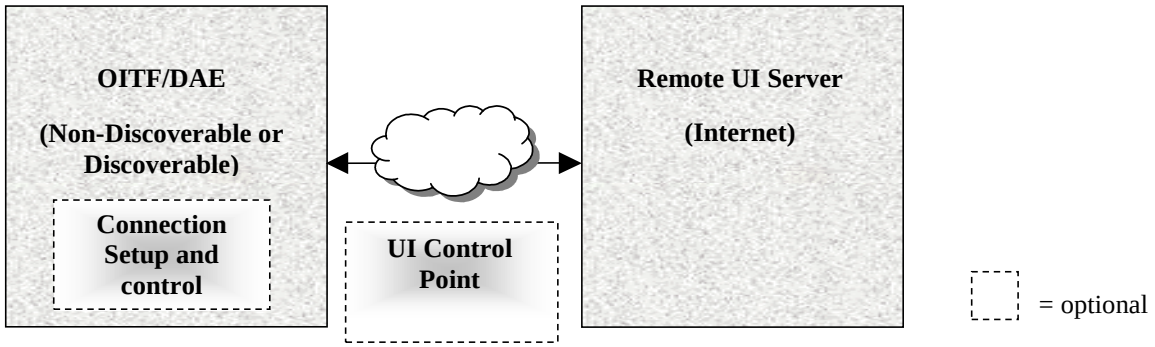


Figure 1: i-Box Model

4.1.1.2 2-Box model

The 2-Box Model describes a configuration in which the server is discoverable in the home network. Since the client is not discoverable, it must have a UI Control Point in order to be functional in the network to be able to discover an AG device description (as defined in Section 10 of [OIPF_PROT2]), or a Remote UI server description as described in Section 5.1 of [CEA-2014-A].

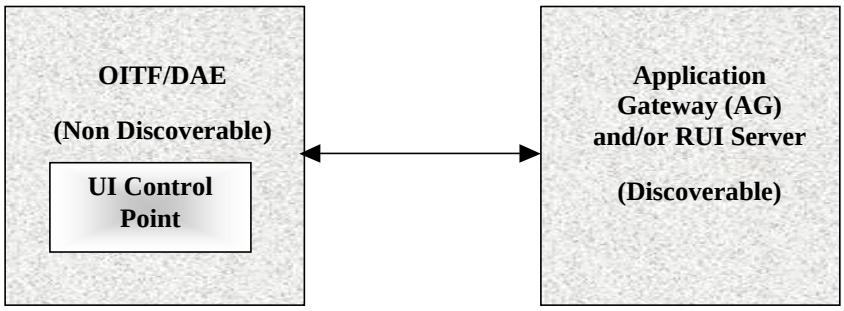


Figure 2: 2-Box Model

4.1.1.3 3-Box model

When both the Remote UI Server and the Remote UI Client are discoverable, the configuration can be described by the 3-Box UI Model. This configuration has no restriction on the location of the UI Control Point for the discovery and connection management, as illustrated in the diagram below.

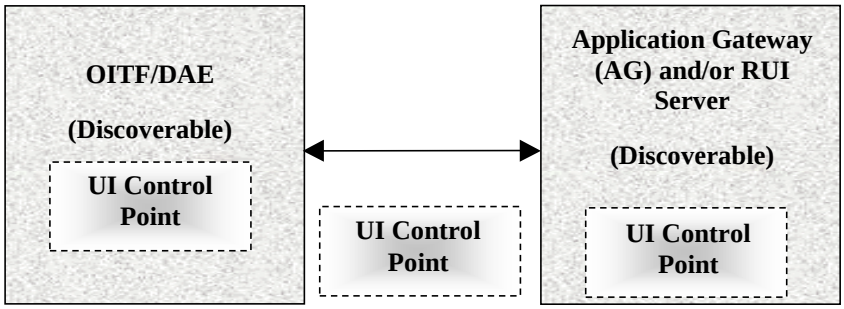


Figure 3: 3-box Model

4.2 Gateway discovery and control

This section describes how DAE applications discover the information of the gateway and subsequently interacts with the gateway. The discovery of the IG and AG by the OITF are defined in Section 10.1 of [OIPF_PROT2]. The discovery takes place prior to the DAE application being initialized. The information about the discovered gateways is made available to DAE applications through the `application/oipfGatewayInfo` embedded object. DAE applications can use this gateway information to interact with the discovered gateways (e.g. IG, AG, CSP gateway and so on). The `application/oipfGatewayInfo` embedded object SHALL be made accessible through the DOM with the interface as defined in Section 7.7.1.

Access to the functionality of the `application/oipfGatewayInfo` embedded object is privileged and SHALL adhere to the security requirements defined in Section 10.1

4.3 Application definition

This section defines what is meant by the concept of a 'DAE application'; which files and assets are considered to be part of a DAE application and how this relates to DAE application security and lifecycle.

A DAE application is an associated collection of documents (typically ECMAScript, CSS and HTML or SVG documents) from the same fully-qualified domain, unless specified differently in Section 5.1.1.3 and with the exception of Widgets as specified in Section 4.3.9. Whilst the document is loaded within the browser, an additional browser object (the `oipfApplicationManager` object), defined in Section 7.2.1 may be instantiated by a DAE application. The `ApplicationManager` object provides access to the `Application` class defined in Section 7.2.2.

The difference between a DAE application and a traditional web page is that web pages are stand-alone with no formal concept of a group of pages or a context within which a group of pages are loaded and execute. For this reason, the definition and details of a DAE application focuses on the application execution environment and the additional capabilities provided to DAE applications. The next subsections describe some of the differences. Additional details about the DAE application lifecycle can be found in Section 5.1

4.3.1 Similarities between applications and traditional web pages

DAE applications are comprised of pages which are conceptually no different from traditional web pages. Both pages in a DAE application and traditional web pages can include the contents of other documents. These included documents can have a variety of types, including Cascading Style Sheets (CSS), ECMAScript, SVG, JPEG, PNG and GIF.

A dynamic DOM, combined with XMLHttpRequest, permits AJAX-style changes to the current page in a DAE application or web page without necessarily replacing the entire document.

4.3.2 Differences between applications and traditional web pages

A DAE application provides shared context and state common to a number of pages – a concept which doesn't formally exist in the web. Loading and unloading pages within the context of a DAE application is the same as loading and unloading web pages. The application context includes information about the state of an application from the platform's perspective – permissions, priority (for example, which to terminate first in the event of insufficient resources) and similar information that spans all documents within an application during the lifetime of that application.

An OITF MAY support the execution of more than one application simultaneously. Applications MAY share the same screen estate in a defined and controlled fashion. This differs from multiple web pages, which are typically handled through different browser "windows" or "tabs" and may not share the same screen estate concurrently (although the details of this behaviour are often browser-dependent). This also differs from the use of frames, which, apart from iframes, do not support overlapping screen estate. Where simultaneous execution of more than one application is supported, both foreground and background applications SHALL be supported simultaneously.

Where simultaneous execution of more than one application is supported, applications SHALL be recorded within a hierarchy of applications. Each object representing an application possesses an interface that provides access to methods and attributes that are uniquely available to applications. For example, facilities to create and destroy applications can be accessed through such methods.

4.3.3 The application tree

Where simultaneous execution of more than one application is supported, applications are organised into a tree structure. Using the `createApplication()` method as defined in Section 7.2.2.2, applications can either be started as child nodes of the application or as a sibling of the application (i.e. added as an additional child of this application's parent). The root node of an application tree is created upon loading an initial application URI or by creating a sibling of an application tree's root node. An OITF MAY keep track of multiple application trees. Each of these individual application trees are connected to a hidden system root node maintained by the OITF that is not accessible by other applications.

Applications created while the DAE environment is running (e.g. as a result of an external notification) that are not created through `createApplication()` SHALL be created as children of the hidden system root node.

4.3.4 The application display model

Applications SHALL be displayed on the OITF in one of the application visualization modes as defined in Section 4.4.6.

The mode used SHALL be determined prior to initialisation of the DAE execution environment and shall persist until termination or re-initialization of the DAE execution environment. The means by which this mode is chosen is outside the scope of this specification.

Each application has at least one associated DOM `Window` object and DOM `Document` object that represents the document or documents that are currently loaded for that application. Even "windowless" applications that are never made visible have an associated DOM `Window` object.

4.3.4.1 Manipulating an application's DOM Window object

Standard DOM `Window` methods are used to resize, scroll, position and access the application document (see Section 4.4.6). Many browsers restrict the size or location of windows; these restrictions SHALL NOT be enforced for windows associated with applications within the browser area. Any area of the display available to DAE applications may be used by any application. Thus, 'Widget'-style applications can create a small window that contains only the application without needing to be concerned with any minimum size restrictions enforced by browsers.

4.3.5 The security model

Each application has a set of permissions to perform various privileged operations within the OITF. The permissions that are granted to an application are defined by the intersection of three permission sets:

1) The permissions requested by the application, using the mechanism defined in Section 10.

2) The permissions supported by the OITF. Some permissions may not be supported due to capability restrictions (e.g. the *permission_pvr* permission will never be granted on a receiver that does not support PVR capability).

3) The permissions that may be granted, as determined by user settings or configuration settings specified by the operator (e.g. blacklists or whitelists; see Section 10 for more information). This is a subset of (2), and may be different for different users.

4.3.6 Inheritance of permissions

Applications created by other applications (e.g. using the methods described in Sections 5.1.1.2 or 5.1.1.3) SHALL NOT inherit the permissions issued to the parent application. The permissions granted to the new application will be defined by the mechanism specified in Section 10.

When an application uses cross-document messaging as defined in [HTML5] to communicate with another application, any action carried out in response to the message SHALL take place in the security context of the application to which the message was sent. Applications SHOULD take care to ensure that privileged actions are only taken in response to messages from an appropriate source.

4.3.7 Privileged application APIs

The privilege model implemented with applications is based upon requiring access to the `Application` object representing an application in order to access the privileged functionality related to application lifecycle management and inter-application communication.

4.3.7.1 Compromising the security

Since applications have access to `Application` objects, it is possible for applications to compromise the security of the framework by passing these objects to untrusted code. For example, an application could raise an event on an untrusted document and pass a reference to its `Application` object in the message. Where simultaneous execution of more than one application is supported, any calls to methods on an `Application` object from pages not running as part of an application from the same provider SHALL throw an error as defined in Section 10.1.1.

4.3.8 Active applications list

Where simultaneous execution of more than one application is supported, the OITF SHALL maintain a list of application nodes ordered in a "most recently activated" order – the active applications list. This list is used by the cross-application event dispatch algorithm as defined in Section 4.4.7 and is not directly visible to applications.

An application is activated through calling the `activateInput()` method of the application node. This marks an application as active and SHALL insert the application at the start of the active application list (removing it from the list first if it is already present).

An application is deactivated through the `deactivateInput()` method of the application node. This marks an application inactive and SHALL remove it from the active application list.

The currently active application is the application at the start of the active application list.

This specification does not define any behaviour if more than one copy of the browser is executing.

4.3.9 Widgets

DAE Widgets are a specialization of DAE applications as defined in Section 4.3 of this document and share aspects with W3C Widgets.

W3C Widgets are standardized by the "Widgets 1.0 family of specifications" as described in Section 1.4 of [Widgets-Packaging]. Section 11 of this document specifies which parts of W3C Widgets specifications are in supported by DAE Widgets. From here on, when using the word "Widget" we will refer to DAE Widgets as defined in this specification.

Widgets can be primarily seen as packaged DAE applications. Since they are packaged, it is possible to have a single download and installation on an OITF. Widgets may also be installed on an OITF via non-HTTP distribution channels and even over off-network channels (e.g. a USB thumb drive). Packaging also provides an easy way to deploy and/or update applications on the OITF when it is installed in the home. The packaging and configuration of a DAE Widget is described in Section 11.1.

Since DAE Widgets are DAE Applications everything that is defined for a DAE Application is also applicable to a Widget unless specified. Furthermore Widgets have several specific features as defined in Section 11.

4.4 Resource Management

This section describes how resources (including non-granular resources such as memory and display area) are shared between multiple applications that may be running simultaneously. Applications SHOULD be able to tolerate the loss of scarce resources if they are needed by another application, and SHOULD follow current industry best practises in order to minimize the resources they consume.

This specification is silent about the mechanism for sharing resources between DAE applications and other applications running on the OITF. In the remainder of this section and this document, the term application refers solely to DAE applications

4.4.1 Application lifecycle issues

Where simultaneous execution of more than one application is supported, if an application attempts to start and not enough resources are available, the application with the lowest priority MAY be terminated until sufficient resources are available for the new application to execute or until no applications with a lower priority are running. Applications without a priority associated with them (e.g. applications started by the DRM agent, see Section 5.1.1.6) SHALL be assumed to have a priority of 0x7F.

Applications may register a listener for `ApplicationUnloaded` events (see Section 7.2.1.4) to receive notification of the termination of a child application where simultaneous execution of more than one application is supported.

Failure to load an asset (e.g. an image file) or CSS file due to a lack of memory SHALL have no effect on the lifecycle of an application, but may result in visual artefacts (e.g. images not being displayed). Failure to load an HTML file due to a lack of memory MAY cause the application to be terminated.

4.4.2 Caching of application files

Application files MAY be cached on the receiver in order to improve performance; this specification is silent about the use of any particular caching strategy.

4.4.3 Memory usage

Applications SHOULD use current industry best practises to avoid memory leaks and to free memory when it is no longer required. In particular, applications SHALL unregister all event listeners before termination, and SHOULD unregister them as soon as they are no longer required.

Where available, applications SHALL use explicit destructor functions to indicate to the platform that resources may be re-used by other applications.

Applications MAY use the `gc()` method on the `application/oipfApplicationManager` embedded object to provide hints to the OITF that a garbage collection cycle should be carried out. The OITF is not required to act on these hints.

The `LowMemory` event described in Section 7.2.1.4 SHALL be generated when the receiver is running low on memory. The amount of free memory that causes this event to be generated is implementation dependent. Applications may register a listener for these events in order to handle low-memory situations as they choose best.

4.4.4 Instantiating embedded objects and claiming scarce system resources

The objects defined in Section 7 of this specification are embedded objects. These are typically instantiated through the standard DOM 2 methods for creating HTML objects or the `oipfObjectFactory` as defined in Section 7.1.

All embedded objects as defined in Section 7 SHALL NOT claim scarce system resources (such as a hybrid tuner) at the time of instantiation. Hence, instantiation SHALL NOT fail if the object type is supported (and sufficient memory is available).

For each embedded object for which scarce resource conflicts may be a problem, the state diagram and the accompanying text define how to deal with claiming (and releasing) scarce system resources.

NOTE: instantiated embedded objects do not have to be added to the DOM tree in order for their ECMAScript API to be usable.

4.4.5 Media control

If insufficient resources are available to present the media, the attempt to play the media SHOULD fail except for the specific case of starting to play audio from memory (see below). For the video/broadcast object, this shall be indicated by a `ChannelChangeError` event with a value of 11 for the error state. For an A/V Control object, the `error` property shall take the value 3.

Instantiation of a `video/broadcast` or A/V Control object does not cause any scarce resources to be claimed. Scarce resources such as a media decoder are only claimed following a call to the `setChannel()`, `bindToCurrentChannel()`, `nextChannel()` or `prevChannel()` methods on a `video/broadcast` object or the `play()` method on an A/V Control object. By implication, instantiating a `video/broadcast` or A/V Control object does not cause the media referred to by the object's data attribute to start playing immediately. See Section 7.13.1.1 for details of when scarce resources are released by a `video/broadcast` object and Section 7.14.1.1 when scarce resources are released by an A/V Control object.

In the specific case of a request to play audio from memory while broadcast or broadband streaming audio is being played and where the terminal does not support mixing the audio from memory with the already playing audio, the following shall apply;

- The audio from memory shall have priority and shall interrupt the already playing audio.
- The interrupted presentation shall be resumed automatically by the terminal when the interrupting audio ends.

When audio from memory is interrupted by a resource loss, or when streaming video or audio presentation is interrupted by a resource loss caused by another request for streaming audio or video presentation, the presentation is cancelled and SHALL NOT be restored automatically by the OITF.

This specification is intentionally silent about handling of resource use by embedded applications including scheduled recordings.

4.4.6 Use of the display

A compliant OITF SHALL support at least one of the following application visualization modes for managing the display of applications:

1) Multiple applications may be visible simultaneously, with the OITF managing focus between applications, but with DAE applications managing their own size, position and visibility. In this mode the following holds:

   a) Many browsers restrict the size or location of windows; in this application visualization mode these restrictions SHALL NOT be enforced for windows associated with applications within the browser area. Any area of the display available to DAE applications may be used by any application, and no minimum size is enforced for applications. An application may choose to resize or display its DOM `Window` as appropriate, using properties and methods on the DOM `Window` object. If this application visualization mode is supported, the following properties and methods SHALL be supported on the window object in addition to what is stated in [CEA-2014-A]: `resizeTo`(), `moveTo`(), and `screen`.

      Note that the display of applications exceeding the maximum size of the browser area or of applications partially positioned outside the browser area may be cropped.

   b) applications from the same service provider that are intended to run simultaneously SHOULD take care to co-ordinate their use of the display in order to ensure that important UI elements are not obscured.

2) Multiple applications may be visible simultaneously, with the OITF managing the size, position, visibility and focus between applications. In this case methods `resizeTo`() and `moveTo`() are either not supported on the `Window` object, or have no effect whilst the OITF renders applications in this mode.

3) Only one application is visible at any time; switching to a different application either hides the currently-visible application (where simultaneous execution of more than one application is supported) or terminates the currently visible application (where simultaneous execution of more than one application is not supported). The mechanism for switching between applications is implementation-dependent. In this case, the `show()`, `hide()`, `activateInput()` and `deactivateInput()` methods of the `Application` object provide hints to the execution environment about whether the user should be notified that an application requires attention. The mechanism for notifying the user is outside the scope of this specification.

Applications SHALL be created with an associated DOM `Window` object, that covers the display area made available by the OITF to a DAE application. The size of the DOM `Window` can be retrieved through properties '`innerWidth`' and '`innerHeight`' of the DOM `Window` object.

Any areas of the browser area outside the DOM `Window` that become visible when it is resized SHALL be transparent – any video (if the hardware supports overlay as per the `<overlay*>` elements defined in Section 9.2 for the capability profiles) or applications (if multiple applications can be visible simultaneously) with a lower Z-index will be visible except where the application has drawn UI elements.

Broadcast-related and service provider related applications SHALL initially be created as invisible to avoid screen flicker during application start-up. Once loaded (as might be indicated through an onload event handler), a broadcast-related

application then typically calls the `show()` and `activateInput()` methods of its parent Application object. Broadcast-independent applications SHALL initially be created as visible and need not call these methods.

If the application does not ever need to be visible, then its DOM `Window` object will never be shown. In that case, the application should take steps to avoid being formatted to reduce computation and memory overheads. This is typically accomplished by setting the default CSS style of the document's `BODY` element to `display: none`.

Because all applications have associated DOM `Window` objects, it is possible to make any application visible even if it is not normally intended to be visible. This is of particular benefit during debugging of hidden service type applications.

The DOM `Window` for an application cannot interact with other DOM `Window` objects of other applications in the system except through the application API. In other words, scripts that are part of the document being displayed inside a DOM `Window` object cannot discover other applications without going through the application API, which acts as a single point of security control.

The default background color of the root of the document (i.e. the <html> rendering 'canvas') SHALL be a non-transparent color and SHOULD be white as most browsers, unless explicitly overriden with the following (or an equivalent) CSS construct to allow the underlying video to be shown for those areas of the screen that are not obscured by overlapping non-transparent (i.e. opaque) children of the <body> element:

```
html { background-color: transparent; }
body { background-color: transparent; }
```

Changing the visibility of an application by calling method `show()` or `hide()` on the `Application` object SHALL NOT affect its use of resources. The application still keeps running and listens to events unless the application gets deactivated (see Section 4.3.8) or destroyed (see Section 5.1.2).

4.4.7 Cross-application event handling

As defined in [DOM 2 Events], standard DOM events are raised on a specific node within a single document. This specification extends the event capability of the OITF through cross-application events handling, but does not change the DOM2 event model for dispatching events within documents. Where simultaneous execution of more than one application is supported, an OITF SHALL implement the cross-application events and cross-application event handling model described in this section.

1) An OITF SHALL implement the following cross-application event handling model. Cancelling the propagation of an event in any phase SHALL abort further raising of the event in subsequent phases: If an event is eligible for cross-application event handling (see below for more information) and is targeted at a node in the most recently activated application, then dispatch the event to that node using the standard DOM 2 bubbling/capturing of events. Default actions normally taken by the browser upon receipt of an event SHALL be carried out at the end of this step, unless overridden using the existing DOM 2 methods (i.e. using method `preventDefault()`).

2) If the cross-application event is not prevented from being propagated beyond the document root node of the application by using the exist DOM 2 methods, the event is dispatched to other active applications in the application hierarchy using the active applications list described in Section 4.3.8. The OITF SHALL iterate over the applications in the active application list, from most recently activated to least recently activated, dispatching the event to the `Application` object of each application in turn. Note that the event SHALL NOT be dispatched to the document, and default browser action SHALL NOT be carried out during this phase. Cancelling the propagation of an event in this phase SHALL abort further raising of the event in subsequent applications.

Event listeners for cross-application events are registered and unregistered using the same mechanism as for DOM2 events. Listeners for cross-application events may be registered on the `Application` object as well as on nodes in the DOM tree.

The following events are valid instances of cross-application events and are applicable for cross application event handling:

System event	Description
`KeyPress`	Generated when a key has been pressed by the user. May also be generated when a key

System event	Description
	is held down during a key-repeat.
KeyUp	Generated when a key pressed by the user has been released.
KeyDown	Generated when a key has been pressed by the user.

Table 1: Events applicable for cross application event handling

The KeyPress, KeyUp and KeyDown events are all targeted cross-application events. The events are targeted at the node that has the input focus.

All events dispatched using the standard dispatchEvent() method are normal DOM events, not cross-application events. As defined in Annex B bullet "Changes to 5.4", the OITF SHALL support the window.postMessage() method for cross-document messaging as defined in [HTML5]. The method takes two arguments; a message (of type String) to be dispatched and the targetOrigin, which defines the expected origin (i.e. domain) of the target window, or "*" if the message can be sent to the target regardless of its origin. The target of the event is the "window" of a specific application. Applications can use this method to send events to other applications. The receiving application MAY receive those events and interpret them, or MAY dispatch them in its DOM using standard DOM dispatchEvent() methods.

The visibility of an application SHALL NOT affect the cross-application event handling algorithm as defined above – an active application SHALL receive cross-application events even when it is not visible.

Incoming key events are dispatched using the cross-application event handling algorithm as defined above.

NOTE: This event dispatch model enables key events to be dispatched to multiple applications. Applications wishing to become the primary receiver for key events SHOULD call Application.activateInput(). Even though Application.activateInput() is called, another application may subsequently be activated. In order to ensure that sensitive key input (e.g. PINs or credit card details) is limited only to the application it is intended for, applications SHOULD check that they are the primary receiver of the key events (using the Application.isPrimaryReceiver property and/or the ApplicationPrimaryReceiver and ApplicationNotPrimaryReceiver events defined in Section 7.2.6) and SHOULD 'absorb' key events by calling the stopPropagation() method on the DOM2 key event.

4.4.7.1 Behaviour of the BACK key

If a remote features a "back" or "back up" key, or one offering similar functionality, the OITF SHALL handle this key as described below:

1) A VK_BACK key event SHALL be dispatched to applications following the normal key handling process described in Section 4.4.7

2) If the default behaviour of the key event is not stopped by an application using preventDefault(), then the OIPF MAY load the previous page in its history list for DAE applications.

4.5 Parental access control

The present document permits a number of different approaches to parental access control.

a) Enforcement in the network.

   An IPTV service provider MAY manage parental access control completely in the network. Applications running on application servers back in the network MAY decide to block access to content or arrange a DAE application to ask for a PIN code as necessary. This approach can apply to any kind of content - streaming on-demand content, IP broadcast content and to downloaded content.

   No specific support is needed for this approach in the specification.

b) Enforcement in the OITF CSP / CSPG for protected MPEG-2 TS content

IPTV service providers MAY use the content protection mechanism for protected content to enforce access control to protected content. If used, this enforcement will happen in the OITF and in some cases in the CSP Gateway as well. In this approach, the content protection mechanism in the OITF would ask for PIN codes as needed.

The OITF CSP/CSPG-based enforcement of this approach and link to DAE API and events are defined in:

- Section 4.1.5.1 of [OIPF_CSP2], for CSP terminal centric approach,
- Sections 4.2.2, 4.2.3.4.1.1.5 and 4.2.3.4.1.1.6 of [OIPF_CSP2] for CI+ CSP Gateway centric approach
- Sections 4.2.2 and 4.2.4.5.1 of [OIPF_CSP2] for DTCP-IP CSP Gateway centric approach

c) enforcement in the OITF

OITFs MAY enforce parental access controls themselves. Examples include embedded applications offering access to:

- IP delivered content based on information delivered to the metadata CG client.
- classical broadcast content in hybrid OITFs
- content delivered to the OITF (either streaming or downloaded)

In approaches b) and c), PIN dialogs would be generated by code forming part of the OITF implementation. The APIs in Section 7.9 provide some control over these dialogs. The PIN would typically be configured by an embedded application but MAY also be configured by a DAE application using the optional APIs defined in Section 7.3.2 of the present document.

These approaches b) and c) are reflected in a number of failure modes as defined in the following Sections of the specification;

- For broadcast channels (both IP and hybrid), in Section 7.13.1, see `onChannelChangeError` where errorState 3 is defined as "parental lock on channel"
- Parental rating errors and parental rating changes during playback of A/V content through the CEA-2014 A/V embedded object and the `video/broadcast` object are reported according to the mechanism described in 7.14.6 and 7.13.5 respectively.

NOTE: Due to the variation in regulatory requirements and deployment scenarios, the present document is intentionally silent about which of these approaches or combination of approaches is used.

4.6 Content download

This requirements in this section apply if the <download> element has been given value `true` in the OITF's capability profile as specified in Section 9.3.4.

4.6.1 Download manager

An OITF SHALL support a native download manager (i.e. "Content Download" component) to perform the actual download and storage of the content, and which allows the user to manage (e.g. suspend/resume, cancel) and monitor the download, in a consistent manner across different service providers. The download manager SHALL continue downloading as a background process even if the browser does not have an active session with the server that originated the download request anymore (e.g. has switched to another DAE application), even if the OITF restarts or even if the OITF suffers a network failure. The download manager continues with the download as a background process until it succeeds or the user has given permission to terminate the download. (see Section 4.6.4 on HTTP Range support to resume HTTP downloads after a power/network failure).

The native download manager SHALL be able to offer a visualization of its status through the `application/oipfStatusView` embedded object as defined in Section 7.15.2.1.

If the attribute manageDownloads of the <download> element in the client capability capability description is unequal to "none", the native download manager SHALL offer control over the active downloads through the Javascript API defined by the application/oipfDownloadManager embedded object in Section 7.4.3.

NOTE 1: Once (sufficient data of) the content has been downloaded, the content MAY be played back using a native application, and MAY be played back using an A/V control object. In the latter case, see method setSource() in Section 7.14.8 for more information.

NOTE 2: Annex D clarifies the content download usage scenario in more detail

4.6.2 Content Access Download Descriptor

An OITF SHALL support parsing and interpretation of the Content Access Download Descriptor document format with the specified semantics, syntax and MIME type as specified in Annex E.

4.6.3 Triggering a download

An OITF SHALL support a non-visual embedded object of type application/oipfDownloadTrigger, with the Javascript API as defined in Sections 7.4.1 and 7.4.2 to trigger a download.

The following subsections define some details about the different ways of triggering a download.

4.6.3.1 Using the registerDownload() method

The registerDownload() method takes a Content Access Download Descriptor as one of its arguments and passes it to the underlying native download manager in order to trigger a download. The following requirements apply:

- The Content Access Download Descriptor MAY be created in Javascript or MAY be fetched using XMLHttpRequest. To this end the OITF SHALL pass the data inside the Content Access Download Descriptor into the XMLHttpRequest.responseXML property in Javascript for further processing, if the OITF encounters an HTTP response message with the Content-Type application/vnd.oipf.ContentAccessDownload+xml, as the result of an XMLHttpRequest.

  NOTE: The behaviour in other cases when the OITF encounters an HTTP response message with the Content-Type application/vnd.oipf.ContentAccessDownload+xml, for example whilst following a link as specified by an anchor element (<a>), is not specified in this document.

- If the OITF supports a DRM agent with a matching DRMSystemID as per Section 9.3.10, the OITF SHALL pass included DRM-information as part of the <DRMControlInformation> elements of a Content Access Download Descriptor to the DRM agent.

- If the Content Access Download Descriptor contains multiple content items to be downloaded, then all items are considered to belong together. Therefore, the download of each individual content item has the same download identifier in that case (whereby the ContentID may be used for differentiation). The order by which the items are downloaded is defined by the OITF.

4.6.3.2 Using the registerDownloadURL() method

The registerDownloadURL() method takes a URL as one of the arguments and passes it to the underlying native download manager in order to trigger a download. The URL MAY point to any type of content. The URL MAY also point to a Content Access Download Descriptor (i.e. with argument contentType having value "application/vnd.oipf.ContentAccessDownload+xml"). In that case, the method returns a download identifier. The OITF will then fetch the Content Access Download Descriptor, after which the same must happen as if method registerDownload() as defined in Section 4.6.3.1 with the given Content Access Download Descriptor as argument was called.

4.6.3.3 Using the optional registerDownloadFromCRID() method

The registerDownloadFromCRID() method is an optional method as defined in Section 7.4.2 and takes a Content Reference Identifier (CRID) as one of its arguments that is passed to the underlying native download manager in order to trigger a download.

4.6.3.4 General behaviour regarding triggering a download

The following are general behavioural requirements apply to triggering downloads:

1) Fetching the content will typically be initiated immediately. However, the OITF MAY defer the download to a later time.

2) An OITF SHOULD offer an easy way to continue the UI interaction with the server from which a download has been initiated, e.g. allowing him/her to continue browsing on the page that triggered the download.

3) An OITF SHOULD inform the user if the content-type of a content item being retrieved cannot be interpreted by the OITF.

4.6.4 Download protocol(s)

The OITF SHALL support the HTTP protocol for download as specified in Section 5.2.3 of [OIPF_PROT2]. In addition, the OITF SHALL support the following requirements:

1) As specified in Section 5.2.3 of [OIPF_PROT2], if a server offers a content item for download using HTTP, the server SHALL make sure that HTTP Range requests as defined in [RFC2616] are supported for HTTP GET requests to the URI of that downloadable content item, in order to be able to resume downloads (e.g. after power or network failure).

2) If the OITF receives an HTTP 404 “File Not Found” status code, the OITF SHALL stop its attempts to resume the download, and go to a “Failed Download” state. The handling of other error codes is implementation dependent.

3) If after downloading a content item the size of the downloaded content item does not match the indicated size parameter or the value for the optional attribute `MD5Hash` of the given `<ContentURL>` does not match the hash of the downloaded content, the OITF SHOULD remove the downloaded content item.

Integration with download protocols other than HTTP are not specified in this document.

4.7 Streaming CoD

This section defines the content-on-demand streaming interfaces for both DRM-protected and non-DRM protected content.

4.7.1 Unicast streaming

An OITF SHALL support unicast streaming by setting the `data` property of the CEA-2014 A/V Control object to any of the following three types of value:

1) A Public Service Identifier (PSI) as defined in Protocol Specification [OIPF_PROT2].

2) The HTTP or RTSP URL of the content to be streamed. See [Req. 5.7.1.f] of [CEA-2014-A] for details.

3) The URL of a Content Access Streaming Descriptor, in the manner as defined in Section 7.14.2. In this case the application SHALL set the `type` attribute to “`application/vnd.oipf.ContentAccessStreaming+xml`”.

   Example:

```
<object id="d1" data=http://www.openiptv.org/fetch?contentID=25
type="application/vnd.oipf.ContentAccessStreaming+xml" width="200" height="100"/>
```

In the first two cases, the application SHALL set the `type` attribute to the MIME type of the content referred to by the value of the `data` attribute to provide a hint about the expected content type, in order for the browser to instantiate the proper CEA-2014 A/V Control object.

In order to support method 3, an OITF SHALL support parsing and interpretation of the Content Access Streaming Descriptor document format with the specified semantics, syntax and MIME type as specified in Annex E.2.

Support for Unicast streaming through the CEA-2014-A A/V Control object SHALL be indicated as defined in Section 9.3.11.

For more details about setting up the A/V stream through a Content Access Streaming Descriptor, see Section 7.14.2, Section 8 and Annex D.

4.7.2 Multicast streaming

If an OITF has indicated support for IPTV channels through a <video/broadcast> element with type ID_IPTV_* (as defined in Section 7.13.12.1) the OITF SHALL support passing a content-access descriptor through the 'contentAccessDescriptorURL' argument of the 'setChannel'-method of the video/broadcast object (as defined in Section 7.13.1.3). If the content-access descriptor includes DRM information, the OITF SHALL pass this information to the DRM agent.

4.8 Scheduled content

If an OITF has indicated support for playback and control of scheduled content, then it SHALL support the video/broadcast embedded object defined in Section 7.13.1. In addition, it SHALL adhere to the requirements for conveyance of the channel list as specified in 4.8.1. To protect against unauthorized access to the tuner functionality and people's personal favourite lists, the OITF SHALL adhere to the security model requirements as specified in Section 10.1, in particular the tuner related security requirements in Section 10.1.4.1.

NOTE: This section and Section 7.13 are focused on control and display of scheduled content received over local tuner functionality available to an OITF. The term "tuner" is used here to identify a piece of functionality to enable switching between different types of scheduled content services that are identified through logical channels. This includes IP broadcast channels, as well as traditional broadcast channels received over a hybrid tuner.

NOTE 2: The APIs in this section allow for deployments whereby the channel line-up and favourite lists for broadcasted content are managed by the client, the server, or a mixture thereof.

4.8.1 Conveyance of channel list information

To enable an application to control the tuner functionality on an OITF, the OITF needs to convey the channel list information that is managed by native code on the OITF device to the application (either the channel list information is provided locally on the OITF via Javascript, or the channel list is communicated directly to a server). This information includes the list of uniquely identifiable channels that can be received by the physical tuner of a hybrid device, including information about how the channels are ordered and whether or not these channels are part of zero or more favourite lists. It also includes the channel line-up and the favourite lists that MAY be managed by an OITF for IP broadcast channels.

The API supports two methods of conveying the channel list information to an application:

1) Method 1: through Javascript, by using the method getChannelConfig(), as defined in Section 4.8.1.1.

2) Method 2: through an HTTP POST message that is sent upon the first connection to a service that requires tuner control, as defined in Section 4.8.1.2.

An OITF SHALL support method 1, and SHOULD support method 2.

If an OITF conveys the channel list information using the HTTP POST message defined in method 2, then the server SHALL, if it supports method 2, receive the conveyed channel list information and SHOULD rely on this information for the purpose of exerting tuner control. If a service supports using the channel list information sent through the HTTP POST method to exert tuner control , the server SHALL indicate this compatibility with method 2 using the postList attribute specified in Section 9.3.1 (i.e., <video_broadcast postList="true">true</video_broadcast>), in the server capability description.

If the server does not support method 2, the service SHALL rely on the getChannelConfig() method defined in Section 7.13.1.3 to access the channel list information. If an OITF does not support method 2, the HTTP message of the first connection to the service that requires tuner control SHALL be an HTTP GET message with an empty payload and the service SHALL instead rely on the getChannelConfig() method defined in Section 7.13.1.3 to access the channel list information. If support for method 2 is indicated by both the OITF and the server (through respective capability exchanges), the OITF SHALL convey the channel list information using method 2.

If an OITF does not manage/maintain the channel line-up (i.e. does not have a locally stored channel line up), the getChannelConfig() method described in Section 7.13.1.3 SHALL return null, and the HTTP message described in Section 4.8.1.2 SHALL be an HTTP GET message with an empty payload. In that case, the application MAY use the

createChannelObject() method as defined in Section 7.13.1.3 to create channel objects that can be used on subsequent setChannel() requests, and in this way can manage/maintain its own channel list.

NOTE: conveyance of the channel list SHALL adhere to the security model requirements as specified in Sections 10.1.4.1 and 10.1.4.1.1.

4.8.1.1 Method 1: Javascript method "getChannelConfig()"

The OITF SHALL support method getChannelConfig() as defined in Section 7.13.1.3 for the video/broadcast embedded object. This method returns a ChannelConfig object as defined in Section 7.13.8.

4.8.1.2 Method 2: HTTP POST message

If an OITF supports sending the channel list through HTTP POST and a server has indicated that it uses the posted channel list information to exert control of the tuner functionality of an OITF (i.e. using attribute postList="true" in the server capability description) for a particular service, then the OITF SHALL issue an HTTP POST if it decides to connect to that service. The body of the HTTP POST request SHALL contain the Client Channel Listing, which SHALL adhere to the semantics, syntax and XML Schema that are defined for the Client Channel Listing in Annex G. The interaction SHALL be secured using Transport Layer Security (TLS). The server SHALL silently ignore unknown elements and attributes that are part of the Client Channel Listing.

The server SHALL return a HTML document.

If the favourite lists are not (partially) managed by the OITF, the Client Channel Listing SHALL neither contain the FavouriteLists nor the CurrentFavouriteList element.

4.8.2 Conveyance of channel list and list of scheduled recordings

This section and the following sections SHALL apply to OITFs that have indicated <recording>true</recording> as defined in Section 9.3.3 in their capability profile.

To enable a service to schedule recordings of content that is to be broadcasted on specific channels, the OITF needs to convey the channel list information that is managed by the native code on the OITF. This information typically includes the channel line-up of the tuner of a hybrid device. The conveyance of channel list information and scheduled recordings is based on the same two methods of conveying the channel list information to a service as defined in Section 4.8.1:

1) Method 1: through Javascript, by using the method getChannelConfig(). To this end, the OITF SHALL support method getChannelConfig() as defined in Section 7.10.1.1 for the application/oipfRecordingScheduler object.

2) Method 2: through an HTTP POST message as defined in Section 4.8.1.2 that is sent upon the first connection to a service that has indicated that it requires control of the recording functionality and that has indicated compatibility with method 2 using the postList attribute specified in Section 9.3.3 (i.e., <recording postList="true">true</recording>), in the server capability description for a particular service.

An OITF SHALL support method 1, and SHOULD support method 2. If support for method 2 is indicated by both the OITF and the server (through respective capability exchanges), the OITF SHALL convey the channel list information using method 2. Otherwise, the HTTP message of the first connection to the service that requires tuner control SHALL be an HTTP GET message with an empty payload.

If a server has indicated that it requires control of both the tuner functionality and the recording functionality available to an OITF (i.e. by including both <video_broadcast> and <recording> with value true in the OITF's capability description), the body of the HTTP POST message SHALL contain a single instance of the Client Channel Listing whereby the <Recordable> element defined in Annex G SHALL be used to indicate whether channels that can be received by the tuner of the OITF can be recorded or not.

If an OITF does not manage the channel line-up, the getChannelConfig() method described in Section 7.10.1.1 SHALL return null, and the HTTP message described in Section 4.8.1.2 SHALL be an HTTP GET message with an empty payload.

In addition, the OITF SHALL also support method getScheduledRecordings() as defined in Section 7.10.1.1. This method returns a ScheduledRecordingCollection object, which is defined in Section 7.10.3.

Note that the conveyance of the channel listing and the scheduled recordings is subject to the security model requirements specified in Section 10.1, and in particular the recording related security requirements in Section 10.1.4.2.

4.9 DLNA RUI Remote Control Function

This section describes the DLNA RUI RCF (Remote Control Function) and the interactions between the different entities involved. It builds on the RUI feature defined by the DLNA Networked Device Interoperability Guidelines (August 2009) [DLNA] and shows how the DLNA RUI can be integrated into an OITF and used by DAE applications.

The DLNA RUI RCF is the feature that enables a Remote Control Device to be able to control the OITF or a DAE application running on it, from that Remote Control Device. To support this feature, a Remote Control Device SHALL support the DLNA RUIC function and an OITF SHALL support the DLNA RUIS function (as defined in Section 3.17).

The DLNA RUI RCF provides two main features:

- Providing a Control UI to the Remote Control Device.
  - The Control UI is a CE-HTML document through which the user will control the OITF directly or a DAE application on the OITF. There are two options based on the origin of the Control UI for sourcing it as follows:
    - Sourcing the Control UI from the OITF itself.
    - Sourcing the Control UI from an IPTV Applications server via the OITF.
- Interactions to exchange control messages and results
  - The Control UI in the DLNA RUIC sends control messages to the OITF or DAE application and receives the corresponding results.

The following sections will introduce the interfaces between the entities that support the DLNA RUI RCF.

4.9.1 Interfaces used by the DLNA RUI Remote Control Function

This section describes interfaces related to the DLNA RUI RCF. There are three entities (Remote Control Device, OITF and IPTV Applications server) that communicate with each other through the interfaces described in Figure 4.

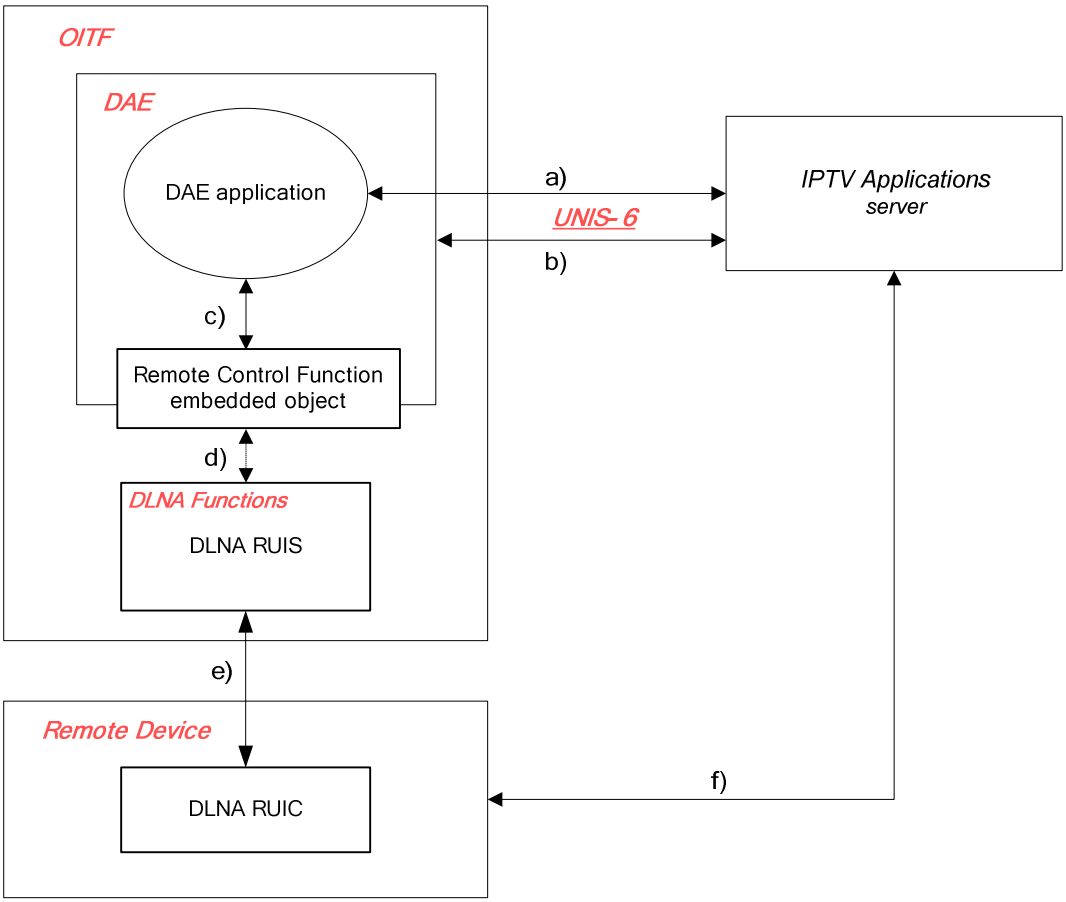


Figure 4 - OIPF architecture with DLNA RUI RCF scenario

Figure 4 shows the entities in the OIPF Architecture involved in the DLNA RCF and the interfaces between them.

The dotted line “d)” between the RCF embedded object and the DLNA RUIS indicates that it is a local interface and hence not defined by this specification. The detailed behaviour of each interface is defined as follows:

1) Interface a)

   This interface is used to retrieve a Control UI from an IPTV Applications server by using XMLHttpRequest object (the Control UI retrieved through interface a) will be delivered to DLNA RUIC via interfaces c), d) and e), sequentially).

2) Interface b)

   This interface is used by the DAE browser to retrieve a DAE application containing an RCF object when the DLNA RUIC requests a DAE application to execute in the OITF.

3) Interface c)

   The DLNA RUI RCF APIs use this interface to enable a DAE application to get the request originating from the DLNA RUIC, through an event dispatched by the OITF, and send the corresponding response or any other information to the DLNA RUIC via the DLNA RUIS.

4) Interface d)

   This is a local interface that is used to pass messages between an RCF object in a DAE application and the DLNA RUIS.

5) Interface e)

   This is a DLNA RUI compatible interface which provides device discovery, sending/receiving HTTP messages and notifications.

When the DLNA RUIC is activated by a user, the DLNA RUIC searches for a DLNA RUIS and does a capability exchange. Then, the DLNA RUIC retrieves the XML UI Listing from the DLNA RUIS and displays it to the user. When the user chooses one of the Control UIs, the DLNA RUIC retrieves the selected Control UI from the DLNA RUIS in the OITF.

The Control UI may send an HTTP request to deliver a message (for example, plays an AV content) and receive a response from the DLNA RUIS.

This interface is also used for the DLNA RUIS to send a 3rd party notification defined in Section 5.6.1 of [CEA-2014-A].

6) Interface f)

This interface is used by the selected Control UI (CE-HTML document) to retrieve resources (For example, images, CE-HTML documents, or css or javascript files) directly from the IPTV Applications server.

4.10 Power Consumption

The power states described in this section relate to states exposed to the DAE application. There may be other states supported by the OITF which are not described here.

The OITF will be in one of a number of power states. Its default state is “off” which consumes no power. The OITF SHALL support an “on” state where it is running in normal operation. The OITF SHALL support at least one standby state where nothing is being output to the display but power is consumed. There MAY be two types of standby states: an “active standby” and a “passive standby” state. An OITF in the “passive standby” state has the smallest possible power consumption (for example, average under 1W) which MAY be in line with European Commission Code of Conduct, US Energy Star or other regional requirements. In this state the IR listener and wakeup clock MAY be active but no DAE application is active. The IR listener allows the user to turn on the OITF using a remote control. A DAE application MAY use the wakeup clock to schedule the OITF enter the “active standby” state, for example to perform a recording.

Note there may be different levels of “active standby” state but the assumption is that, at least, nothing is being output to the display and one or more DAE applications may execute in the background.

The following explanation describes the behaviour of the OITF when transitioning between the mentioned states and how a DAE application is affected.

A DAE application SHALL be able to execute in the “on” and “active standby” states but SHALL NOT be able to execute in the “off” or “passive standby” states.

When an OITF is turned “on” from an “off” state a DAE application has to be explicitly selected by the user to be executed or the OITF has identified a DAE application to be auto-started. A DAE application has no direct control if it shall auto-start or not and this is left for the OITF to manage. A DAE application MAY auto-start if the Service Discovery and Selection has taken place and the user has selected a service provider.

When an OITF changes to an “off” or “passive standby” state from an “on” or “active standby” state, the DAE application SHALL get an `ApplicationDestroyRequest` event. The DAE application has an opportunity to take a final action and gracefully quit or it shall be killed forcibly.

4.10.1 DAE application wake-up support

The OITF MAY support wake-up requests from a “passive standby”. There are two types of wake-up requests, one on an individual DAE application and one on the OITF. The supported wakeup is indicated in the power consumption capability information.

4.10.1.1 Single DAE application wakeup

The OITF MAY support wake-up requests for individual DAE applications when in “passive standby”. Similar to a scheduled recording, a DAE application may need to execute at a predetermined time. At the wake-up point the DAE application executes and when it completes its task returns to a “passive standby” state by exiting.

There SHALL only be one wake-up request per DAE application. There MAY be multiple wake-up requests from different DAE applications which SHALL execute independently. The OITF SHALL silently ignore all wake-up requests whose timers expire when it is not in the “passive standby” state.

When the DAE application terminates and the OITF changes to an “active standby” or “on” state for other reasons than a wake-up request the OITF SHALL NOT change power states.

Through capability information it is possible to determine if wake-up and standby modes are supported by OITF.

This is an example of how a DAE application may setup a wake-up request in OITF.

Precondition: The DAE application is actively running and the OITF is either in “on” or “active standby” states.

1) End user selects to go into “passive standby” natively.

2) An `ApplicationDestroyRequest` event is generated

3) The DAE application calls the `prepareWakeupApplication()` method and sets a token, time for wake-up and URI associated with the DAE application. The DAE application then quits, e.g. by calling `destroyApplication()` on its parent `Application` object..

4) The OITF goes into “passive standby” state.

5) When the wake-up time triggers, the OITF changes to “active standby” and the DAE application is initiated with the URI specified in the prior call to `prepareWakeupApplication()`.

6) The DAE application then runs `clearWakeupToken()` to get the token set in the prior call to `prepareWakeupApplication()`.

7) DAE applications executes.

8) Once the DAE application completes execution it shall exit. The OITF changes automatically to a “passive standby” state.

If the OITF is turned “on” while in this mode the OITF SHALL NOT enter “passive standby” state.

4.10.1.2 OITF wakeup

The OITF MAY support wake-up requests for the OITF when in “passive standby”. The application when receiving an event onApplicationRequest may request to wake-up the OITF at a set time using method `pepareWakeupOITF()`.

OITF SHALL silently ignore all wake-up requests whose timers expire when it is not in the “passive standby” state.

4.10.2 OITF hibernate support

The OITF MAY support a hibernate mode which allows DAE applications and their state to be stored in memory when in a “passive standby” state. The support of a hibernate mode greatly reduces the start-up time for DAE applications (for example, start-up times of 3 seconds may be reached).

When the OITF resumes from the hibernate mode, it shall restore all of the previous DAE applications with their previous state and SHOULD assign the same resources to the DAE applications as they had prior to the hibernate mode. If this is not possible, the regular callback functions SHALL be used to inform the affected DAE application.

If hibernate mode is supported the event `ApplicationHibernateRequest` is generated instead of `ApplicationDestroyRequest` when the OITF enters a “passive standby” state.

If the OITF supports hibernate mode only the OITF wake-up request is supported. The single DAE application wake-up SHALL NOT be supported. The reason for this limitation is due to the difficulty to support both options.

A wake-up support SHALL NOT make the OITF resume from the hibernate mode. The wake-up support SHALL be supported independently.

The OITF SHALL indicate support for hibernate mode through the capabilities information.

4.10.3 State diagram for the power state

The following state machine provides an overview of the power state changes that may occur relating to power consumption. The transitions in the state machine due to `setPowerState()` may be also be triggered by user generated events handled natively by the OITF.

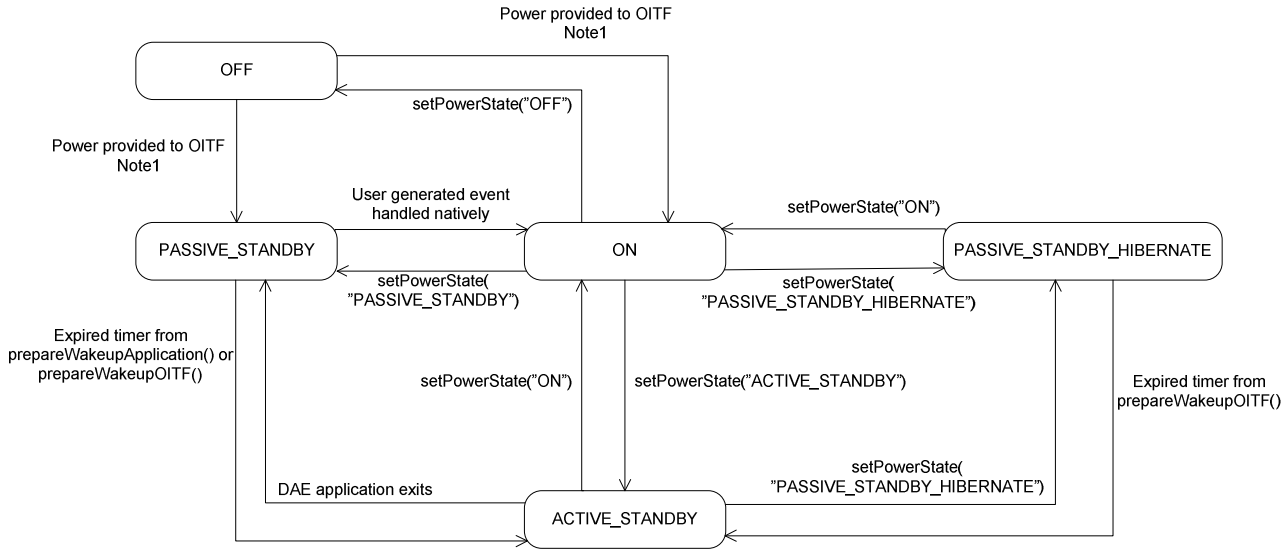


Figure 5 - State diagram of OITF power states

Note1 – The transition from the OFF state to the `PASSIVE_STANDBY` or `ON` states is manufacturer dependent.

4.11 Display Model

Annex H describes the logical display model of an OITF and the relationship between DAE application graphics and video.

5 DAE Application Model

5.1 Application lifecycle

This section describes the lifecycle of a DAE application, including when an application is launched, when it is terminated and the behaviour when a DAE leaves the boundary of one application and enters another.

APIs related to DAE applications are described in Section 7.2.

5.1.1 Creating a new application

5.1.1.1 General

The present document defines a number of different application lifecycle models. These include;

- Applications started through an OITF-specific user interface
- Using the `Application.createApplication()` API call
- CE-HTML third party notifications
- Service-related applications (from SD&S signalling)
- Broadcast-related applications (either from SD&S signalling or from broadcast signalling in a hybrid device)
- Applications started by the DRM agent
- Applications provided by the AG through the remote UI
- Broadcast independent applications
- Widgets started through the `Application` class

All HTML, ECMAScript and SVG files that comprise an application SHALL be retrieved from sites within the current application boundary of that application (see Section 5.1.3) . If the application attempts to access files of these types from outside the application boundary, this access SHALL fail as if the content did not exist. Files with other MIME types supported by OITF may be retrieved from outside the application boundary.

If the document of an application is modified (or even replaced entirely by other pages loaded from within the application boundary), the `Application` object is retained. This means that the permission set granted when the application is created applies to all “edits” of the document or other pages in the application, until the application is destroyed.

5.1.1.2 Applications started through an OITF-specific user interface

These SHALL be presented as broadcast-independent applications.

5.1.1.3 Using the Application.createApplication API call

Creating a new application is accomplished by creating a new `Application` object via the `Application.createApplication()` method. Calling this method will create a new application and add it to the application tree in the appropriate location.

```
// Assumes that the application/oipfApplicationManager object has the ID
// “applicationmanager”
Var appMgr = document.getElementById(“applicationmanager”);
var self = appMgr.getOwnerApplication(Window.document);

// create the application as a child of the current application
var child = self.createApplication( url_of_application, true );
```

The URL passed to the `createApplication()` method SHALL be one of the following;

- An HTTP or HTTPS URL referring to an XHTML page as defined by Section 6.1 of this specification.
- An HTTP or HTTPS URL referring to an XML AIT as defined by Section 5.2.8 of this specification.
- The DVB URI for launching service provider related applications signalled through SD&S as defined in Section 8.3 of this specification
- The DVB URI for launching broadcast-related applications from the current service signalled through SD&S as defined in Section 8.3 of this specification. Where an OITF supports the MPEG-2 encoding of the AIT as defined in Section 5.2.7.2, this form of the DVB URI SHALL also be supported for launching broadcast-related applications from the current service when that service includes an MPEG-2 AIT.

5.1.1.4 CE-HTML third party notifications

The lifecycle of these is defined by [CEA-2014-A] and summarised in Section 5.3.1 of the present document.

5.1.1.5 Starting applications from SD&S Signalling

These are described in Section 5.2. All applications started by SD&S signalling are treated as siblings and are children of the hidden system root node (see Section 4.3.3).

5.1.1.6 Applications started by the DRM agent

These SHALL be presented as broadcast-independent applications.

5.1.1.7 Applications provided by the AG through the remote UI

OITFs MAY include the capability to start these applications from an embedded application. OITFs SHALL include the ability for applications to discover these as defined by the `application/oipfGatewayInfo` embedded object in Section 7.7.1.

5.1.2 Stopping an application

The `destroyApplication()` method (as specified in Section 7.2.2.2) SHALL terminate the application. An application may register a listener on the `ApplicationDestroyRequest` event in order to do some clean-up before being destroyed completely. After the `destroyApplication()` method returns, further execution of the specified application SHALL NOT occur.

When an application is terminated, all associated resources SHALL be freed (or marked available for garbage collection). Any active network connections will be terminated. Any media content being presented by the application is stopped, although recordings or content downloads initiated by the application will not be affected.

Note that terminating an application does not imply any effect on the state of the DAE execution environment.

Additional requirements are defined for stopping selected service provider applications and applications part of scheduled content services in Sections 5.2.4.3 and 5.2.3.2 respectively.

5.1.3 Application Boundaries

All of the pages that make up an application are contained within its application boundary. This is the “fully qualified domain name” (FQDN) of the initial page of the application in the absence of an application_boundary_descriptor.

If an `applicationBoundary` element is present in the SD&S signalling for an application as defined in [TS 102 809], the application boundary SHALL also include the FQDNs listed in the `applicationBoundary` element. Note that this element also defines the protocols over which these FQDNs may be accessed.

For files requested with `XMLHttpRequest`, the Same-Origin Policy SHALL be extended using the application domain; i.e. any domain in the application domain SHALL be considered of same origin.

Extending the origin of XMLHttpRequest is potentially dangerous, and may lead to undesired leaking of private information. To make sure that the integrity of the user is not compromised, the OITF SHOULD include a mechanism which allows the user to exclude domains from application boundaries of applications.

5.2 Application announcement & signalling

5.2.1 Introduction

This specification defines 3 basic types of application;

- Applications related to one or more broadcast TV or radio channels. These MAY run while one of the channels which they are related to is being presented by the OITF. These are signalled through the SD&S broadcast or package discovery records or included in an application discovery record which is referenced from the broadcast or package discovery record.
- Applications related to the service provider selected through the service selection process. These MAY run at any time until the service provider selection process is repeated and a different service provider selected. These are signalled through the SD&S service provider discovery record or included in an application discovery record which is referenced from the service provider discovery record.
- Applications independent of either of the above. These MAY run at any time. These are started by other applications and are not signalled anywhere.

Each of these types is described in more detail below.

5.2.2 General

Section 4.3.3 of this specification describes how one application may start another application either as a sibling or as a child. All applications started via SD&S signalling as described in this section SHALL be started as children of the hidden system root node, as described in Section 5.1.1.5.

Any application may be signalled as AUTOSTART or PRESENT (see Table 3 below and Section 5.2.4.3 of [TS 102 809]). Applications signalled as AUTOSTART are intended to be automatically started by the OITF. Applications signalled as PRESENT are intended to be started only by other applications. Broadcast related applications may alternatively be signalled as KILL (see below) or PREFETCH.

It is up to the OITF manufacturer to ensure a good quality of experience concerning;

- Navigation within a DAE application.
- Accessing the available DAE applications, both available for launch, and those already running.
- Managing the life cycles of all DAE applications able to be used concurrently.

It is outside the scope of this specification whether there are dedicated keys on a remote control (e.g. the "menu", "home" or "guide" key), there is an entry in an on-screen menu or there are some other mechanism.

It is OPTIONAL for the OITF to support an exit mechanism directly accessible by the end-user. If one is supported, it is outside the scope of this specification whether this mechanism is a button on a remote control, an item in an on-screen menu or something else. If such a mechanism is supported then it SHALL only stop the application the end-user is currently interacting with and any children. The parent application and any siblings SHALL NOT be stopped.

Additionally any application MAY be stopped under the following circumstances;

- The application itself exits.
- The application's parent exits.
- It is stopped by the application which started it or another application which has a reference to it's application object.
- In response to changes in the application signalling as defined below for broadcast related applications and service provider related applications.

In all these above cases except the first (when an application itself exits) when an application is stopped by the OITF, an `ApplicationDestroyRequest` event (as defined in Section 7.2.6) SHALL be raised on the application. In the following error conditions, an application being stopped SHOULD have an `ApplicationDestroyRequest` event raised if this is possible.

- The OITF runs out of resources for applications and has to stop some of them in order to keep operating correctly.
- The OITF has determined that an application is non-responsive or has crashed.

5.2.3 Broadcast related applications

5.2.3.1 General

Providers of broadcast TV channels may signal broadcast related applications as part of the SD&S broadcast discovery record (see Section 3.2.3 of [OIPF_META2], also Sections 4.2.1 and 5.4.3.2 of [TS 102 809]). As an optimisation, broadcast related applications which are associated with a group of channels may be signalled as part of the SD&S package discovery record (see Section 3.2.3 of [OIPF_META2], also Section 5.4.3.1 of [TS 102 809]). Broadcast related applications may be included in the SD&S broadcast or package discovery records or included in an application discovery record which is referenced from the broadcast discovery record.

Broadcast-related applications can also be signalled in-line in an MPEG-2 transport stream using the MPEG-2 encoding of the AIT as defined in Section 5.2.7.2 below.

When a broadcast TV channel starts being presented, the OITF SHALL follow the “Procedure for Starting and Stopping Broadcast Related Applications on Channel Change” defined below.

While a broadcast TV channel is being presented, the OITF SHALL monitor for changes in the SD&S information as defined by Section 4.1.1.3 of [OIPF_META2]

When changes are detected, the OITF SHALL follow the “Procedure for Starting and Stopping Broadcast Related Applications When Signalling is Updated” defined below.

NOTE: The typical “red button” behaviour can be achieved by having the first page of an AUTOSTART broadcast related application be full screen and transparent to video except for an image showing a red button. Only when the user generates a “red” key event does the application display more of its user interface.

OITFs MAY include the capability to start and stop a broadcast-related DAE application instead of analogue teletext services as part of a scheduled content service or channel. Typically this would re-purpose the same mechanism used to start an analogue teletext service – for example a “text” button on a remote control. These are identified using the application usage mechanism defined in [TS 102 809] and Section 5.2.7 below.

5.2.3.2 Stopping

In addition to what is stated in 5.2.2, broadcast related applications are stopped when

- Changing between channels as defined in the “Procedure for Starting and Stopping Broadcast Related Applications on Channel Change” below.
- The OITF detects an update to the signalling for a currently presented channel as defined in “Procedure for Starting and Stopping Broadcast Related Applications When Signalling is Updated” below.
- The OITF stops presenting any broadcast channel.

5.2.3.3 Procedure for starting and stopping broadcast related applications on channel change

When a scheduled content service is selected, the following SHALL apply;

- The OITF shall determine if there are any applications signalled as part of the service as defined by Sections 3.2.3.1 and 3.2.3.2 of [OIPF_META2].
- Applications which are related to that scheduled content service and which are signalled with a control code of AUTOSTART SHALL be started if not still running from any previously presented linear TV service. They SHALL be started commencing with the highest priority application working downwards in priority while resources in the OITF permit.
- Applications which are related to that scheduled content service, which are signalled with a control code of AUTOSTART and which are already running from a previously presented scheduled content service SHALL

a) continue to run uninterrupted if the `serviceBound` element of the `ApplicationDescriptor` in their signalling has value `false`.

b) be stopped and re-started if the `serviceBound` element of the `ApplicationDescriptor` in their signalling has value `true`.

- Applications which are related to that scheduled content service and which are signalled with a control code of PRESENT SHALL continue to run if already running but SHALL NOT be started if not already running.
- Running applications from any previously presented scheduled content service which are not part of the new scheduled content service SHALL be stopped as part of the change of presented service.

The following flowchart shows the behaviour that SHALL apply when the selected channel changes:

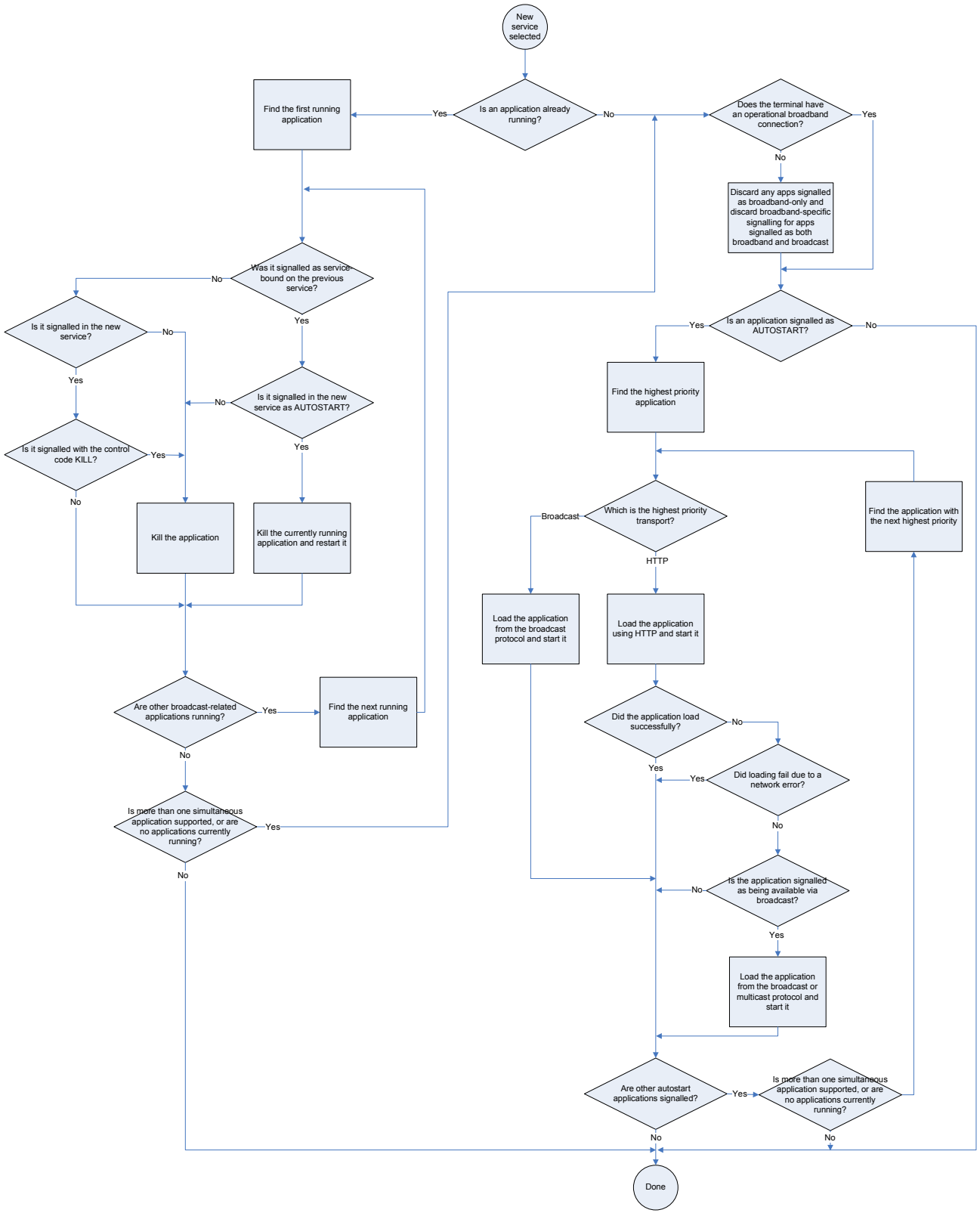


Figure 6: Behaviour when the selected channel changes

5.2.3.4 Procedure for starting and stopping broadcast related applications when signalling is updated

When the application signalling for a scheduled content service is updated, the following apply;

- Applications which are added to the service with a control code of AUTOSTART SHALL be automatically started when their addition is detected by the OITF. They SHALL be started commencing with the highest priority application working downwards in priority while resources in the OITF permit. Applications added to the service with any other control code SHALL NOT be automatically started.

- Applications which are part of the service whose control code changes to AUTOSTART from some other value SHALL be automatically started unless already running.

- An application which is removed from the service or whose control code changes to KILL SHALL be stopped.

- If application signalling is removed from a service, all running broadcast-related applications SHALL be stopped (i.e. the same behaviour as signalling an empty AIT).

The following flowchart shows the behaviour that SHALL apply when the application signalling for the currently selected service changes, or when a running broadcast-related application exits:

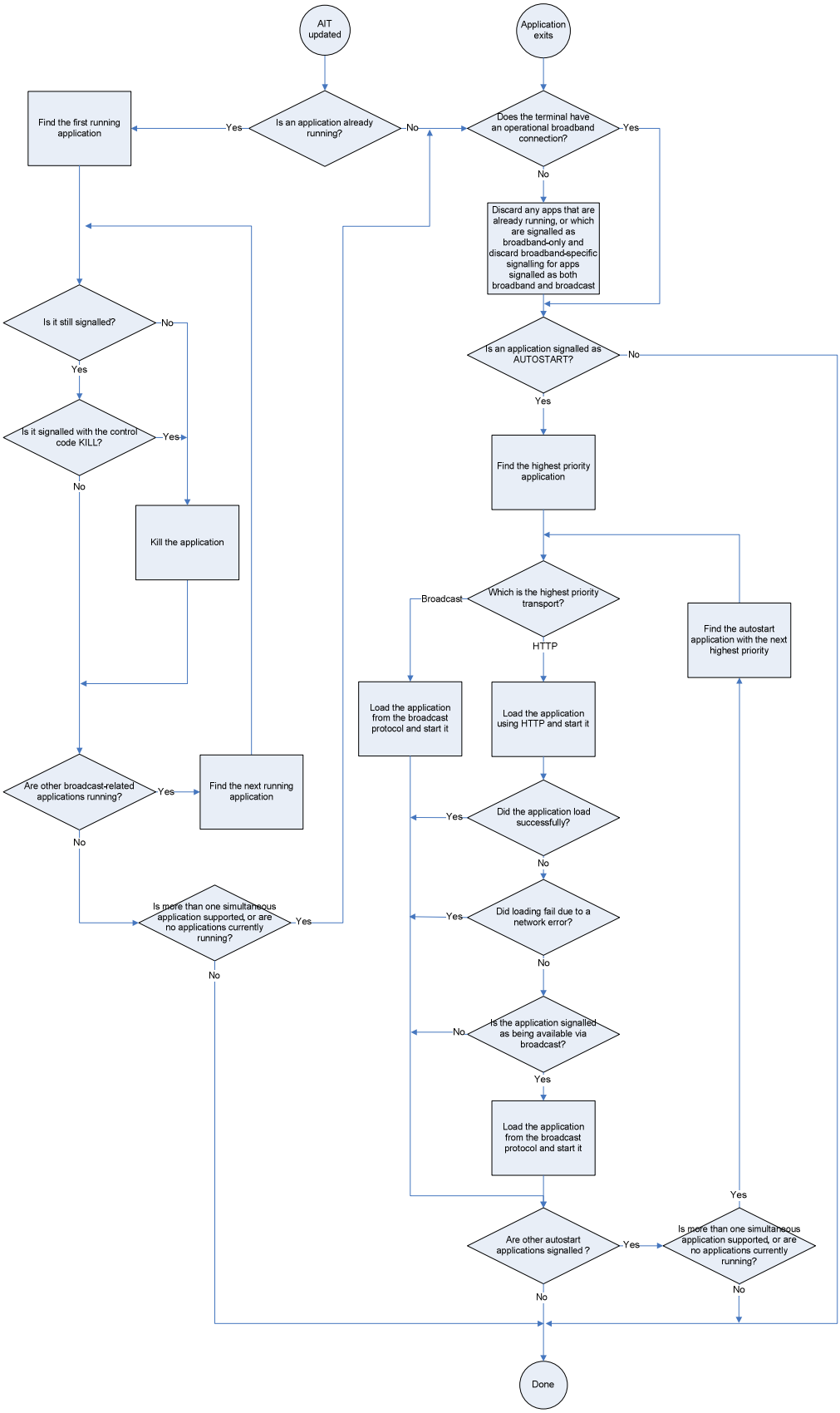


Figure 7: Behaviour when the application signalling for the currently selected channel changes or when a running broadcast-related application exits

5.2.4 Service provider related applications

5.2.4.1 Signalling

Service providers may signal service provider related applications as part of their SD&S service provider discovery record (see Section 3.2.3 of [OIPF_META2], also Sections 4.2.3 and 5.4.3.3 of [TS 102 809] where they are referred to as “unbound applications”). Service provider related applications may either be directly included in the SD&S service provider discovery record or included in an application discovery record which is referenced from the service provider discovery record.

Service providers MAY label one of the applications in their SD&S service provider discovery record using the application usage values defined in Section 3.2.3.3.3 of [OIPF_META2] as follows;

- A service discovery application using the ApplicationUsage identifier “urn:oipf:cs:ApplicationUsageCS:2009:servicediscovery”. An application labelled in this way SHOULD be the highest priority AUTOSTART application signalled.
- An EPG application using the ApplicationUsage identifier “urn:oipf:cs:ApplicationUsageCS:2009:epg”.
- A VoD application using the ApplicationUsage identifier “urn:oipf:cs:ApplicationUsageCS:2009:vod”.
- A communication service application using the ApplicationUsage identifier “urn:oipf:cs:ApplicationUsageCS:2009:communication”.
- An application implementing non-native HNI-IGI using the ApplicationUsage identifier “urn:oipf:cs:ApplicationUsageCS:2009:hni-igi”.

5.2.4.2 Starting

Service provider related applications are started under the following circumstances;

- When a service provider is selected, the OITF SHALL start the AUTOSTART applications signalled by that service provider starting with the highest priority one working downwards in priority while resources in the OITF permit.
- By the end-user using a mechanism provided by the OITF.
- By other service provider related applications.

The OITF SHALL include a mechanism to show the service discovery application and MAY include mechanisms to show the EPG, VoD and the communication service applications. These mechanisms;

- SHALL load the application into the browser if not already loaded.
- SHALL show this application to the end-user.
- SHALL work at all times when the currently selected service provider has an application labelled in this way.

It is outside the scope of this specification whether these mechanisms are buttons on a remote control, items in an on-screen menu or something else. If a button is used, this mechanism SHALL work regardless of which application has focus and the key event corresponding to the button used SHALL NOT be delivered to DAE applications.

5.2.4.3 Stopping

In addition to what is stated in 5.2.2, service provider related applications are stopped when

- The service provider selection process is re-run and a different service provider is selected.
- The selected service provider updates the list of applications in their SD&S service provider discovery record, an application is removed and the OITF detects this update (see Section 4.1.1.3 of [OIPF_META2]).

5.2.5 Broadcast independent applications

Applications which are independent of both broadcasters and the currently selected service provider are started and stopped as described in Section 5.2.2 above. They do not require any signalling. If they are signalled then this shall be

done using the XML encoding of the AIT as defined in Section 5.4 of [TS 102 809]. The XML file shall contain an application discovery record containing exactly one application. The XML file shall be delivered with HTTP or HTTPS using the "`application/vnd.dvb.ait+xml`" MIME type as defined in Section 5.4 of [TS 102 809].

|

5.2.6 Switching between applications

Two cases of switching between applications are relevant in this specification;

- Switching between visible applications and invisible ones.

  NOTE: Switching between a visible application and an invisible one is conceptually a little like changing between tabs in a PC browser however without any implication of a particular user interface.

- Switching between simultaneously visible applications where this OPTIONAL feature is supported.

A number of possible mechanisms exist for switching between visible applications and invisible ones. Some examples include the following;

- Hard coded mechanisms in the terminal for switching to a specific application (e.g. to the service discovery application, the content guide, the communication service application).
- An OPTIONAL terminal specific UI showing available DAE applications which the user can switch to.

5.2.7 Signalling format

5.2.7.1 XML Encoding

The following table defines how the signalling defined in [TS 102 809] SHALL be interpreted when used to signal DAE applications.

Table 2: Application signalling

Descriptor or Element	Summary	Status in this specification
5.4.4.1 ApplicationList	List of applications	Required
5.4.4.2 Application	Name, identifier, type specific descriptor	Required
5.4.4.3 ApplicationIdentifier	2 numbers	Required
5.4.4.4 ApplicationDescriptor	Numerous application attributes	Required The serviceBound element is only applicable to broadcast related applications and SHALL be ignored for other applications.

Descriptor or Element	Summary	Status in this specification
5.4.4.5 VisibilityDescriptor	Attribute – indicate if application can be visible to users and/or other applications	Optional. If this element is not present, OITFs SHALL use a default value of `VISIBLE_ALL`.
5.4.4.6 IconDescriptor	Icon for application	The filename in the IconDescriptor SHALL be an HTTP URL. Use of the icon signaled here by the OITF is OPTIONAL.
5.4.4.7 AspectRatio	Preferred aspect ratio for icons	Only relevant if the OITF uses the IconDescriptor.
5.4.4.8 MhpVersion	Specification version	As defined in Section 3.2.3.3.2 of [OIPF_META2].
5.4.4.9 StorageCapabilities	Can the application be stored or cached	Ignored
5.4.4.10 StorageType	Enumeration used in 5.4.4.9	As 5.4.4.9
5.4.4.11 ApplicationType	Application type	For DAE and PAE applications, the appropriate value from the ApplicationTypeCS scheme from [OIPF_META2] SHALL be used.
5.4.4.12 DvbApplicationType	Enumeration for 5.4.4.11	Ignored
5.4.4.13 ApplicationControlCode	Enumeration for 5.4.4.4.	See below
5.4.4.14 ApplicationSpecificDescriptor	Container	Ignored
5.4.4.15 AbstractIPService	Supports grouping of unbound applications	Only one group SHALL be signalled
5.4.4.16 ApplicationOfferingType	Used as part of application discovery record	Required

Descriptor or Element	Summary	Status in this specification
5.4.4.17 ServiceDiscovery	Used as part of application discovery record	Required
5.4.4.18 ApplicationUsageDescriptor	Indicates that an application provides a specific service	Required
5.4.4.19 TransportProtocolDescriptorType	Abstract base type	Required
5.4.4.20 HTTPTransportType	Type for applications accessed by HTTP	Required
5.4.4.21 OCTransportType	Type for applications accessed by DSM-CC object carousel	Ignored
5.4.4.22 ComponentTagType	Encodes a DVB component tag	Ignored
5.4.4.23 SimpleApplicationLocationDescriptorType	Encodes the location of the start page of an application relative to one of the transport types.	Required
5.4.4.24 SimpleApplicationBoundaryDescriptorType	Encodes an application boundary.	Required
FLUTESessionDescriptor as defined by Section B.13 of [OIPF_META2]	Support for distributing applications through multicast.	SHALL be supported if OITFs support FLUTE.

Elements and descriptors marked as ‘Ignored’ SHALL NOT be processed for DAE applications. Servers MAY include these in application signalling.

The application control code SHALL be interpreted as follows for DAE applications:

Value	Description
AUTOSTART	The application is eligible to be started automatically. Sections 5.2.3.2 and 5.2.4.1 above define the order in which AUTOSTART applications are started if more than one is signalled.
PRESENT	The OITF SHALL take no action. The OITF MAY provide a mechanism to allow the end-user to start applications signalled as PRESENT. However since there is no requirement for such a mechanism, an IPTV service provider who signals applications with this control code SHALL provide an application able to start them.
KILL	The application SHALL be terminated (see ApplicationDestroyRequest in Section 7.2.6).
PREFETCH	The OITF MAY start fetching files, data or other information needed to start the application but SHALL NOT start the application. Implementations MAY consider this control code to be the same as PRESENT.

Table 3: DAE application control codes

The other control codes from [TS 102 809] are not defined for DAE applications. Other control codes are not required to be supported but MAY be supported if required by another specification. The OITF SHALL discard any AIT entry containing an unsupported control code.

5.2.7.2 MPEG-2 Encoding

In a hybrid device where the broadcast channel is based on DVB network technologies and uses DVB-SI as specified in [EN300468], the OITF SHALL support the MPEG-2 encoding of the AIT from [TS 102 809] as defined in the following table. This encoding MAY be supported in other devices.

Section	Status	Notes
5.2.2 Application types	M	The application type shall be 0x0011.
5.2.3 Application identification	M	Applications which only need the default permissions SHALL be signalled using application_ids from the range for unsigned applications. Applications which need more permissions than the default SHALL be signalled using application_ids from the range for signed applications. The range of application_ids for privileged applications SHALL NOT be used.
5.2.4 Application control codes	M	The following control codes shall be supported: 0x01 AUTOSTART 0x02 PRESENT 0x04 KILL 0x07 DISABLED

		The application life cycle shall follow the rules defined in TS 102 809 [TS 102 809] and in this specification.
5.2.5 Platform profiles	M	The encoding of the `application_profile` is not defined in this specification. The version fields shall be set as follows: version.major = 2 version.minor = 1 version.macro = 0
5.2.6 Application visibility	M	
5.2.7 Application priority	M	
5.2.8 Application icons	O	The icon locator information shall be relative to the base part (constructed from the URL_base_bytes) of the URL as signalled in the transport_protocol_descriptor.
5.2.9 Graphics constraints	-	
5.2.10 Application usage	M	Usage type 0x01 shall be supported as described in Section 5.2.3.1 of the present document.
5.2.11 Stored applications	-	
5.2.12 Application Description File	-	
5.3.2 Program specific information	M	
5.3.4 Application Information Table	M	See [AVC] for MPEG-2 system related requirements and constraints.
5.3.5.1 Application signalling descriptor	M	
5.3.5.2 Data broadcast id descriptor	O	The value to be used for the data_broadcast_id field of the data_broadcast_id_descriptor for OIPF carousels shall be 0x0150. By supporting this optional feature, terminals can reduce the time needed to mount a carousel.
5.3.5.3 Application descriptor	M	
5.3.5.4 Application recording descriptor	-	

5.3.5.5 Application usage descriptor	M	Usage type 0x01 shall be supported as described in Section 5.2.3.1.
5.3.5.6 User information descriptors	M	
5.3.5.7 External application authorization descriptor	M	
5.3.5.8 Graphics constraints descriptor	-	
5.3.6 Transport protocol descriptors	M	The following protocol_ids shall be supported: 0x0001 object carousel over broadcast channel (as defined in [AVC]) 0x0003 HTTP over broadband connection
5.3.7 Simple application location descriptor	M	
5.3.8 Simple application boundary descriptor	M	Only strict prefixes starting with "dvb://", "http://" or "https://" shall be supported. Only prefixes forming at least a second-level domain shall be supported. Path elements shall be ignored.
5.3.9 Service information	-	
5.3.10 Stored applications	-	

Table 4: Supported application signalling features

Status	Description
M	MANDATORY The signalling may be restricted to a subset specified in the "Notes" column. In that case all additional signalling is optional.
O	OPTIONAL
-	NOT INCLUDED The referenced signalling is not included in this specification.

Table 5: Key to status column

5.2.8 Widgets lifecycle

As Widgets are packaged as ZIP archives, they only require a single download and installation on an OITF before being executed. Widgets can also be downloaded over non-HTTP distribution channels and even over off-network channels (USB drives, CD/DVD, etc.).

The Widget lifecycle has 3 main steps:

1) Installation: The Widget is installed on the OITF

2) Execution: The Widget is executed (end eventually stopped)

3) Removal: The Widget is uninstalled from the OITF

Step 1, installation, is only needed before the first execution of the Widget or if its version is obsolete and the user or the OITF want to update it (see Section 5.2.8.4).

Step 2, execution, may be performed at any time after the Widget has been installed. It can be triggered by an action from the user, or it may be done automatically by the OITF either through a DAE application or a native application in the OITF. Note that it is not possible to have two running instances of a single Widget simultaneously.

Step 3, removal, is performed if the user wants to uninstall the Widget from the OITF. An uninstalled Widget needs to be reinstalled by a user to be executed again.

Detail descriptions of each step above are provided in the following sections.

5.2.8.1 Widget installation

In order to be able to execute a Widget, we first have to acquire a Widget package and install it on the OITF. Steps for acquiring and processing a Widget package and associated processing rules are described in Section 9 of [Widgets-Packaging]. In this specification the expression "Widget installation succeed" means that the afore-mentioned procedure is completed successfully.

Although [Widgets-Packaging] does not limit or mandate any specific data transfer protocol or distribution channel through which Widgets are delivered, an OITF SHALL support the use of HTTP and HTTPS as the transfer protocols. Support for other transfer protocols is OPTIONAL. Widget installation is done through the `ApplicationManager.installWidget()` API call. After a call to this function, if the installation succeeds, the installed Widget SHALL be available in the list of installed Widgets that can be retrieved using `ApplicationManager.widgets`. The application installing the Widget is notified about the installation success/failure through the "WidgetInstallation" event as specified in Section 7.2.1.2 and 7.2.1.4.

When installing a Widget, the OITF SHOULD notify the user if there is already an installed Widget with the same "id" value (where "id" is defined in Section 7.6.1 of [Widgets-Packaging] along with the extension defined in Section 11.1 of this specification). In this case the OITF SHALL proceed as specified in the description of `installWidget()` method in Section 7.2.1.3.

5.2.8.2 Widget execution

In order to be executed, a Widget needs to be installed as described in the previous section. After the installation, a Widget can be started either using the `Application.createApplication()` API call or through the `ApplicationManager.startWidget()` API call. The behaviour of these two methods is equivalent. `startWidget()` is the preferred method; `createApplication()` is kept for consistency with other DAE applications. A list of installed Widgets can be retrieved using `ApplicationManager.widgets`. Note that only one running instance per Widget at time is allowed. A Widget can be stopped using `Application.stopWidget()` or `Application.destroyApplication()`. `stopWidget()` is the preferred method; `destroyApplication()` is kept for consistency with other DAE applications.

If the installed Widget has been run on the OITF before, any "storage areas" associated with the Widget, as defined in [Widgets-APIs], SHALL be restored. Saved data is accessible through the preferences attribute of the Widget object as defined in Section 11.3 of this specification.

See related sections in Section 7 for more details about the above mentioned API calls.

5.2.8.3 Uninstalling a Widget

An installed Widget can be uninstalled from an OITF through the `ApplicationManager.uninstallWidget()` API call. Calling this method on a running Widget will cause the Widget to be stopped before the Widget is uninstalled. The application uninstalling the Widget is notified about the uninstallation success/failure through the "WidgetUninstallation" event as specified in Section 7.2.1.2 and 7.2.1.4. Any storage areas associated with the uninstalled Widget SHALL be deleted.

5.2.8.4 Widget updates

An installed Widget can be updated by installing a new version of it.

5.3 Event Notifications

This section describes 4 different notification frameworks (In-session notification based on the home network domain, In-session notification based on the Internet domain, 3rd Party notification based on home network domain, and 3rd Party notification based on the Internet domain) defined by [CEA-2014-A]. Moreover, it defines a new notification framework for IMS based notifications such as CallerID, Incoming Call Message, Chat Invite not only when a DAE application is active but also inactive.

The event notification mechanism allows OITFs to receive important UI updates or information from IPTV service provider or home network devices such as IG, AG or DLNA RUI compatible devices. CEA 2014 mandates 4 unique notification models which are dependent on whether the server exists on the internet domain or home network domain. Each of these domain models have two unique scenarios depending on whether or not a DAE application is running. If a DAE application is active, the in-session notifications are used to support dynamic UI interaction between the server and the DAE application without the need to reload the XHTML page. Otherwise, 3rd party event notifications should be used to receive and display a notification message outside of the current user session with a DAE application on the OITF, for example an event coming from another server, e.g. to receive emergency alerts, or events regarding news, weather, stock or other information. Generally, 3rd party event notification creates a new DAE application to display notification information.

IMS event notifications for Caller ID, Messaging and Chatting have different behavior from general event notification defined by [CEA-2014-A] because IMS communication service should be accessed by authorized users and devices within the approval of IPTV service provider. Considering the issue of user's privacy, the DAE specification not only adopts the general Event Notification Frameworks from [CEA-2014-A] as defined in Section 5.3.1, but also defines a new IMS Event Notification Framework in Section 5.3.2.

5.3.1 Event notification framework based on CEA 2014

An OITF must be capable of displaying various event notifications from both Internet domain and home network domain. Event notification can be conveyed through active UI interaction's channel or out of session. As described in the diagram below, in-session notification is associated with a running DAE application, whereas a 3rd party event notification is delivered through an independent communication channel. If an OITF receives a 3rd party event after subscribing to a certain internet url or the OITF receives a multicasted event notification message, the OITF needs to perform 3rd party event notification and display its information inside a new DAE application.

The diagram below describes a general overview of the Event Notification architecture.

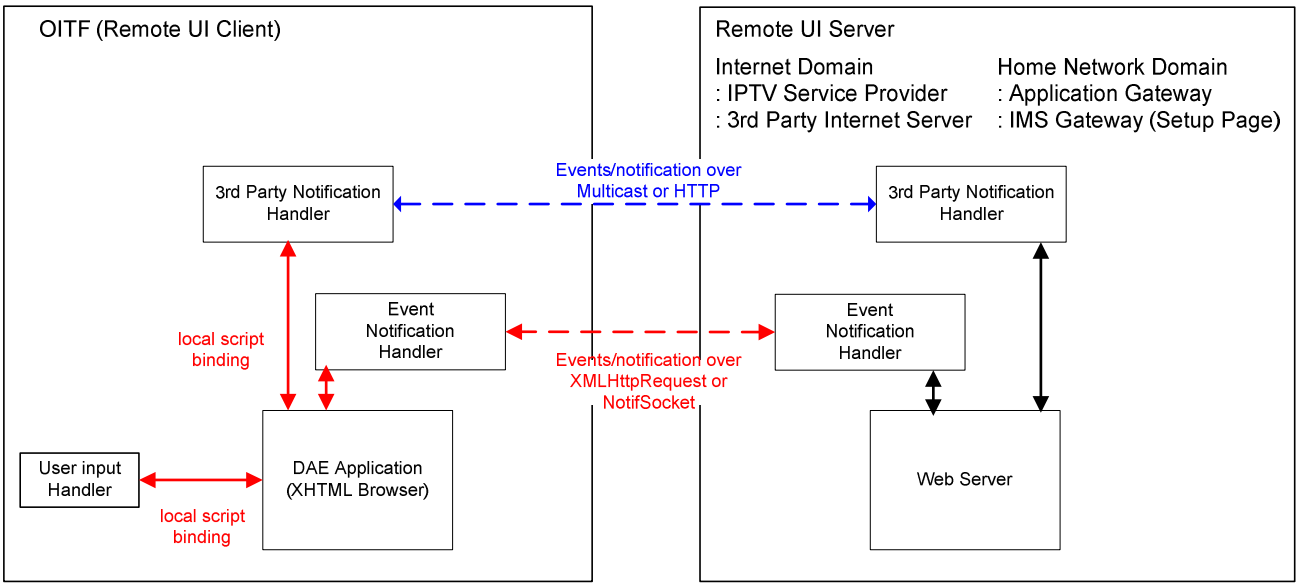


Figure 8: General Event Notification Architecture on OITF and Remote UI Server

In-Session notifications are performed to update partial or whole DAE application UI through the `NotifSocket` object and/or the `XMLHttpRequest` object as defined by [CEA-2014-A]. `NotifSocket` object creates a persistent TCP connection between a DAE application and Remote UI server in order to support burst event notifications. In addition, a DAE application can create an `XMLHttpRequest` object to make asynchronous HTTP requests to a web server on the internet domain. This establishes an independent HTTP connection channel to support XML updates between the DAE application and the Remote UI server.

On the other hand, if the OITF receives an incoming notification outside of an active interaction (i.e. session) with the server, a 3rd Party Event Notification must be executed to invoke a DAE application to fetch and render the UI content using the url contained within the notification message. This allows servers to "broadcast" important messages, such as Emergency alert messages, to an OITF at anytime, even when the DAE application would currently not be running. This should be done through a push-method with multicast message for the home network domain. and a pull-method for the internet case.

The next two subsections describe the requirements for the event mechanisms in more detail.

5.3.1.1 In-session event notification

In-Session notification can be defined as "Dynamic UI Update." With this mechanism, a server should be able to send a notification message during a UI interaction to update the UI dynamically without the need to reload the XHTML-page. The OITF SHALL support the two following scripting objects for In-session event notification:

- `XMLHttpRequest` Scripting Object (as defined in Section 5.5.2 of [CEA-2014-A])
  - The `XMLHttpRequest` is an embedded object on the browser and enables scripts to make HTTP request to a web server without the need to reload the page. It can be used by ECMAScript to transfer and manipulate XML data to and from a web server using HTTP, establishing an independent connection channel between a web server and DAE applications. Whenever a DAE application needs to update the UI, it sends a request to the UI server, IPTV service provider or 3rd Party Internet Server, to monitor the change of status or event. In case an event, the UI server sends an HTTP response to the XMLHttpRequest.
- `NotifSocket` Scripting Object (as defined in Section 5.5.1 of [CEA-2014-A])
  - Even though `XMLHttpRequest` object has become more widespread on browsers and Internet Portal servers, it has a difficulty in supporting dynamic UI update on home domain's devices because it is required to be invoked by the request of `XMLHttpRequest` on DAE application side. `NotifSocket` creates a persistent TCP connection between DAE application and UI server in order to support burst event notifications. Whenever the UI server needs to notify the DAE application running on the OITF of a UI update, it sends any types of update message, such as encoded binary or string, through the `NotifSocket` connection. The `NotifSocket` object allows an UI server to push any event information through the independent TCP/IP channel at any time.

5.3.1.2 Out of session event notification

Out of session event notifications are defined as "3rd Party Notifications" in CEA 2014. Since these notifications are not part of an active remote UI interaction with a Remote UI Server, the OITF must launch a new DAE application to render the UI content using the url contained within the notification message.

The OITF SHALL support multicast notifications for 3rd party event notifications for the home network domain and the internet domain respectively as defined below. Support for polling-based notifications as defined below is OPTIONAL and support can be indicated through the OITF's capability description by using element <pollingNotifications> as defined in Section 9.3.14 or the +POLLNOTIF name fragment as defined in Section 9.2.

- Multicast Notifications (as defined in Section 5.6.1 of [CEA-2014-A])
  - The OITF SHALL support receiving of Multicast Notifications over multicast UDP, with a UPnP event message format defined by CEA 2014 if the incoming message comes from home network domain. After interpreting the message, the OITF should create a new notification window with specified <`ruiEventURL`>. In order to ensure a reliable transmission of a multicast notification message, a Remote UI Server shall transmit the same notification message, with the same HTTP SEQ header value 2 or 3 times, where the time between transmissions should be a random time between 0 and 10 seconds.
- Polling-based Notification (as defined in Section 5.6.2 of [CEA-2014-A])
  - The OITF SHALL support polling-based 3rd Party notifications from an IPTV Service Provider or a 3rd Party Internet Server. To this end, the OITF subscribes to certain URIs to display web contents such as news, weather, stock or other information from Internet side on executing the `subscribeToNotifications()` method. An OITF should poll for notifications even when the CE-HTML browser is not active. If a new notification is received, this MAY be notified to the user in a vendor defined way, including direct rendering on the display and using a non-intrusive prompt.

    Note that in Annex B we have defined a `subscribeToNotificationsASync()` method to provide a way of subscribing to polling-based notifications that is non-blocking.

5.3.2 IMS event notification framework

This section covers the DAE interactions needed to drive the message exchanges on the HNI-IGI interface in the case where the Service Provider offers an IMS application.

The HNI-IGI framework defines how an OITF interacts with an IMS Gateway (IG) via the HNI-IGI interface ([OIPF_PROT2] Section 5.5.1).

Every message on the HNI-IGI interface SHALL be carried in a HTTP transaction where the OITF sends the HTTP request and the IG responds to the request. The HNI-IGI In-session framework, in the case of a DAE application, uses the XMLHttpRequest Script Object, as defined in Section 5.5.2 of [CEA-2014-A] .

There are two message directions on the HNI-IGI interface, corresponding to outgoing and incoming messages from and to the OITF.

5.3.2.1 HNI-IGI transactions for in-session out-going request messages

This message direction applies to outgoing messages from the OITF on the HNI-IGI interface. The OITF sends a request and the IG responds to the request. The following figure illustrates the sequences for in-session transactions for outgoing requests from DAE application to the IG.

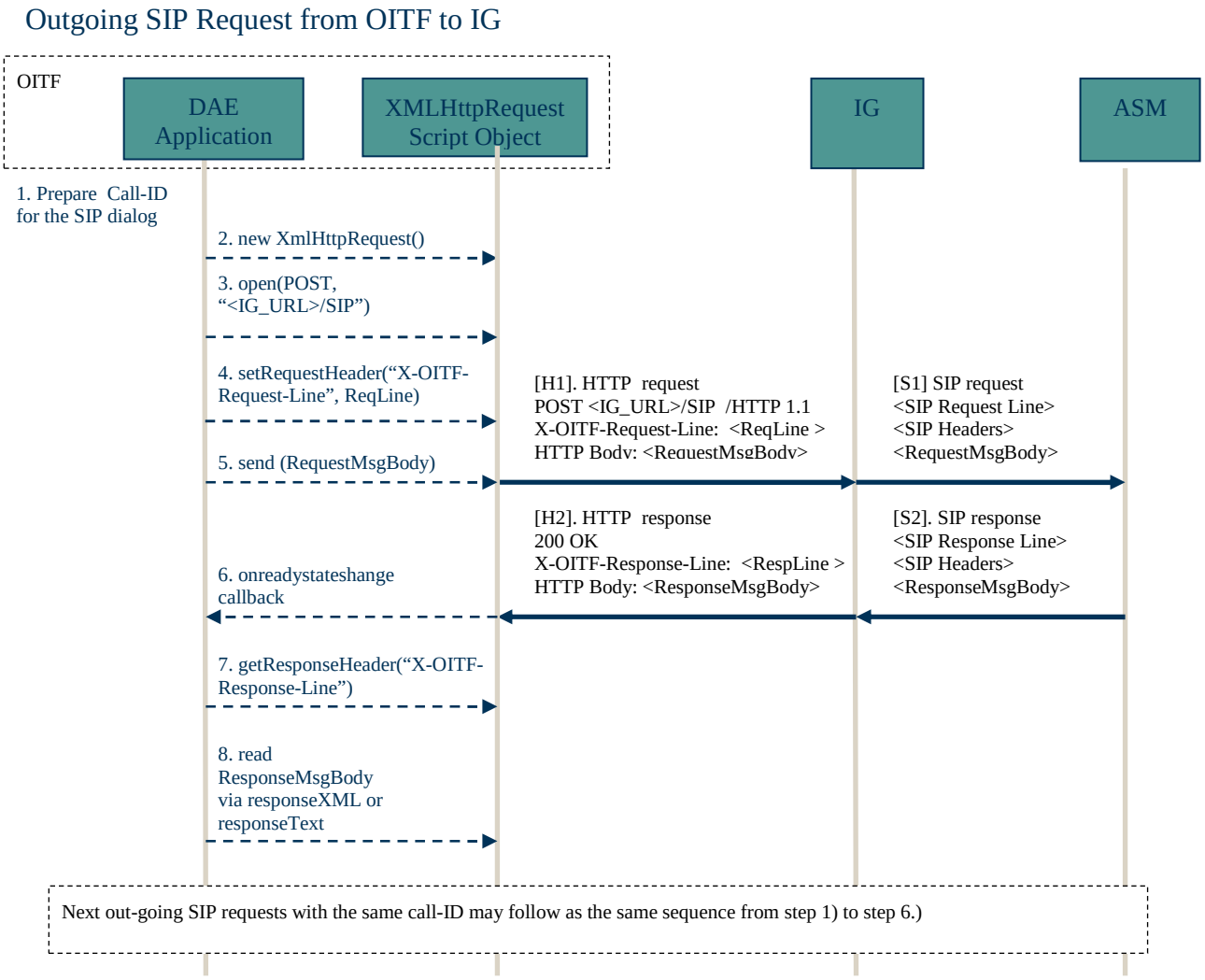


Figure 9: HNI-IGI transaction for outgoing SIP requests from a DAE application

1. Prepare the Call-ID for a SIP request. The Call-ID SHALL be generated by the DAE application for an outgoing SIP request. This Call-ID SHALL be locally unique across all OITFs in a residential network.
   NOTE: How uniqueness is achieved is currently not defined.

2. The DAE application SHALL create a new `XMLHttpRequest` object using the constructor "`new XMLHttpRequest()`".

3. The DAE application SHALL invoke the `open()` method to specify the HTTP method and Request-URI for the request. In this case, the HTTP POST method with the Request-URI of <IG_URL>/SIP SHALL be used as specified in [OIPF_PROT2].

4. The DAE application SHALL invoke the `setRequestHeader()` method to specify the required HTTP headers as specified in [OIPF_PROT2]. This method SHALL be invoked for each required HTTP header. For example, the X-OITF-Request-Line HTTP header specifies the SIP request line for the SIP request. The Call-ID is specified in the X-OITF-Call-ID header.

5. The DAE application SHALL invoke the `send()` method to send the HTTP request. The SIP Message Request body is specified in a parameter of this method.

6. When the HTTP response is received, the `onreadystatechange` callback function SHALL be invoked on the DAE application.

7. The DAE application SHALL invoke the `getRequestHeader()` method to retrieve each HTTP header. The SIP Response Line is specified in the X-OITF-Response-Line header.

8. If the `readyState` property of the `XMLHttpRequest` object has the value 4, the HTTP response body SHALL be retrieved via the `responseXML` or `responseText` properties of the `XMLHttpRequest` object. The SIP response body is specified in the HTTP response body.

5.3.2.2 HNI-IGI transaction for in-session incoming request messages

This message direction applies to incoming messages to the OITF on the HNI-IGI interface which are related to an existing IMS session. An example of this is a SIP NOTIFY message received from the network in response to a previous SIP SUBSCRIBE sent from the IG. The OITF sends a HTTP request and the IG responds to the request when it receives an incoming message from the network related to an existing session. The following figure illustrates the sequences for in-session transactions for incoming requests from the IG to the DAE application.

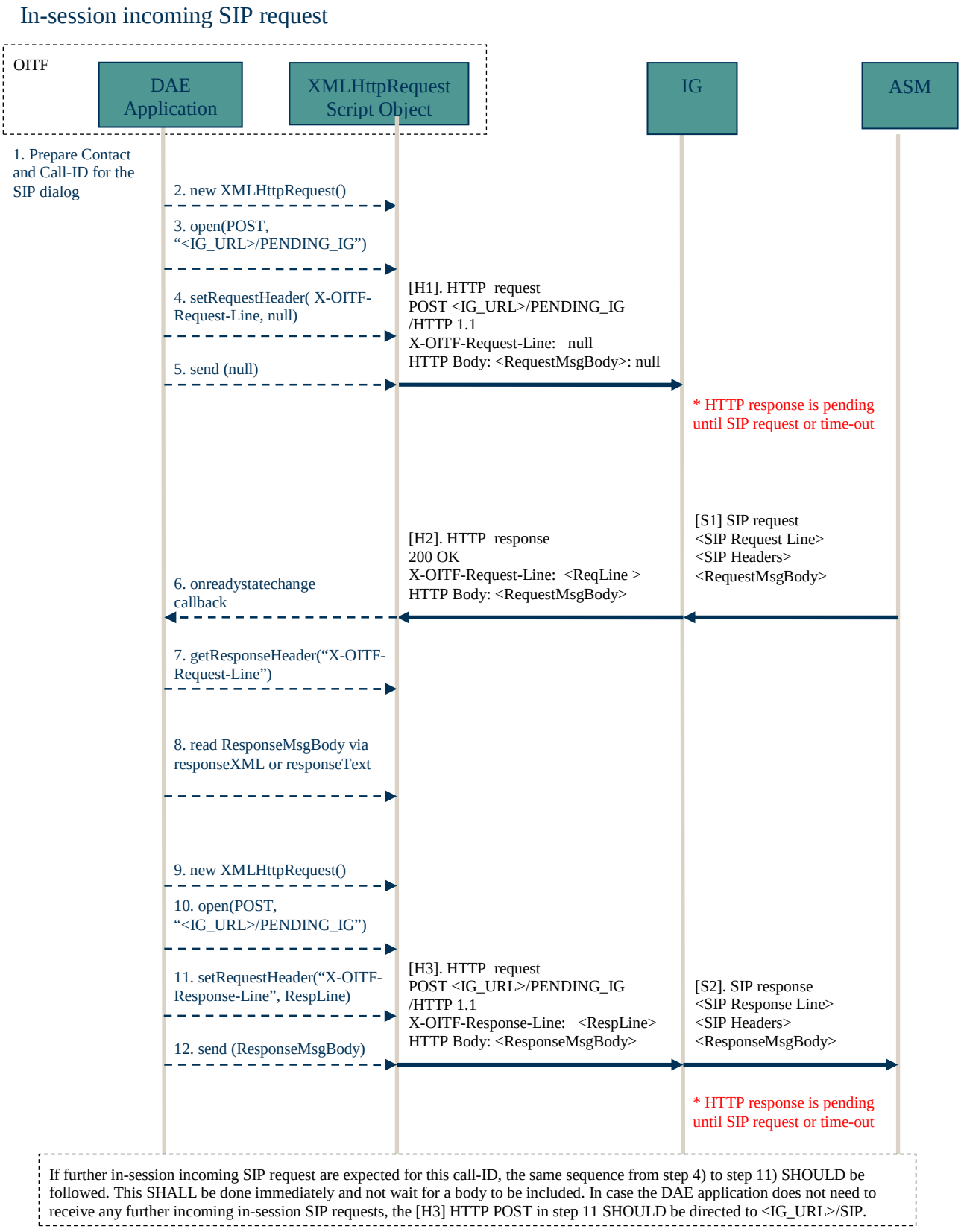


Figure 10: HNI-IGI transaction for in-session incoming SIP request

1. Prepare the Call-ID for this SIP session for which a message is expected. The Call ID SHALL be the same as the one created initially for this session.

2. The DAE application SHALL create a new XMLHttpRequest object using the constructor "`new XMLHttpRequest()`".

3. The DAE application SHALL invoke the `open()` method to specify the HTTP method and the Request-URI for the request. In this case, the POST method with a Request-URI of <IG URL>/PENDING_IG SHALL be used as specified in [OIPF_PROT2].

4. The DAE application SHALL invoke the `setRequestHeader()` method to specify the required HTTP headers, as specified in [OIPF_PROT2]. This method is invoked for each HTTP header that is required. In this case, the X-OITF-Request-Line, which specifies the SIP request line for the SIP request, is set to the value `null`. The SIP Call-ID is specified in the X-OITF-Call-ID header.

5. The DAE application SHALL invoke the `send()` method to send the HTTP request. For the HTTP request that sets up the initial long poll, no X-OITF headers are allowed for the HTTP request to the PENDING_IG Request-URI.

6. When the HTTP response is received, the specified `onreadystatechange()` callback function is invoked.

7. The DAE application SHALL invoke the `getResponseHeader()` method to retrieve each HTTP header. The SIP Request Line is specified in the X-OITF-Request-Line HTTP header.

8. If the `readyState` property of the `XMLHttpRequest` object has the value 4, the HTTP response body SHALL be retrieved via the `responseXML` or `responseText` properties of the `XMLHttpRequest` object. The SIP response body is specified in the HTTP response body.

9. The DAE application SHALL create a new `XMLHttpRequest` object using the constructor "`new XMLHttpRequest()`".

10. The DAE application SHALL invoke the `open()` method to specify the HTTP method and the Request-URI for the request. In this case, the POST method with a Request-URI of <IG URL>/PENDING_IG SHALL be used as specified in [OIPF_PROT2].

11. The DAE application SHALL invoke the `setRequestHeader()` method to populate each HTTP header as specified in [OIPF_PROT2]. This method SHALL be invoked for each required HTTP header. For example, the X-OITF-Response-Line specifies the SIP response line for the SIP response. The Call-ID is specified in the X-OITF-Call-ID header.

12. The DAE application SHALL invoke the `send()` method to send the HTTP request. If there is a SIP response body, it is included as a parameter to the `send()` method. The SIP response body message is carried in the HTTP body for the HTTP request to the PENDING_IG Request-URI.

In the case where the OITF does not need to receive any further incoming in-session SIP requests, the [H3] HTTP POST in step 11 SHALL be directed to the <IG_URL>/SIP Request-URI.

5.3.2.3 HNI-IGI transaction for out of session incoming request messages

This message direction applies to incoming messages on the HNI-IGI interface which are not related to an existing session. An example of this is a SIP MESSAGE message received from the network, coming e.g. from an IPTV application or from another user. The following figure illustrates the sequences of out-of-session transactions for incoming requests from the IG to OITF.

Figure 11 describes what happens when the OITF is first turned on.

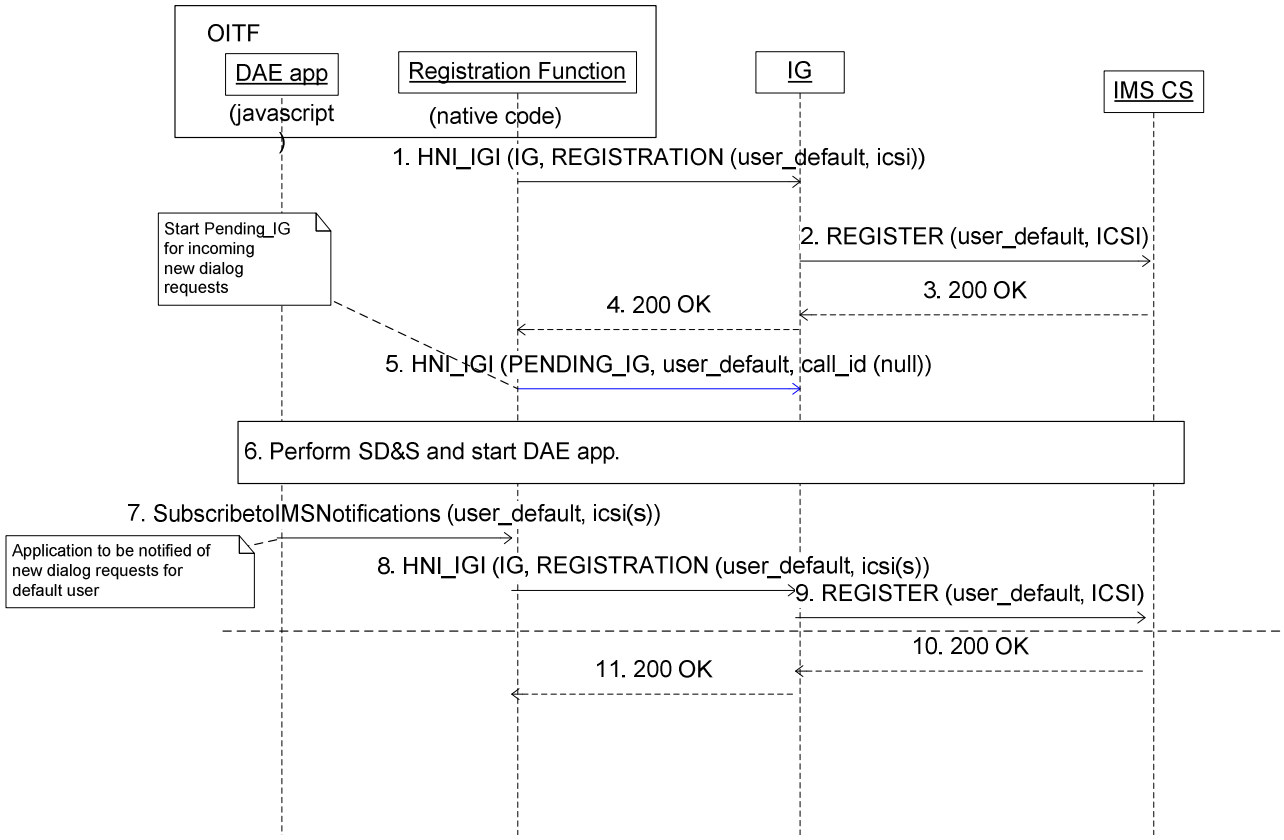


Figure 11: What happens when the OITF is first turned on

1) When the OITF is turned on the OITF SHALL send a HNI_IGI IG registration message to register the default user.

2) The IG Registers the default user in the IMS network.

3) The IMS network returns 200 OK.

4) a 200 OK message SHALL be returned on the HNI_IGI.

5) If there are native IMS applications that may receive unsolicited messages the OITF SHALL send a PENDING_IG message to the IG, for the default user and with the call_id set to `null`. The steps to send PENDING_IG are the same as steps 8-11 from Section 5.3.2.2.

6) The OITF performs service selection and discovery and loads the initial DAE page.

7) DAE IMS applications that desires to receive unsolicited notifications SHALL issue a `subscribetoIMSNotifications()` method (as defined in Section 7.8).

8) When applicable the OITF SHALL send a HNI_IGI IG registration message to re-register the default user, including new applications.

9) The IG re-registers the default user in the IMS network.

10) The IMS network returns 200 OK.

11) A 200 OK message SHALL be returned on the HNI_IGI.

Figure 12 describes what happens when a specific user logs in using the DAE interface.

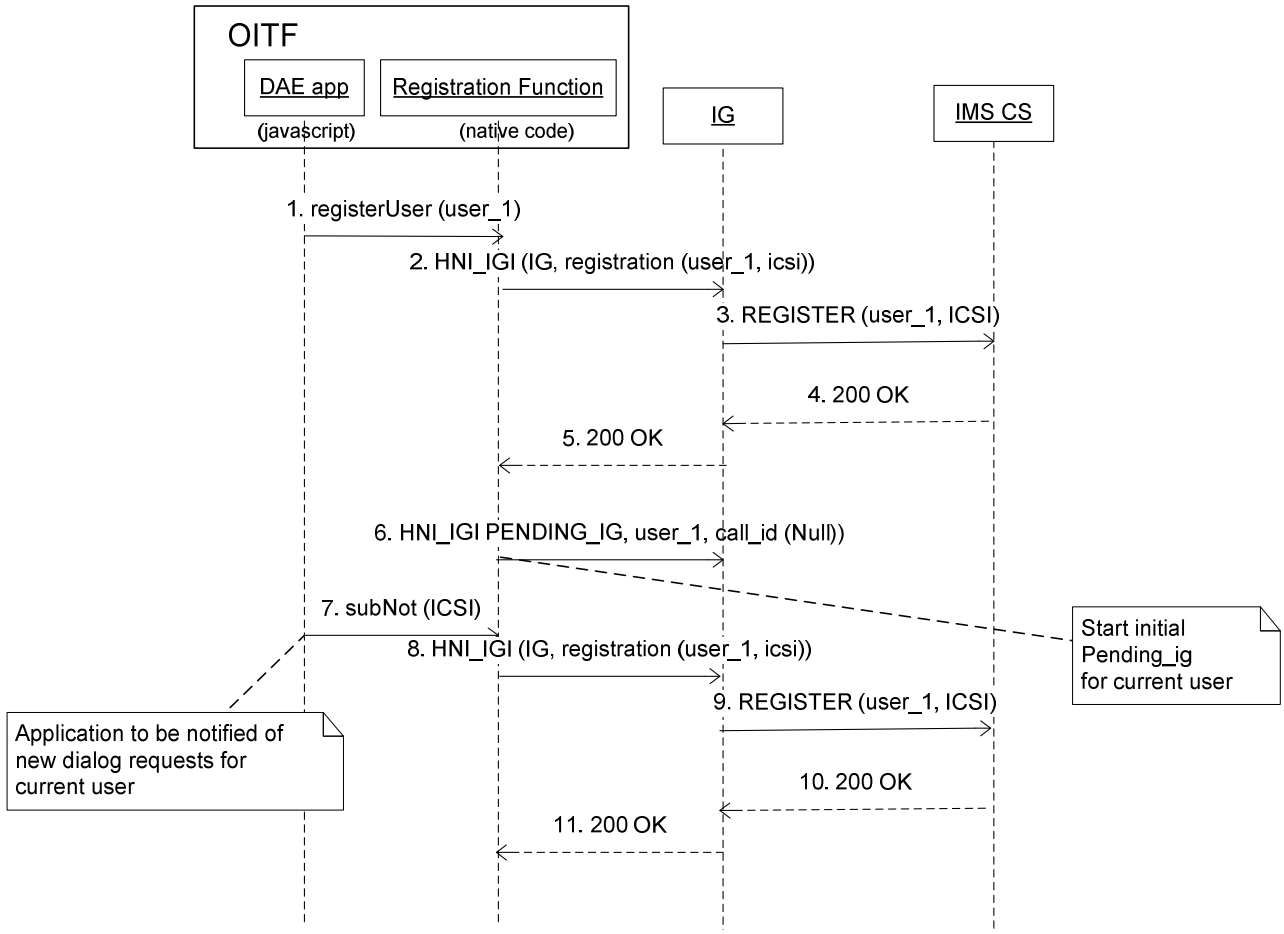


Figure 12: User logs in using the DAE interface

1) When the user desires to login the DAE SHALL call the registerUser() method to register the user.

2) The OITF SHALL send a HNI_IGI IG registration message to register the user.

3) The IG Registers the user in the IMS network.

4) The IMS network returns 200 OK.

5) A 200 OK message SHALL be returned on the HNI_IGI.

6) If there are native IMS applications that may receive unsolicited messages the OITF SHALL send a PENDING_IG message to the IG, for the default user and with the call_id set to `null`. The steps to send PENDING_IG are the same as steps 8-11 from Section 5.3.2.2.

7) DAE IMS applications for the user that desires to receive unsolicited notifications SHALL issue a `subscribetoIMSNotifications()` method (as defined in Section 7.8).

8) When applicable the OITF SHALL send a HNI_IGI IG registration message to re-register the user, including new applications.

9) The IG re-registers the default user in the IMS network.

10) The IMS network returns 200 OK.

11) a 200 OK message SHALL be returned on the HNI_IGI.

Figure 13 describes what happens when an unsolicited message arrives from the network. The precondition is that a DAE application is already running and subscribed to the IMS notifications (refer to previous sequence when user logs in).

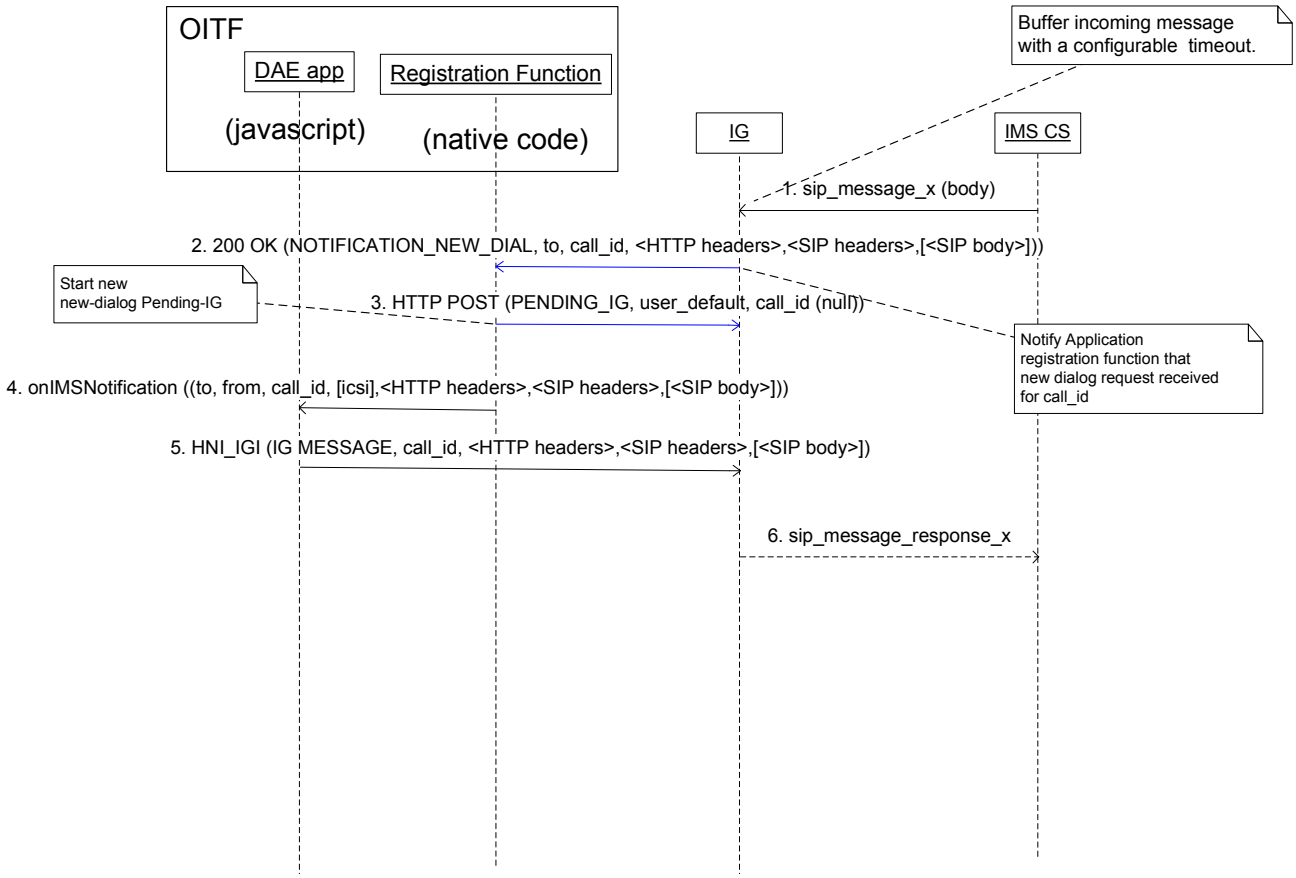


Figure 13: Unsolicited message from the network

1) A SIP message arrives from the network.

2) The IG responds to the PENDING_IG request.

3) The OITF SHALL immediately issue a new PENDING_IG request after receiving a response on a PENDING_IG request. The steps to send PENDING_IG are the same as steps 8-11 from Section 5.3.2.2.

4) The OITF SHALL call the callback function `onIMSNotification` for the corresponding application. This includes the IMS message.

5) The OITF MAY respond to the network with a new outgoing message. The steps to send PENDING_IG are the same as steps 8-11 from Section 5.3.2.2.

6) If the OITF sends a message the IG SHALL forward it to the network.

6 Formats

6.1 CE-HTML

An OITF SHALL support the XHTML profile defined in Section 5.4 of [CEA-2014-A], with the exceptions as defined in Annex B.

NOTE: the list of default embedded objects and related Javascript APIs are defined in Section 7.

6.2 CE-HTML referenced formats

This section provides more details about formats used by CE-HTML

This section modifies the sections of [CEA-2014-A] which reference externally defined formats. In the absence of modifications below, those sections SHALL apply.

- **JPEG**: Support for lossless and hierarchical modes and arithmetic coding of DCT coefficients is OPTIONAL. The thumbnail feature of [JFIF] is OPTIONAL. OITFs not supporting thumbnails SHALL skip them if present and continue decoding the rest of the image.

6.3 Media formats

This section describes the main requirements for the format and usage of codecs in media referred to by DAE applications. This section also describes memory audio.

6.3.1 Media format of A/V media except for audio from memory

This section describes the format and usage of the A/V media codec except for audio from memory.

- Format and usage of video codec SHALL adhere to Section 5 of [OIPF_MEDIA2].
- Format and usage of subtitles format SHALL adhere to Section 6 of [OIPF_MEDIA2].
- Format and usage of teletext format SHALL adhere to Section 7 of [OIPF_MEDIA2].
- Format and usage of audio codec SHALL adhere to Section 8 of [OIPF_MEDIA2], except for Section 8.1.1.2, 8.1.5 and 8.2.1 which are covered in Section 6.3.2.

6.3.2 Media format of A/V media for audio from memory

This section describes the format and usage of the A/V media codec for audio from memory. Usage of corresponding A/V media object is described in Section 7.14 of this document.

For the audio from memory format, HE-AAC SHALL be supported by the OITF and WAVE MAY be supported by the OITF.

- Format and usage of HE-AAC audio from memory SHALL adhere to Section 8.1.1.2 and 8.2.1 of [OIPF_MEDIA2].
- Format and usage of WAVE audio from memory SHALL adhere to Section 8.1.5 and 8.2.1 of [OIPF_MEDIA2].

6.3.3 Media transport

Format and usage of media transports referred to by DAE applications SHALL adhere to Section 4 of [OIPF_MEDIA2].

6.4 SVG

This section contains extensions and modifications to [SVG Tiny 1.2] and to [CEA-2014-A].

6.4.1 Supporting SVG documents

OITF SHALL support [SVG Tiny 1.2] documents with the extensions to [CEA-2014-A] described in this subsection. These extensions SHALL be accomplished by means of the following text:

[Req 5.2.1.a] The following extensions apply:

- A Remote UI Client Capability Description SHALL include the following element in order to convey support for SVG:

```
<mime-extensions>image/svg+xml</mime-extensions>
```

[Req 5.2.2.f] The following extensions apply:

- Referenced content SHALL adhere to the `image/svg+xml` MIME type.

[Req. 5.3.a] The following extensions apply:

- If an `Accept` request header is used, then its value SHALL contain the string "`image/svg+xml`".
- If an `Accept-Encoding` and an `Accept` request header are used, then the value of the `Accept-Encoding` header SHALL contain the string "`gzip`" and "`deflate`".

[Req. 5.4.a] The following extensions apply:

- A Remote UI Client SHALL include a *Conforming Dynamic SVG Viewer* as defined by [SVG Tiny 1.2] .

The following applies to item 8):

- Compliant image content SHALL include the MIME type `image/svg+xml` as defined by [SVG Tiny 1.2] .

[Req. 5.10.b] The following extensions apply:

- SVG viewer SHALL support SVG image content which uses logical coordinates greater than the resolution supported by the <width> and <height> parameters of the Remote UI Client capability.

[Annex G, Table 5] The following extensions apply:

- The `type` attribute of an `<a></a>` element tag SHALL specify the value `image/svg+xml` if a link to an SVG document is defined.
- The `<img/>` element tag SHALL allow image of content-type `image/svg+xml` to be used.
- The `<object/>` element tag SHALL allow content of content-type `image/svg+xml` to be used.
- Elements, attributes or properties other than those defined in [SVG Tiny 1.2] MAY be ignored.

6.4.2 Supporting DOM access between CE-HTML and SVG

6.4.2.1 Parent CE-HTML access to child SVG

In order to enable scripts in a CE-HTML document to access DOM objects in a child SVG document, the following extensions SHALL be applied to [CEA-2014-A]:

- [5.4.a] XHTML Profile (CE-HTML); The following applies to item 3) d):
  - The `HTMLObjectElement` interface, including the `contentDocument` attribute of this interface, SHALL be supported for SVG documents. If the `contentDocument` property of `HTMLObjectElement` refers to a [SVG Tiny 1.2] document, then the available methods and properties for the `contentDocument` are limited to the common subset of the [SVG Tiny 1.2] uDOM and the `Element` interface defined in [DOM 2 Core].
  - Methods `blur()` and `focus()` SHALL be supported for SVG documents and SHALL have the same semantics as specified for interface `HTMLInputElement`.
- [Annex I, Table 9] The following extensions apply:

add HTMLObjectElement interface with the following properties and functions as defined by [DOM 2 HTML]:
align, border, contentDocument, data, height, hspace, name, tabindex, type, vspace, width, blur(), focus();

Scripting Interface (informative)	Properties and Methods (informative)	Additional Requirements and Recommendations (in addition to that defined above)

HTMLObjectElement	#HTMLElement
	align(*)
	border(*)
	contentDocument(**)
	data
	height
	hspace(*)
	name(*)
	tabindex
	type
	vspace(*)
	width
	blur()(**)
	focus()(**)
		(*) use of this attribute is deprecated
		
		(**) at least supported for SVG content

Table 6: HTMLObjectElement interface

6.4.2.2 Child SVG access to parent CE-HTML

In order to enable scripts in an SVG document to access DOM objects in its parent CE-HTML document, the following extensions SHALL be applied to [CEA-2014-A]:

- [5.4.2.a] The following extensions to be added to item 1) Properties - j) **readonly String name**:
  - If a window object is associated with an embedded document, then the name property of the window SHALL match the name property of the element that generated the embedded document.
- [5.4.2.a] The following extensions to be added to item 1) Properties x):
  - x)**readonly Element frameElement** - Property frameElement SHALL resolve to the embedding element object or null if there is no such element.
- [Annex I, Table 9] The following extensions apply:
  - under window object entry, add read-only property frameElement;

Scripting Interface (informative)	Properties and Methods (informative)	Additional Requirements and Recommendations (in addition to that defined above)
Window	frameElement(available to DocumentViews of embedded SVG documents) cea2014_protocol_version cea2014_protocol_subversionNr document frames history innerHeight innerWidth location id name onblur onfocus onkeypress	Additional implementation/authoring requirements: The methods and properties SHALL adhere to [Req.5.4.2.a]. (*) Method download() is only mandatory for Remote UI Clients for which <download> is true in their capability profile. (**) Method

	onkeydown onkeyup httptimeout(****) parent top maxHeight(****) maxWidth(****) topmost(****) height(****) width(****) focus() setTimeout() clearTimeout() setRenderMode() openURL()(****) reload()(****) replace()(****) requestFocus()(****) setHttpTimeout()(****) setTimer()(****) clearTimer()(****) getFrame()(****) escapeBeyondTopmost()(****) exitUnit()(****) download()(*) subscribeToNotifications()(**) XMLHttpRequest(***)	subscribeToNotifications is only mandatory for i-Box clients. (***) Property XMLHttpRequest is only mandatory for i-Box clients. (****) CEA-2027-A specific method that may not be supported as per Annex B of this DAE specification.

Table 7: Window interface

Add the DocumentView interface (defined in Table 8) to uDOM defined in [SVG Tiny 1.2]. It is a subset of [DOM 2 Views]. The DocumentView interface provides the access to innermost Window object so that child document can access to parent document. It has defaultView property described as follows:

interface **DocumentView**
{
readonly Window defaultView;
}
	defaultView resolves to the innermost Window object into which the Document is presented.
	
	If the window object is CE-HTML based, then the available methods and properties for the defaultView.frameElement are limited to the common subset of the [SVG Tiny 1.2] uDOM and DOM Core L2 Element interface.

Table 8: DocumentView interface to be added to uDOM

The SVGDocument interface also changes to inherit the DocumentView interface.

6.4.2.3 Parent SVG access to child CE-HTML

In order to enable scripts in an SVG document to access DOM objects in a child CE-HTML document, the following extensions SHALL be applied to [SVG Tiny 1.2] :

- Add SVGForeignElement interface to uDOM defined in [SVG Tiny 1.2]. This interface represents the 'foreignObject' element in the SVG document.

interface **SVGForeignObjectElement**
{
Document contentDocument;
}
	The document this object contains, if there is any and it is available, or null otherwise.
	
	If this document is CE-HTML based, then the available methods and properties for the document are limited to the common subset of the [SVG Tiny 1.2] uDOM and DOM Core L2 Element interface.

Table 9: SVGForeignObjectElement interface to be added to uDOM

6.4.2.4 Child CE-HTML access to parent SVG

In order to enable scripts in a CE-HTML document to access DOM objects in its parent document, the following extensions SHALL be applied to [CEA-2014-A]:

- [5.4.a] XHTML Profile (CE-HTML); The following to be added to item 3) DOM2 - f)
  - **f) DOM level 2 Views** , with at least providing support property `defaultView` which SHALL resolve to the innermost `Window` scripting object into which the `Document` is presented. If `Window` object is [SVG Tiny 1.2] based, then the available methods and properties for the `defaultView.frameElement` are limited to the common subset of the [SVG Tiny 1.2] uDOM and [DOM 2 Core] `Element` interface.
- [Annex I, Table 9] The following extensions apply:
  - under `Document` interface entry, add `read-only` property `defaultView`;

Scripting Interface (informative)	Properties and Methods (informative)	Additional Requirements and Recommendations (in addition to that defined above)

`Document`	`#Node`
	`defaultView`
	`doctype`
	`documentElement`
	`implementation`
	`createAttribute()`
	`createAttributeNS()`
	`createCDATASection()`
	`createComment()`
	`createDocumentFragment()`
	`createElement()`
	`createElementNS(),`
	`createEntityReference()`
	`createProcessingInstruction()`
	`createTextNode()`
	`getElementById()`
	`getElementsByTagName()`
	`getElementsByTagNameNS()`
	`importNode()`
		Additional implementation/authoring guideline:
		
		CE-HTML clients MAY not provide full support for XML namespaces and processing instructions, hence methods `getElementByTagNameNS()`, `createAttributeNS()`, `createElementNS()`,and `createProcessingInstruction()` MAY not be supported.

Table 10: Document interface

In order to support access from [SVG Tiny 1.2] document to the CE-HTML document, the following extensions SHALL be applied to [SVG Tiny 1.2]:

- Add `Window` interface to the uDOM defined in [SVG Tiny 1.2]. `Window` interface is subset to the `Window` object defined in the W3C WebAPI activity [Window Object]. The `Window` interface provides the access to other documents in a compound document by reference.

`interface Window`
`{ readonly String name;`
`readonly Element frameElement;`
`}`
	If a `Window` object is associated with an embedded document, then the `name` property of the window SHALL match the `name` property of the element that generated the embedded document.
	
	`frameElement` property contains reference to embedded element or `null` if there is no such element.

Table 11: Window interface to be added to uDOM

6.4.2.5 Event propagation

For documents embedded as children of another document (e.g. an SVG document embedded in a CE-HTML document), events targeted at elements in the child document typically are not dispatched to nodes in the parent document.. However, events will still be dispatched to other applications as defined in Section 7.2.6.

No event listener in parent catches any event in child document. If user pushes key button when an [SVG Tiny 1.2] element is focused, then `KeyEvent` occurs on the focused [SVG Tiny 1.2] element and it typically does not propagate to the CE-HTML document.

To accomplish setting and moving focus through [SVG Tiny 1.2] and CE-HTML document, following extension SHALL be applied.

- [Req. 5.4.1.m] The following extensions apply:
  - If an HTML document includes `<object>` elements whose type attribute value is `image/svg+xml`, then the Remote UI Client SHALL (1) offer a means to set focus to any SVG element type for which an event listener SHALL be registered, and (2) generate appropriate DOM 2 focus events accordingly.
- [Req. 5.4.1.n] The following extensions apply:
  - If an HTML document includes `<object>` elements whose type attribute value is `image/svg+xml`, then the Remote UI Client SHALL (1) offer a means to move focus away from any SVG element type for which an event listener SHALL be registered, and (2) generate appropriate DOM 2 focus events accordingly.

In order to pass an event that occurred in the CE-HTML document to a script in [SVG Tiny 1.2], the following extensions SHALL be applied to [SVG Tiny 1.2] :

- Add `DocumentEvent` interface to uDOM defined in [SVG Tiny 1.2]. It is same as `DocumentEvent` in DOM Level 2 Events. `SVGDocument` interface also changes to inherit the `DocumentEvent` interface.
- Add the `dispatchEvent` method to `EventTarget` defined in [SVG Tiny 1.2]

6.4.2.5.1 DocumentEvent

The `DocumentEvent` interface provides a mechanism by which the user can create an `Event` of a type supported by the implementation.

6.4.2.5.1.1 Methods

`Event` **`createEvent`**`( DOMString eventType )`		
Description	Create a specified event. If specified *eventType* is supported, newly created `Event` object is returned. Otherwise, `null` is returned.	
Arguments	*eventType*	The type of `Event` interface to be created.

6.4.2.5.2 EventTarget

6.4.2.5.2.1 Methods

`Boolean` **`dispatchEvent`**`( Event evt )`	
Description	This method allows the dispatch of events into the implementations event model. The return value of `dispatchEvent` indicates whether any of the listeners which handled the event called `preventDefault`. If `preventDefault` was called the value is false, else the value is true.

Arguments	*evt*	Specifies the event type, behavior, and contextual information to be used in processing the event.

NOTE: The following methods are described in the uDOM defined in [SVG Tiny 1.2]:

```
void addEventListener( String type, EventListener listener, Boolean useCapture )
```

```
void removeEventListener( String type, EventListener listener, Boolean useCapture)
```

```
void addEventListenerNS( String namespaceURI, String type, EventListener listener,
Boolean useCapture, DOMObject evtGroup )
```

```
void removeEventListenerNS( String namespaceURI, String type, EventListener
listener, Boolean useCapture, DOMObject evtGroup )
```

6.4.3 Attention to DAE application developers

6.4.3.1 Script APIs defined in DAE

The use of any script APIs defined in the DAE specification in script code inside an SVG document is not defined. The script code in [SVG Tiny 1.2] document SHALL be able to call functions on DOM nodes in [CEA-2014-A] document and vice versa. The present document does not define how to include CE-HTML embedded objects directly in [SVG Tiny 1.2] documents.

6.4.3.2 Codec and connection supporting in SVG

DAE applications SHALL NOT rely upon codec support for the use of audio and video elements from [SVG Tiny 1.2].

DAE applications SHALL NOT rely upon support for use of `Connection` from [SVG Tiny 1.2].

7 APIs

7.1 Object factory API

This section defines the methods to check and create an instance of the DAE defined embedded objects within Javascript.

The OITF SHALL support a globally accessible object of type "`OipfObjectFactory`" as a static property "`oipfObjectFactory`" of the `Window` interface with the API as defined in this section. The object factory SHALL ensure that the referenced objects are correctly set up. This is an alternative to instantiating embedded objects (or plug-ins) outside of Javascript.

The factory object can be accessed as a property of the window object (i.e. `window.oipfObjectFactory` or `oipfObjectFactory`).

7.1.1 Methods

<table>
<tr><td colspan="3"><code>Boolean OipfObjectFactory.isObjectSupported(String mimeType)</code></td></tr>
<tr><td>Description</td><td colspan="2">This method SHALL return <code>true</code> if and only if an object of the specified type is supported by the OITF. The method SHALL return <code>false</code> if the MIME type passed as a parameter is not supported by the client.</td></tr>
<tr><td>Arguments</td><td>mimeType</td><td>The mimeType may have any of the MIME types defined in tables 1 to 4 of [OIPF_META2] (for example, "<code>video/mpeg</code>" or "<code>audio/x-wav</code>") or any of the DAE defined mime types listed below.
<table>
<tr><th>DAE MIME Type</th></tr>
<tr><td>application/notifsocket</td></tr>
<tr><td>application/oipfApplicationManager</td></tr>
<tr><td>application/oipfCapabilities</td></tr>
<tr><td>application/oipfCodManager</td></tr>
<tr><td>application/oipfConfiguration</td></tr>
<tr><td>application/oipfDownloadManager</td></tr>
<tr><td>application/oipfDownloadTrigger</td></tr>
<tr><td>application/oipfDrmAgent</td></tr>
<tr><td>application/oipfGatewayInfo</td></tr>
<tr><td>application/oipfIMS</td></tr>
<tr><td>application/oipfMDTF</td></tr>
<tr><td>application/oipfParentalControlManager</td></tr>
<tr><td>application/oipfRecordingScheduler</td></tr>
<tr><td>application/oipfRemoteControlFunction</td></tr>
</table>
</td></tr>
</table>

			application/oipfRemoteManagement	
			application/oipfSearchManager	
			application/oipfStatusView	
			video/broadcast	

7.1.1.1 Visual objects

The methods in this section all return `HTMLObjectElement` objects which can be inserted in to the DOM tree. All objects in Section 7 which have a visual representation on the screen can be created using methods in this section. Only for objects defined in Section 7, that are supported by the device (i.e. as indicated through the client capability description), a corresponding method name to instantiate the object through the `OipfObjectFactory` class can be assumed to be present on the `oipfObjectFactory` object. For any other object, a corresponding method name cannot be assumed to be present.

`HTMLObjectElement oipfObjectFactory.`**`createVideoBroadcastObject`**`()`
`HTMLObjectElement oipfObjectFactory.`**`createVideoMpegObject`**`()`
`HTMLObjectElement oipfObjectFactory.`**`createStatusViewObject`**`()`	

Description	If the object type is supported, each of these methods shall return an instance of the corresponding embedded object. Since objects do not claim scarce resources when they are instantiated, instantiation shall never fail if the object type is supported. If the method name to create the object is not supported, the OITF SHALL throw an error with the error.name set to the value "`TypeError`". If the object type is supported, the method shall return an `HTMLObjectElement` equivalent to the specified object. The value of the type attribute of the HTMLObjectElement SHALL match the mimetype of the instantiated object, for example "`video/broadcast`" in case of method `oipfObjectFactory.createVideoBroadcastObject()`.

7.1.1.2 Non-Visual objects

The methods in this section all return javascript objects which implement the interfaces of their corresponding objects. They can not be inserted in the DOM tree. All objects in Section 7 which do *not* have a visual representation on the screen can be created using methods in this section. Only for objects defined in Section 7, that are supported by the device (i.e. as indicated through the client capability description), a corresponding method name to instantiate the object through the `OipfObjectFactory` class can be assumed to be present on the `oipfObjectFactory` object. For any other object, a corresponding method name cannot be assumed to be present.

```
Object oipfObjectFactory.createApplicationManagerObject()

Object oipfObjectFactory.createCapabilitiesObject()

ChannelConfig oipfObjectFactory.createChannelConfig()

Object oipfObjectFactory.createCodManagerObject()
```

`Object oipfObjectFactory.createConfigurationObject()`
`Object oipfObjectFactory.createDownloadManagerObject()`
`Object oipfObjectFactory.createDownloadTriggerObject()`
`Object oipfObjectFactory.createDrmAgentObject()`
`Object oipfObjectFactory.createGatewayInfoObject()`
`Object oipfObjectFactory.createIMSObject()`
`Object oipfObjectFactory.createMDTFObject()`
`Object oipfObjectFactory.createNotifSocketObject()`
`Object oipfObjectFactory.createParentalControlManagerObject()`
`Object oipfObjectFactory.createRecordingSchedulerObject()`
`Object oipfObjectFactory.createRemoteControlFunctionObject()`
`Object oipfObjectFactory.createRemoteManagementObject()`
`Object oipfObjectFactory.createSearchManagerObject()`	

Description	If the object type is supported, each of these methods SHALL return an instance of the corresponding embedded object. This may be a new instance or existing instance. For example, the object will likely be a global singleton object and calls to this method may return the same instance. Since objects do not claim scarce resources when they are instantiated, instantiation SHALL never fail if the object type is supported. If the method name to create the object is not supported, the OITF SHALL throw an error with `name` property set to the value "`TypeError`". If the object is supported, the method SHALL return a javascript `object` which implements the interface for the specified object.

7.1.2 Examples

This section provides examples of the usage of the methods.

The first example shows how to query whether an instance of the A/V Control object for a specified MIME type can be created without the application having to attempt to instantiate the object.

```
var videoPlayer;
if (window.oipfObjectFactory.isObjectSupported("video/mpeg")) {
    videoPlayer = window.oipfObjectFactory.createVideoMpegObject();
    // append object to document
    document.getElementById('playerDiv').appendChild(videoPlayer);
    videoPlayer.data = "rtsp://server/barker_channel";
}
```

If the OITF does not support the created object the OITF SHALL throw an error with the `error.name` set to the value "`TypeError`". The example below shows how this can be used by applications:

```
try {
    configuration = window.oipfObjectFactory.createConfigurationObject();
}
catch (error) {
```

```
    alert("application/oipfConfiguration object could not be created - error name: "
+ error.name + " - error message: " + error.message);
}
```

7.2 Application Management APIs

An OITF providing DAE application capability SHALL implement the behaviour of the classes defined in this section.

7.2.1 The application/oipfApplicationManager embedded object

An OITF SHALL support a non-visual embedded object of type "application/oipfApplicationManager", with the following Javascript API, to enable applications to access the privileged functionality related to application lifecycle and management that is provided by the application model defined in this section.

If one of the methods on the application/oipfApplicationManager is called by a webpage that is not a privileged DAE application, the OITF SHALL throw an error as defined in Section 10.1.1.

7.2.1.1 Constants

The following constants are defined as properties of the application/oipfApplicationManager embedded object:

Name	Value	Use
WIDGET_INSTALLATION_STARTED	0	The Widget installation has started
WIDGET_INSTALLATION_COMPLETED	1	The Widget installation has completed successfully
WIDGET_INSTALLATION_FAILED	2	The Widget installation has failed
WIDGET_UNINSTALLATION_STARTED	3	The Widget uninstallation has started
WIDGET_UNINSTALLATION_COMPLETED	4	The Widget uninstallation has completed successfully
WIDGET_UNINSTALLATION_FAILED	5	The Widget uninstallation has failed
WIDGET_ERROR_STORAGE_AREA_FULL	10	The local storage device is full
WIDGET_ERROR_DOWNLOAD	11	The Widget cannot be downloaded
WIDGET_ERROR_INVALID_ZIP_ARCHIVE	12	The Widget package is corrupted or is an Invalid Zip Archive (as defined in [Widgets-Packaging])
WIDGET_ERROR_INVALID_SIGNATURE	13	Widget's Signature Validation failed
WIDGET_ERROR_GENERIC	14	Other reason

7.2.1.2 Properties

function **onLowMemory**()
The function that is called when the OITF is running low on available memory for running DAE

applications. The exact criteria determining when to generate such an event is implementation specific.

`function` **`onApplicationLoaded`**`( Application appl )`

The function that is called immediately prior to a `load` event being generated in the affected application. The specified function is called with one argument `appl`, which provides a reference to the affected application.

`function` **`onApplicationUnloaded`**`( Application appl )`

The function that is called immediately prior to an `unload` event being generated in the affected application. The specified function is called with one argument `appl`, which provides a reference to the affected application.

`function` **`onWidgetInstallation`**`( WidgetDescriptor wd, Integer state, Integer reason )`

The callback function that is called during the installation process of a Widget. The function is called with three arguments:

- `WidgetDescriptor wd`: the `WidgetDescriptor` for the installed Widget. Some attributes of this argument may not have been initialised and may be null when the function is called until the Widget is successfully installed.
- `Integer state`: the state of the installation; valid values are:
  - `WIDGET_INSTALLATION_STARTED,`
  - `WIDGET_INSTALLATION_COMPLETED`
  - `WIDGET_INSTALLATION_FAILED`

as defined in Section 7.2.1.4.

- `Integer reason`: indicates the reason for installation failure. This is only valid if the value of the `state` argument is `WIDGET_INSTALLATION_FAILED` otherwise this argument SHALL be `null`. Valid values for this field are:
  - `WIDGET_ERROR_STORAGE_AREA_FULL`
  - `WIDGET_ERROR_DOWNLOAD`
  - `WIDGET_ERROR_INVALID_ZIP_ARCHIVE`
  - `WIDGET_ERROR_INVALID_SIGNATURE`
  - `WIDGET_ERROR_GENERIC`

as defined in Section 7.2.1.4.

`function` **`onWidgetUninstallation`**`( WidgetDescriptor wd, Integer state )`

The function that is called during the uninstallation process of a Widget. The function is called with two

arguments, defined below:

- `WidgetDescriptor wd`: the WidgetDescriptor of the Widget to be uninstalled.
- `Integer state`: the state of the installation; valid values are:
  - `WIDGET_UNINSTALLATION_STARTED,`
  - `WIDGET_UNINSTALLATION_COMPLETED`
  - `WIDGET_UNINSTALLATION_FAILED`

as defined in Section 7.2.1.4.

`readonly WidgetDescriptorCollection` **`widgets`**
A collection of `WidgetDescriptor` objects for the Widgets currently installed on the OITF.

7.2.1.3 Methods

`Integer` **`getApplicationVisualizationMode`**`()`	
Description	Returns the current mode used by the OITF to visualize applications, whereby a return value:

1	corresponds to the application visualization mode as defined by bullet 1) of Section 4.4.6, i.e. multiple applications visible simultaneously with DAE applications managing their own size, position and visibility
2	corresponds to the application visualization mode as defined by bullet 2) of Section 4.4.6, i.e. multiple applications visible simultaneously with OITF managing the size, position, visibility of applications
3	corresponds to the application visualization mode as defined by bullet 3) of Section 4.4.6, i.e. only a single application visible at any time.

`Application` **`getOwnerApplication`**`( Document document )`		
Description	Get the application that the specified document is part of. If the document is not part of an application, or the calling application does not have permission to access that application, this method will return `null`.	
Arguments	*document*	The document for which the `Application` object should be obtained.

`ApplicationCollection` **`getChildApplications`**`( Application application )`		
Description	Get the applications that are children of the specified application.	
Arguments	*application*	The application whose children should be returned.

void **gc**()	
Description	Provide a hint to the execution environment that a garbage collection cycle should be initiated. The OITF is not required to act upon this hint.

void **installWidget**(String uri)		
Description	Attempts to install on the OITF a Widget located at the URI passed. If the Widget is stored on a remote server it SHALL first be downloaded. This specification does not specify where the OITF stores the Widget package, nor does it define what happens to the original package after the installation process has finished (regardless of whether it succeeded or failed). When trying to install a Widget with an "id" that collides with the id of an already installed Widget (where the "id" is defined in Section 7.6.1 of [Widgets-Packaging] along with the extension defined in Section 11.1 of this specification), the OITF SHOULD ask the user for confirmation before installing the Widget. The OITF SHOULD provide information about the conflict (e.g. the version numbers, if available) to allow the user to decide whether to proceed with the installation or to cancel it. If the user confirms the installation, then the new Widget SHALL replace the one already installed; any storage area associated with the replaced Widget SHALL be retained. Note that the user can also choose to downgrade a Widget, i.e. install an old version of the Widget to replace the installed, more recent, one.	
Arguments	*uri*	The resource locator in form of a URI, which points to a Widget package to be installed.

void **uninstallWidget**(WidgetDescriptor wd)		
Description	Uninstalls a Widget. If this Widget is running it will be stopped. Any storage areas associated with the uninstalled Widget SHALL be deleted.	
Arguments	*wd*	A `WidgetDescriptor` object for a Widget installed on the OITF.

7.2.1.4 Events

For the intrinsic events listed in the table below a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onLowMemory`	`LowMemory`	Bubbles: No Cancelable: No Context Info: None
`onApplicationLoaded`	`ApplicationLoaded`	Bubbles: No Cancelable: No

		Context Info: appl
onApplicationUnloaded	ApplicationUnloaded	Bubbles: No Cancelable: No Context Info: appl
onWidgetInstallation	WidgetInstallation	Bubbles: No Cancelable: No Context: wd, state, reason
onWidgetUninstallation	WidgetUninstallation	Bubbles: No Cancelable: No Context: wd, state

NOTE: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving the events listed above during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `application/oipfApplicationManager` object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.2.2 The Application class

The `Application` class is used to implement the characteristics of a DAE application.

7.2.2.1 Properties

readonly Boolean **visible**
`true` if the application is visible, `false` otherwise. The value of this property is not affected by the application's Z-index or position relative to other applications. Only calls to the `show()` and `hide()` methods will affect its value.

readonly Boolean **active**
`true` if the application is in the list of currently active applications, `false` otherwise (as defined in Section 4.3.8).

readonly StringCollection **permissions**
`StringCollection` object containing the names of the permissions granted to this application.

readonly Boolean **isPrimaryReceiver**

`true` if the application receives cross application events before any other application, `false` otherwise.

`readonly Window` **`window`**
A strict subset of the DOM `Window` object representing the application. No symbols from the window object are accessible through this property except the following: • `void` **`postMessage`**`(` *`any message`*`, String targetOrigin )`

`readonly ApplicationPrivateData` **`privateData`**
Access the current application's private data object. If the application accessing the `privateData` property is not the current application, the OITF SHALL throw an error as defined in section 10.1.1.

`function` **`onApplicationActivated`**`()`
`function` **`onApplicationDeactivated`**`()`
`function` **`onApplicationShown`**`()`
`function` **`onApplicationHidden`**`()`
`function` **`onApplicationPrimaryReceiver`**`()`
`function` **`onApplicationNotPrimaryReceiver`**`()`
`function` **`onApplicationTopmost`**`()`
`function` **`onApplicationNotTopmost`**`()`
`function` **`onApplicationDestroyRequest`**`()`
`function` **`onApplicationHibernateRequest`**`()function` **`onKeyPress`**
`function` **`onKeyUp`**
`function` **`onKeyDown`**

Each of these event handlers represents a DOM 0 event handler that corresponds to one of the events listed in Sections 7.2.1.4 and 7.2.6.

7.2.2.2 Methods

`void` **`show`**`()`	
Description	If the application visualization mode as defined by method `getApplicationVisualizationMode()` in Section 7.2.1.3 is: 1 : Make the application visible.

	2 : Make the application visible. Calling this method from the application itself may have no effect. 3 : Request to make the application visible. This method only affects the visibility of an application. In the case where more than one application is visible, calls to this method will not affect the z-index of the application with respect to any other visible applications.

void **hide**()	
Description	If the application visualization mode as defined by method `getApplicationVisualizationMode()` in Section 7.2.1.3 is: 1 : Make the application invisible. 2 : Make the application invisible. Calling this method from the application itself may have no effect. 3 : Request to make the application invisible. Calling this method has no effect on the lifecycle of the application.

void **activateInput**(Boolean gainFocus)	
Description	Move the application to the front of the active applications list. If the application has been hidden using `Application.hide()`, this method does not cause the application to be shown. If the application visualization mode as defined by method `getApplicationVisualizationMode()` in Section 7.2.1.3 is: 1 : The application's `Window` object SHALL be moved to the top of the stack of visible applications. In addition, the application's `Window` object SHALL gain input focus if argument `gainFocus` has value `true.` 2 : The application's `Window` object SHALL be moved to the top of the stack of visible applications. In addition, the application's `Window` object SHALL gain input focus if argument `gainFocus` has value `true.`Calling this method from the application itself MAY have no effect. 3 : Request to make the application's `Window` object visible. Once visible, the application SHALL be given input focus, irrespective of the value for argument `gainFocus`.

void **deactivateInput**()	
Description	Remove the application from the active applications list. This has no effect on the lifecycle of the application and MAY have no effect on the resources it uses. Applications which are not active will receive no cross-application events, unless their `Application` object is the target of the event (as for the events defined in Section 7.2.6). Applications may still be manipulated via their `Application` object or their DOM tree.

Application **createApplication**(String uri, Boolean createChild)		
Description	Create a new application and add it to the application tree. Calling this method does not automatically show the newly-created application. This call is asynchronous and may return before the new application is fully loaded. An ApplicationLoaded event will be targeted at the Application object when the new application has fully loaded. If the application cannot be created, this method SHALL return null.	
Arguments	*uri*	The URI of the first page of the application to be created or the localURI of a Widget as defined in Section 7.2.8.1.1.
	createChild	Flag indicating whether the new application is a child of the current application. A value of true indicates that the new application should be a child of the current application; a value of false indicates that it should be a sibling.

void **destroyApplication**()	
Description	Terminate the application, detach it from the application tree, and make any resources used available to other applications. When an application is terminated, any child applications shall also be terminated.

Application **startWidget**(WidgetDescriptor wd, Boolean createChild)		
Description	Starts a Widget installed on the OITF. The behaviour of this method is equivalent to that of Application.createApplication(). The Widget is identified by its WidgetDescriptor. To get a list of the WidgetDescriptor objects for the installed Widgets one can check ApplicationManager.widgets property. If the Widget is already running or fails to start this call will return null.	
Arguments	*wd*	a WidgetDescriptor object for a Widget installed on the OITF.
	createChild	Flag indicating whether the new application is a child of the current application. A value of true indicates that the new application should be a child of the current application; a value of false indicates that it should be a sibling.

void **stopWidget**(WidgetDescriptor wd)		
Description	Terminate a running Widget. The behaviour of this method is equivalent to that of Application.destroyApplication(). Calling this method will detach the Widget from the application tree, and make any resources used available to other applications. When a Widget is terminated, any child applications shall also be terminated.	
Arguments	*wd*	A WidgetDescriptor object for a Widget installed on the OITF.

7.2.3 The ApplicationCollection class

```
typedef Collection<Application> ApplicationCollection
```

The `ApplicationCollection` class represents a collection of `Application` objects. See annex K for the definition of the collection template.

7.2.4 The ApplicationPrivateData class

7.2.4.1 Properties

`readonly Keyset` **keyset**
The object representing the user input events sent to the DAE application.

`readonly Boolean` **wakeupApplication**
The `wakeupApplication` property is set if there has been a `prepareWakeupApplication()` request by that application.

`readonly Boolean` **wakeupOITF**
The `wakeupOITF` property is set if there has been a `prepareWakeupOITF()`.

7.2.4.2 Methods

`Integer` **getFreeMem**`()`	
Description	Let application developer query information about the current memory available to the application. This is used to help during application development to find application memory leaks and possibly allow an application to make decisions related to its caching strategy (e.g. for images). Returns the available memory to the application or -1 if the information is not available. For example: `debug("[APP] free mem = " + appman.getOwnerApplication(window.document).privateData.getFreeMem() + "\n");`

`Boolean` **prepareWakeupApplication**`( String URI, String token, Date time )`	
Description	The `prepareWakeupApplication()` method allows the DAE application to set-up the OITF to wake-up at a specified time. The wake-up is limited to the OITF being in the PASSIVE_STANDBY at the specified time. If the timer expires while the DAE application is in a different state it is silently ignored. Only one wake-up is to be supported for a DAE application. If a previous wake-up request had been registered it SHALL be overwritten.

	If the wake-up fails to be set-up this operation SHALL return false. Failure may be due to OITF expecting to change to an OFF power state which would not allow the wake-up request to survive.	
Arguments	*URI*	The URI from which the content can be fetched.
	token	The token is a string which the application may retrieve with `clearWakeupToken()`.
	time	The time when the wake-up is to occur.

`Boolean` **`prepareWakeupOITF`**`(Date time )`		
Description	The `prepareWakeupOITF()` method allows the DAE application to set-up the OITF to wake-up at a specified time. The wake-up is limited to the OITF being in the `PASSIVE_STANDBY` or `PASSIVE_STANDBY_HIBERNATE` state at the specified time. If the timer expires while the DAE application is in a different state it is silently ignored. Unlike `prepareWakeupApplication()` this method applies to all the DAE applications and not limited to a single DAE application If the wake-up fails to be set-up this operation SHALL return false. Failure may be due to OITF expecting to change to an `OFF` power state which would not allow the wake-up request to survive.	
Arguments	*time*	The time when the wake-up is to occur.

`String` **`clearWakeupToken`**`( )`	
Description	The `clearWakeupToken()` method shall return the token set in prepareWakeupApplication( ) method. The wake-up token should be cleared once it is read in order to limit usage to only when the DAE application starts up.

7.2.5 The Keyset class

The `Keyset` object permits applications to define which key events they request to receive. There are two means of defining this. Common key events are represented by constants defined in this class which are combined in a bit-wise mask to identify a set of key events. Less common key events are not included in one of the defined constants and form part of an array.

The supported key events indicated through the capability mechanism in Section 9.3 SHALL be the same as the maximum set of key events available to the browser as indicated through this object

The default set of key events available to applications which do not call `Keyset.setValue` SHALL be all those indicated by the constants in this class which are supported by the OITF excluding those indicated by OTHER.

7.2.5.1 Constants

The following constants are defined as properties of the `KeySet` class:

Constant name	Numeric Value	Use
RED	0x1	Used to identify the VK_RED key event.
GREEN	0x2	Used to identify the VK_GREEN key event.
YELLOW	0x4	Used to identify the VK_YELLOW key event.
BLUE	0x8	Used to identify the VK_BLUE key event.
NAVIGATION	0x10	Used to identify the VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_ENTER and VK_BACK key events.
VCR	0x20	Used to identify the VK_PLAY, VK_PAUSE, VK_STOP, VK_NEXT, VK_PREV, VK_FAST_FWD, VK_REWIND, VK_PLAY_PAUSE key events.
SCROLL	0x40	Used to identify the VK_PAGE_UP and VK_PAGE_DOWN key events.
INFO	0x80	Used to identify the VK_INFO key event.
NUMERIC	0x100	Used to identify the number events, 0 to 9.
ALPHA	0x200	Used to identify all alphabetic events.
OTHER	0x400	Used to indicate key events not included in one of the other constants in this class.

7.2.5.2 Properties

readonly Integer **value**
The value of the keyset which this DAE application will receive.

readonly Integer **otherKeys[]**
If the OTHER bit in the value property is set then this indicates those key events which are available to the browser which are not included in one of the constants defined in this class, If the OTHER bit in the value property is not set then this property is meaningless.

readonly Integer **maximumValue**
In combination with maximumOtherKeys, this indicates the maximum set of key events which are available to the browser. When a bit in this maximumValue has value 0, the corresponding key events are never available to the browser.

readonly Integer **maximumOtherKeys[]**
If the OTHER bit in the maximumValue property is set then, in combination with maximumValue, this

indicates the maximum set of key events which are available to the browser. For key events which are not included in one of the constants defined in this class, if the VK_* constant representing the key is not listed in this array then it is never available to the browser. If the OTHER bit in the value property is not set then this property SHALL take the value `undefined`.

7.2.5.3 Methods

`Integer` **`setValue`**`( Integer value, Integer otherKeys[] )`		
Description	Sets the value of the keyset which this DAE application requests to receive. Where more than one DAE application is running, the events delivered to the browser SHALL be the union of the events requested by all running DAE applications. Under these circumstances, applications may receive events which they have not requested to receive. The return value indicates which keys will be delivered to this DAE application encoded as bit-wise mask of the constants defined in this class. These key events are defined using the VK_* constants defined in Annex F of [CEA-2014-A].	

Arguments	value	The value is a number which is a bit-wise mask of the constants defined in this class. For example;
		`myKeyset = myApplication.privateData.keyset;`
		`myKeyset.setValue(0x00000013);`
		`myKeyset.setValue(myKeyset.INFO | myKeyset.NUMERIC);`

	otherkeys	This parameter is optional. If the value parameter has the OTHER bit set then it is used to indicate the key events that the application wishes to receive which are not represented by constants defined in this class.

7.2.6 New DOM events for application support

New events have been created that are raised on the `Application` objects in the application tree. These are normal events, not cross-application events, and are used to indicate changes in the state of an application.

Event	Description
`ApplicationActivated`	Issued when an application focus change occurs to inform the recipient of the event that the application is now focussed.
`ApplicationDeactivated`	Issued when an application focus change occurs to inform the recipient of the event that the application is now no longer focussed.
`ApplicationShown`	Issued when an application has become visible.
`ApplicationHidden`	Issued when an application has become hidden.
`ApplicationPrimaryReceiver`	This event is issued to indicate that the target is now at the front of the active application list.

Event	Description
ApplicationNotPrimaryReceiver	This event is issued to indicate that the target is no longer at the front of the active application list.
ApplicationTopmost	This event is issued to indicate that the target is now the topmost (i.e. it has the highest Z-index and is not obscured by any other visible applications, for OITFs where multiple applications are visible simultaneously.
ApplicationNotTopmost	This event is issued to indicate that the target is no longer at the topmost application. For OITFs where only one application is visible at a time, this event indicates that the application is no longer visible to the user.
ApplicationDestroyRequest	This event is issued to indicate that the target application is about to be terminated. It is not issued when an application calls destroyApplication() method for itself (i.e. to exit itself). Non-responsive applications SHOULD be forcibly terminated by the OITF, including the case where listeners for ApplicationDestroyRequest events do not return promptly. The determination of when an application is "non-responsive" is terminal-specific. If an application does not register a listener for this event and there is a need for the system to terminate the application, then the application SHALL be terminated immediately.
ApplicationHibernateRequest	This event is issued to indicate that the OITF is about to enter a hibernate mode. The OITF SHALL start a short watchdog timer (e.g. 2 seconds). During this period the application may take any actions (for example to store the currently viewed channel in case of an unsuccessful start-up).

Table 12: New DOM events for application support

These events do not bubble and cannot be cancelled. Each of these events has a corresponding DOM 0 event handler property on the Application object.

7.2.7 Examples (informative)

The examples below illustrate some aspects of the application model.

7.2.7.1 Locating the Application object

The ApplicationManager class provides the getOwnerApplication() method, which returns the document's owning application node:

```
var appMgr = oipfObjectFactory.createApplicationmanagerObject();
var self = appMgr.getOwnerApplication(Window.document);
```

All other application functionality is available from this object.

7.2.7.2 Creating a new application

Creating a new application is a simple matter of creating a new Application object.

```
// This example assumes that the application/oipfApplicationManager object
// is already instantiated in the DOM tree with the ID
// “applicationmanager”
var appMgr = document.getElementById(“applicationmanager”);
var self = appMgr.getOwnerApplication(Window.document);
var child = self.createApplication( url_of_application, true );
```

A typical requirement on an application is to only become visible once it has fully loaded. To do this, it can take advantage of `load` events. Here is an example from a clock application, which wants to load an image to become the background of the clock, upon which it can write the text of the clock. This example makes use of the additional window methods `resizeTo`(), `moveTo`() and property '`screen`', which are only available in application visualization mode 1, as defined in Section 4.4.6.

```
<script>
function loaded() {

    var screen = document.defaultView.screen;
    var clock = document.getElementById('clock');
    window.resizeTo( clock.width, clock.height );

    // position in bottom left
    window.moveTo( clock.width, screen.availHeight - clock.height );

    setup_clock( clock.width, clock.height );

    // Assumes that the application/oipfApplicationManager object has the ID
    // “applicationmanager”
    Var appMgr = document.getElementById(“applicationmanager”);
    var self = appMgr.getOwnerApplication(Window.document);
    self.show();
}
</script>

<style> * { margin: 0cm } </style>

<body onload="loaded()">
   <img id="clock" src="clockbackground.png" style="position: absolute; top: 0px;
left=0px">
</body>
```

7.2.8 Widget APIs

This section defines APIs an author can use to interact with Widgets installed on the OITF. Note that the Widget lifecycle is managed through the application manager as defined in the previous sections.

7.2.8.1 The WidgetDescriptor class

The `WidgetDescriptor` class is used to implement the characteristics of a DAE Widget. It extends the Widget interface defined in Section 11.3 of this specification with the properties below.

7.2.8.1.1 Properties

`readonly String` **`localURI`**
The URI on the local storage where the Widget has been installed. It can be used as an argument to `ApplicationManager.createApplication()` to run an installed Widget.

`readonly StringCollection` **`defaultIcon`**
A collection of URI strings for all the available default icons. Default icons are defined in [Widgets-Packaging]. This collection only contains URIs for the icons currently available in the Widget package.

`readonly StringCollection` **`customIcons`**
A collection of URI strings for all the custom icons of the current Widget. Custom icons are defined in

[Widgets-Packaging].

readonly boolean **running**
This flag indicates the running state of the Widget.

7.2.8.1.2 Clarifications

The `WidgetDescriptor` class is used to identify an installed Widget regardless of whether it is running or not, and so some clarification on the attribute values defined for the Widget interfaces [Widgets-APIs] is needed. The attributes `height` and `width` are defined in [Widgets-APIs] on the "Widget instance's viewport". When the Widget is not running those attributes SHALL take the value defined in the Widget Manifest (if any) otherwise they SHALL be null. When the Widget is running these attributes SHALL adhere to what's defined in [Widgets-APIs].

7.2.8.2 The WidgetDescriptorCollection class

```
typedef Collection<WidgetDescriptor> WidgetDescriptorCollection
```

The `WidgetDescriptorCollection` class represents a collection of `WidgetDescriptor` objects.

7.3 Configuration and setting APIs

This section defines the interface to configuration and user settings information. Hardware configuration of the OITF is managed via an instance of the `LocalSystem` object. This provides access to hardware information and provides an entry point to configure the outputs and network interfaces of the OIF. Settings relating to the user interface and behaviour of the platform software are managed via an instance of the `Configuration` object.

This section is subject to security control, (see 10.1.4.7) and only applies if `<configurationChanges>` has value `true`

7.3.1 The application/oipfConfiguration embedded object

The OITF SHALL implement the "`application/oipfConfiguration`" object as defined below. This object provides an interface to the configuration and user settings facilities within the OITF.

7.3.1.1 Properties

readonly Configuration **configuration**
Accesses the configuration object that sets defaults and shows system settings.

readonly LocalSystem **localSystem**
Accesses the object representing the platform hardware.

7.3.2 The Configuration class

The `Configuration` object allows configuration items within the system to be read and modified. This includes settings such as audio and subtitle languages, display aspect ratios and other similar settings. Unlike the `LocalSystem` object, this is concerned with software- and application-related settings rather than hardware configuration and control.

NOTE: The following properties and methods present in earlier revisions of this specification have been moved to the application/oipfParentalControlManager embedded object described in Section 7.9.1: `isPINEntryLocked`, `setParentalControlPIN()`, `unlockWithParentalControlPIN()`, `verifyParentalControlPIN()` and `setBlockUnrated()`.

7.3.2.1 Properties

`String` **`preferredAudioLanguage`**
A comma-separated set of languages to be used for audio playback, in order of preference. Each language SHALL be indicated by its ISO 639 language code.

`String` **`preferredSubtitleLanguage`**
A comma-separated set of languages to be used for subtitle playback, in order of preference. Each language SHALL be indicated by its ISO 639 language code.

`String` **`countryId`**
An ISO-3166 three character country code identifying the country in which the receiver is deployed.

`Integer` **`regionId`**
An integer indicating the time zone within a country in which the receiver is deployed. A value of 0 SHALL represent the eastern-most time zone in the country, a value of 1 SHALL represent the next time zone to the west, and so on. Valid values are in the range 0 – 60.

`Integer` **`pvrPolicy`**

The policy dictates what mechanism the system should use when storage space is exceeded.

Valid values are shown in the table below.

Value	Description
0	Indicates a recording management policy where no recordings are to be deleted.
1	Indicates a recording management policy where only watched recordings MAY be deleted.
2	Indicates a recording management policy where only recordings older than the specified threshold (given by the `pvrSaveDays` and `pvrSaveEpisodes` properties) MAY be deleted.

Integer **pvrSaveEpisodes**
When the pvrPolicy property is set to the value 2, this property indicates the minimum number of episodes that SHALL be saved for series-link recordings.

Integer **pvrSaveDays**
When the pvrPolicy property is set to the value 2, this property indicates the minimum save time (in days) for individual recordings. Only recordings older than the save time MAY be deleted.

Integer **pvrStartPadding**
The default padding (measured in seconds) to be added at the start of a recording.

Integer **pvrEndPadding**
The default padding (measured in seconds) to be added at the end of a recording.

Integer **preferredTimeShiftMode**
The time shift mode indicates the preferred mode of operation for support of timeshift playback in the video/broadcast object. Valid values are defined in the timeShiftMode property in Section7.13.2.2. The default value is 0, timeshift is turned off.

7.3.2.2 Methods

String **getText**(String key)		
Description	Get the system text string that has been set for the specified key.	
Arguments	*key*	A key identifying the system text string to be retrieved.

void **setText**(String key, String value)		
Description	Set the system text string that has been set for the specified key. System text strings are used for automatically-generated messages in certain cases, e.g. parental control messages.	
Arguments	*key*	The key for the text string to be set. Valid keys are:

Key	Description
no_title	Text string used as the title for programmes and channels where no guide information is available. Defaults to “No information”
no_synopsis	Text string used as the synopsis for programmes where no guide

			information is available. Defaults to “No further information available”
		blocked_title	Text string used as the title for programmes and channels blocked by parental control settings (if metadata hiding is enabled). Defaults to “BLOCKED”
		blocked_synopsis	Text string used as the synopsis for programmes blocked by parental control settings (if metadata hiding is enabled). Defaults to “Program blocked by user”
		manual_recording	Text string used to identify a manual recording. Defaults to “Manual Recording”
	value	The new value for the system text string.	

7.3.3 The LocalSystem class

The `LocalSystem` object allows hardware settings related to the local device to be read and modified.

7.3.3.1 Constants

The following constants are defined as properties of the `LocalSystem` class:

Name	Value	Use
OFF	0	The OITF is in the off state and no power is consumed. This is the case of a power outage or if the OITF has the ability to be completely turned off. Scheduled recording is not expected to work.
ON	1	The OITF is in normal working mode with user interaction. The DAE applications may render any presentation graphically.
PASSIVE_STANDBY	2	The OITF is in the lowest possible power consumption state (meeting regulations and certifications). The OITF may support wake-up from a passive standby in order, for example, to perform a scheduled recording.
ACTIVE_STANDBY	3	The OITF is in an intermediate power consumption state. The output to the display shall be inactive. In this state DAE applications may continue to operate.

PASSIVE_STANDBY_HIBERNATE	4	The OITF is in the lowest possible power consumption state (meeting regulations and certifications). If the platform supports hibernate mode then the OITF stores all applications in volatile memory to allow for quick startup.

7.3.3.2 Properties

readonly String **deviceID**
Private OITF Identifier. Unique identifier which SHALL be the same as `X-HNI-IGI-OITF-DeviceID` in [OIPF_PROT2]. This property SHALL take the value `undefined` except when accessed by applications meeting either of the following criteria: • The application is signalled in an SD&S service provider discovery record with an application usage of `urn:oipf:cs:ApplicationUsageCS:2009:hni-igi` where the SD&S service provider discovery record was obtained by the OITF through the procedure defined in Section 5.3.1.2 of [PROT]. • The URL of the application was discovered directly through the procedure defined in Section 5.3.1.2 of [PROT].

readonly Boolean **systemReady**
Indicates whether the system has finished initialising. A value of `true` indicates that the system is ready.

readonly String **vendorName**
String identifying the vendor name of the device.

readonly String **modelName**
String identifying the model name of the device.

readonly String **softwareVersion**
String identifying the version number of the platform firmware.

readonly String **hardwareVersion**
String identifying the version number of the platform hardware.

readonly String **serialNumber**

String containing the serial number of the platform hardware.

readonly Integer **releaseVersion**
Release version of the OIPF specification implemented by the OITF. For instance, if the OITF implements release 2 version “1.0”, this property should be set to 2.

readonly Integer **majorVersion**
Major version of the OIPF specification implemented by the OITF. For instance, if the OITF implements release 2 version “2.0”, this property should be set to 2.

readonly Integer **minorVersion**
Minor version of the OIPF specification implemented by the OITF. For instance, if the OITF implements release 2 version “2.0”, this property should be set to 0.

readonly String **oipfProfile**
Profile of the OIPF specification implemented by the OITF. Values of this field are not defined in this specification.

readonly Boolean **pvrEnabled**
Flag indicating whether the platform has PVR capability (local PVR). Note: This property is deprecated in favour of the `pvrSupport` property.

readonly Boolean **ciplusEnabled**
Flag indicating whether the platform has CI+ capability.

Boolean **standbyState**
Get or set the standby state of the receiver. A value of `true` indicates that the receiver is in standby mode. Note - the property is deprecated in favour of the `powerState` property.

readonly Integer **powerState**
The `powerState` property provides the DAE application the ability to determine the current state of the

OITF. The property is limited to the ACTIVE_STANDBY or ON states.

Note this state deprecates the standbyState property.

readonly Integer **previousPowerState**

The previousPowerState property provides the DAE application the ability to retrieve the previous state.

readonly Integer **timeCurrentPowerState**

The time that the OITF entered the current power state. The time is represented in seconds since midnight (GMT) on 1/1/1970.

function **onPowerStateChange**(Integer powerState)

The function that is called when the power state has changed. The specified function is called with the argument powerState:

- Integer powerState – the new power state.

Integer **volume**

Get or set the overall system volume. Valid values for this property are in the range 0 - 100.

Boolean **mute**

Get or set the mute status of the default audio output(s). A value of true indicates that the default output(s) are currently muted.

readonly AVOutputCollection **outputs**

A collection of AVOutput objects representing the audio and video outputs of the platform. Applications MAY use these objects to configure and control the available outputs.

readonly NetworkInterfaceCollection **networkInterfaces**

A collection of NetworkInterface objects representing the available network interfaces.

readonly Integer **tvStandard**

Get the TV standard(s) for which the system is configured. This enables the user interface to only display those options relevant to the available TV standard(s).

This property can take one or more of the following values:

Value	Description
1	Indicates platform support for the NTSC TV standard.
2	Indicates platform support for the PAL TV standard.
4	Indicates platform support for the SECAM TV standard.

Values are stored as a bitfield.

`readonly Integer` **`pvrSupport`**

Flag indicating the type of PVR support used by the application. This property may take zero or more of the following values:

Value	Description
0	PVR functionality is not supported. This is the default value if <recording> as specified in Section 9.3.3 has value `false`.
1	PVR functionality is supported in the OITF. This is the default value if <recording> as specified in Section 9.3.3 has value `true`.

Values are stored as a bitfield.

7.3.3.3 Methods

`Boolean` **`setScreenSize`**`( Integer width, Integer height )`		
Description	Set the resolution of the graphics plane. If the specified resolution is not supported by the OITF, this method SHALL return `false`. Otherwise, this method SHALL return `true`.	
Arguments	*width*	The width of the display, in pixels.
	height	The height of the display, in pixels.

`Integer` **`setPvrSupport`**`( Integer state )`		
Description	Set the type of PVR support used by the application. The types of PVR supported by the receiver MAY not be supported by the application; in this case, the return value indicates the pvr support that has been set.	
Arguments	*state*	The type of PVR support desired by the application. More than one type of PVR functionality MAY be specified, allowing the receiver to automatically select the appropriate mechanism. Valid values are:

Value	Description

		<table><tr><td>0</td><td>PVR functionality is not supported. This is the default value if <recording> as specified in Section 9.3.3 has value `false`.</td></tr><tr><td>1</td><td>PVR functionality is supported in the OITF. This is the default value if <recording> as specified in Section 9.3.3 has value `true`.</td></tr></table> Values are stored as a bitfield.

`Boolean` **`setPowerState`**`( Integer type )`		
Description	The `setPowerState()` method allows the DAE application to modify the OITF state. The power state change may be restricted for some values of `type`, for example `OFF` and `PASSIVE_STANDBY`. A call to `setPowerState` with a restricted value of type SHALL return `false`.	
Arguments	*type*	The type values that may be specified are defined in Section 7.3.3.1

7.3.3.4 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onPowerStateChange`	`PowerStateChange`	Bubbles: No Cancelable: No Context Info: `powerState`

NOTE: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving the events listed above during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `LocalSystem` object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.3.4 The NetworkInterface class

The `NetworkInterface` class represents a physical or logical network interface in the receiver.

7.3.4.1 Properties

`readonly String` **`ipAddress`**
The IP address of the network interface, in dotted-quad notation for IPv4 or colon-hexadecimal notation for IPv6.

readonly String **macAddress**
The colon-separated MAC address of the network interface.

readonly Boolean **connected**
Flag indicating whether the network interface is currently connected.

Boolean **enabled**
Flag indicating whether the network interface is enabled. Setting this property SHALL enable or disable the network interface.

7.3.5 The AVOutput class

The AVOutput class represents an audio or video output on the local platform.

7.3.5.1 Properties

readonly String **name**
The name of the output. Each output SHALL have a name that is unique on the local system. At least one of the outputs SHALL have the name "all" and SHALL represent all available outputs on the platform.

readonly String **type**
The type of the output. Valid values are “audio”, “video”, or “both”.

Boolean **enabled**
Flag indicating whether the output is enabled. Setting this property SHALL enable or disable the output.

Boolean **subtitleEnabled**
Flag indicating whether the subtitles are enabled. The language of the displayed subtitles is determined by a combination of the value of the Configuration.preferredSubtitleLanguage property (see Section 7.3.2.1) and the subtitles available in the stream. For audio outputs, setting this property will have no effect.

String **videoMode**
Read or set the video format conversion mode, for which hardware support MAY be available on the

device, used when displaying a 4:3 signal on a 16:9 display. Valid values are:

- normal
- stretch
- zoom

The actual effect on the display, for example how bars are introduced when stretching an input video, depends on the value of this property, the aspect ratio of the display device and the aspect ratio of the input video (represented by the `aspectRatio` property of the appropriate instance of the `AVVideoComponent` class).

An OITF that does not support its own display (e.g. STB) may also signal over the interface (ex. HDMI and SCART) to the TV set which may also have effect on the actual display. This specification remains silent on the actual effect.

The DAE application graphical layer is unaffected by the `videoMode`.

For audio-only outputs, setting this property SHALL have no effect.

`String` **`digitalAudioMode`**

Set the output mode for digital audio outputs for which hardware support MAY be available on the device. Valid values are shown below.

Value	Behaviour
ac3	Output AC-3 audio.
uncompressed	Output uncompressed PCM audio.

For video-only outputs, setting this property SHALL have no effect.

`String` **`audioRange`**

Set the range for digital audio outputs for which hardware support MAY be available on the device. Valid values are shown below

Value	Behaviour
normal	Use the normal audio range.
narrow	Use a narrow audio range.
wide	Use a wide audio range.

For video-only outputs, setting this property SHALL have no effect.

`String` **`hdVideoFormat`**

Set the video format for HD video outputs for which hardware support MAY be available on the device.

Valid values are:

- 480i
- 480p
- 576i
- 576p
- 720p
- 1080i
- 1080p

For audio-only or standard-definition outputs, setting this property SHALL have no effect.

`String` **`tvAspectRatio`**

Indicates the display aspect ratio of the display device connected to this output for which hardware support MAY be available on the device. Valid values are:

- 4:3
- 16:9

Other values may be indicated but are not listed.

For audio-only outputs, setting this property SHALL have no effect.

`readonly StringCollection` **`supportedVideoModes`**

Read the video format conversion modes that may be used. See the definition of the `videoModes` property for valid values.

For audio outputs, this property will have the value `null`.

`readonly StringCollection` **`supportedDigitalAudioModes`**

Read the supported output modes for digital audio outputs. See the definition of the `digitalAudioMode` property for valid values.

For video outputs, this property will have the value `null`.

`readonly StringCollection` **`supportedAudioRanges`**

Read the supported ranges for digital audio outputs. See the definition of the `audioRange` property for valid values.

For video outputs, this property will have the value `null`.

readonly StringCollection **supportedHdVideoFormats**
Read the supported HD video formats. See the definition of the hdVideoFormat property for valid values. For audio outputs, this property will have the value null.

readonly StringCollection **supportedAspectRatios**
Read the supported TV aspect ratios. See the definition of the tvAspectRatio property for valid values. For audio outputs, this property will have the value null.

7.3.6 The NetworkInterfaceCollection class

```
typedef Collection<NetworkInterface> NetworkInterfaceCollection
```

The NetworkInterfaceCollection class represents a collection of NetworkInterface objects. See annex K for the definition of the collection template.

7.3.7 The AVOutputCollection class

```
typedef Collection<AVOutput> AVOutputCollection
```

The AVOutputCollection class represents a collection of AVOutput objects. See annex K for the definition of the collection template.

7.4 Content download APIs

This section defines the content-on-demand download interfaces for both DRM-protected and non-DRM protected content.

An OITF and a DAE application which have indicated support for downloading content by providing value "true" for element <download> in their capability profile as specified in Section 9.3.4 SHALL adhere to the following requirements.

NOTE: Annex D clarifies the purpose and the use of these interfaces in more detail.

7.4.1 The application/oipfDownloadTrigger embedded object

An OITF SHALL support a non-visual embedded object of type application/oipfDownloadTrigger, with the following Javascript API to enable passing a content-access descriptor to an underlying download manager using Javascript.

The functionality as described in this section is subject to the security model of Section 10.

7.4.1.1 Methods

String **registerDownload**(String contentAccessDownloadDescriptor, Date downloadStart)	
Description	Send contentAccessDownloadDescriptor to underlying download manager as a String formatted according to the Content Access Download Descriptor XML Schema as

	specified in Annex E. Returns a `String` value representing a unique identifier to identify the download, if the `contentAccessDownloadDescriptor` is valid and is accepted for triggering a download. If the OITF supports the `application/oipfDownloadManager` as specified in Section 7.4.3, this SHALL be value of the "`id`" attribute of the associated the `Download` object. Note that if the Content Access Download Descriptor contains multiple content items to be downloaded, the associated `Download` objects for each of these content items SHALL have the same value for the "`id`" value. The associated `Download` objects can be retrieved through the method `getDownloads()` as defined in Section 7.4.3.3. The OITF SHALL guarantee that download identifiers are unique in relation to recording identifiers and `CODAsset` identifiers. The method returns `undefined` if the `contentAccessDownloadDescriptor` is not accepted for triggering a download.	
Arguments	*contentAccessDownloadDescriptor*	String formatted according to the Content Access Download Descriptor XML Schema as specified in Annex E.
	downloadStart	Optional argument indicating the time at which the download should be started. If the argument is not included, or takes a value of `null` then the download should start as soon as possible.

`String` **`registerDownloadURL`**`( String URL, String contentType, Date downloadStart )`	
Description	This method triggers the OITF to initiate a download of the content pointed to by the URL and the given content type. The `contentType` attribute SHALL reflect the expected type of content returned by the content server when connecting to the URL. The `contentType` can be used to evaluate if the content type is part of the list of accepted content types of the OITF. For example, if the OITF does not support content type "`video/MP2T`", then the `registerDownloadURL` method could return `undefined` to indicate this to the application in advance of the download. If `contentType` has value "`application/vnd.oipf.ContentAccessDownload+xml`", the method SHALL return a download identifier, after which the OITF SHALL immediately fetch the Content Access Download Descriptor, after which the same SHALL happen as if `registerDownload()` as defined in Section 4.6.3.1 with the given Content Access Download Descriptor as argument was called. The downloadStart argument only applies to the individual Download objects described by the Content Access Download Descriptor and SHALL NOT apply to the retrieval of the Content Access Download Descriptor itself. Note that if the Content Access Download Descriptor contains multiple content items to be downloaded, the associated `Download` objects for each of these content items SHALL have the same value for the "`id`" value. The associated `Download` objects can be retrieved through method `getDownloads()` as defined in Section 7.4.3.3. Returns a `String` value representing a unique identifier to identify the download, if the given arguments are acceptable by the OITF to trigger a download. If the OITF supports the `application/oipfDownloadManager` as specified in Section 7.4.3, this SHALL be the value of the "`id`" attribute of the associated `Download` object(s). The OITF SHALL guarantee that download identifiers are unique in relation to recording

	identifiers and `CODAsset` identifiers. The method returns `undefined` if the given arguments are not acceptable by the OITF to trigger a download.	
Arguments	*URL*	The URL from which the content can be fetched.
	contentType	The type of content referred to by the URL attribute. The `contentType` can be used to evaluate if the content type is part of the list of supported content types of the OITF.
	downloadStart	Optional argument indicating the time at which the download should be started. If the argument is not included, or takes a value of `null` then the download should start as soon as possible.

`Integer` **`checkDownloadPossible`**`( Integer sizeInBytes )`		
Description	Checks whether a download of a given `sizeInBytes` would be possible at this moment in time. Possible return values are:	

Value	Semantics
0	Successful, i.e. the download could be successfully completed if it would be started at this moment in time.
1	Insufficient Storage, i.e. the download could be started, but is unlikely to complete successfully, since insufficient storage capacity is available to fully store the content to be downloaded.
2	Storage not available, i.e. the download would fail, since the storage is currently unavailable, e.g. in case of removable storage.

Arguments	*sizeInBytes*	Integer value with the given size of the download in bytes.

7.4.2 Extensions to application/oipfDownloadTrigger

If an OITF has indicated support for both BCG metadata (i.e. by giving element `<clientMetadata>` value "`true`" and a `type` attribute with value "`bcg`"), and the download management APIs defined in Section 7.4.3 (i.e. by giving attribute `manageDownloads` of the `<download>` element a value unequal to "`none`") in the client capability description, then the following additional method SHALL be supported by the `application/oipfDownloadTrigger` object defined in Section 7.4.1

The functionality as described in this section is subject to the security model of Section 10.

`String` **`registerDownloadFromCRID`**`( String CRID, String IMI, Date downloadStart )`	
Description	Send `(CRID,IMI)` to underlying download manager. Returns a `String` value representing a unique identifier to identify the download if `the (CRID,IMI) tuple` is valid and is accepted for triggering a download. If the OITF supports the `application/oipfDownloadManager` as specified in Section 7.4.3, this SHALL be the value of the "`id`" attribute of the associated `Download` object(s), which corresponds to the `CRID` in this case.

	The OITF SHALL guarantee that download identifiers are unique in relation to recording identifiers and `CODAsset` identifiers. The method returns `undefined` if the given `(CRID, IMI)` tuple is not accepted for triggering a download. The values of the `name`, `description`, `parentalRating` and `DRMControl` properties SHALL be based on the metadata provided for the item matching that CRID.	
Arguments	*CRID*	The TV-Anytime Content reference ID that points to the general information about the item to download that does not change regardless of how the content is published or broadcast
	IMI	The TV-Anytime Instance Metadata ID that points to the specific information related to the item to download, such as content location, usage rules (pay-per-view, etc.) and delivery parameters (e.g. video format).
	downloadStart	Optional argument indicating the time at which the download should be started. If the argument is not included, or takes a value of `null` then the download should start as soon as possible.

7.4.3 The application/oipfDownloadManager embedded object

In a managed network, privileged applications may need access to the download management functionality in a CoD system. This access may be required to implement a UI to the download manager, to queue a download or to display the progress of a specific download. OITFs SHOULD support an "`application/oipfDownloadManager`" object with the following interface.

Clients supporting the download management APIs as specified in this section SHALL indicate this by adding the attribute `manageDownloads` to the <`download`> element with a value unequal to "`none`" in the client capability description as defined in Section 9.3.4.

The functionality as described in this section is subject to the security model of Section 10.

7.4.3.1 State diagram for the application/oipfDownloadManager object

The following state machine provides an overview of the state changes that may occur in the download manager. The states reflect the changes signalled to applications via the `onDownloadStateChange` event handler.

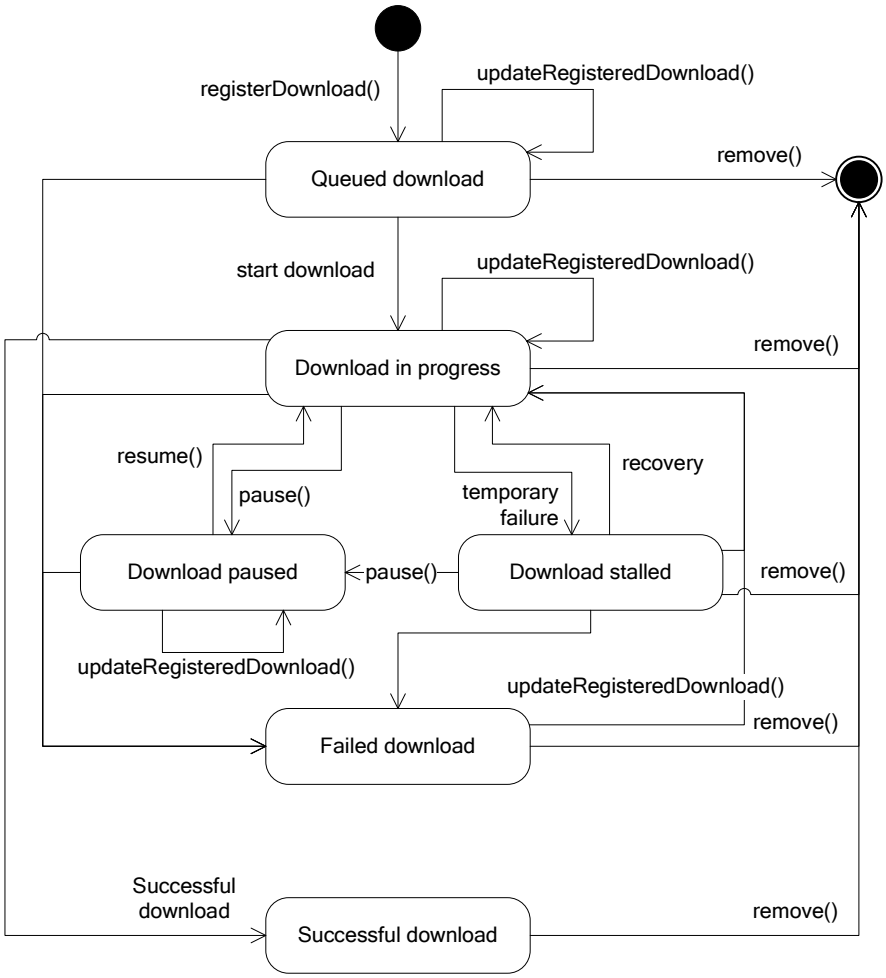


Figure 14: State diagram for embedded `application/oipfDownloadManager` objects

7.4.3.2 Properties

`function` **`onDownloadStateChange`**`( Download item, Integer state, Integer reason )`
The function that is called when the status of a download has changed. The specified function is called with three arguments `item`, `state` and `reason`, which are defined as follows: • `Download item` – the Download object whose state has changed. • `Integer state` – the new state of the download. Valid values are given in Section 7.4.4.1. • `Integer reason`. Extended reason code. This is only valid if the value of the `state` argument is 8. Valid values are given in Section 7.4.4.2. If no error has occurred, this argument SHALL take the value `undefined`.

`readonly DiscInfo` **`discInfo`**
Get information about the status of the local storage device. The `DiscInfo` class is defined in Section 7.16.4.

7.4.3.3 Methods

Boolean **pause**(Download download)		
Description	Pause an in-progress, queued or stalled download and return `true`. For in-progress downloads, more data SHALL NOT be downloaded until the download is resumed. The HTTP request and TCP socket are interrupted and closed. For completed or failed downloads, this operation SHALL return `false`.	
Arguments	*download*	The download to be paused.

Boolean **resume**(Download download)		
Description	Resume a paused download. If the download is not paused, this operation SHALL return `false`.	
Arguments	*download*	The download to be resumed.

Boolean **remove**(Download download)		
Description	Remove the download and any data and media content associated with it and return `true`. Return `false` if the `download` attribute does not refer to a valid download. As a side effect of this method, all properties on `download` SHALL be set to `undefined`. Any method calls subsequently performed by an application which pass `download` as an argument SHALL return `false`. If an A/V Control object is currently playing the specified `Download` object, the A/V Control object SHALL transition to the error state.	
Arguments	*download*	The download to be deleted.

DownloadCollection **getDownloads**(String id)		
Description	Returns a collection of downloads, for which the value of the `Download.id` property corresponds to the given `id` parameter. The downloads returned in the collection SHALL be filtered according to the value of the `manageDownloads` attribute of the <download> element in the OITF's capability description (i.e. from the same application, same domain or from all applications). For downloads initiated from registerDownloadURL() with a contentType value "`application/vnd.oipf.ContentAccessDownload+xml`" SHALL return null until the Content Access Download Descriptor has been retrieved and parsed. If the value of id is `null`, it returns all downloads for the scope indicated by the `manageDownloads` attribute.	
Arguments	*id*	Optional argument identifying the downloads to be retrieved. If present and not `null`, this is an identifier corresponding to the "`id`" attribute of zero or more Download objects. If the value of `id` is `null`, or the argument is not included, all downloads for the scope indicated

		by the `manageDownloads` attribute in the capability description are returned.

`DownloadCollection` **`createFilteredList`**`( Boolean currentDomain, Integer states )`		
Description	Create a filtered list of downloads. Returns a subset of downloads that are managed by the receiver. The `currentDomain` flag indicates whether downloads from FQDNs other than the current page are included in the returned collection. This flag MAY be set to one of three values: **Value** / **Meaning** `true` — The download is added if and only if it was initiated from the FQDN of the calling document. If the application has the permission *permission_downloadmanager* (see Section 10.1.5), only downloads initiated by the calling application shall be added. `false` — The download is added if and only if it was not initiated from the FQDN of the calling document. If the application does not have the permission *permission_downloadmanager_all* (see Section 10.1.5), the OITF SHALL return an empty collection. `undefined` — The download is added regardless of the domain that the download was initiated from. If the application has the permission *permission_downloadmanager* (see Section 10.1.5), only downloads initiated by the calling application shall be added. If the application has the permission *permission_downloadmanager_samedomain* (see Section 10.1.5), only downloads initiated by applications from the same FQDN shall be added. The `states` flag indicates which state(s) of downloads that should be included in the list. The value of this flag is the arithmetic sum of one or more possible values of the `state` property of the `Download` object; only downloads whose state matches one of the values included in this sum are included in the returned collection.	
Arguments	*currentDomain*	Flag indicating whether downloads from other domains SHALL be added to the list.
	states	Indicates that states of downloads that should be included in the returned list.

`Boolean` **`updateRegisteredDownload`**`( Download download, string newURL )`

Description	The method `updateRegisteredDownload()` provides a way to update the URL to be used for a download. The OITF SHALL use the new URL for any future retrieval. If the download is already in progress or paused (indicated by a `state` property value of 4), it SHALL be stopped. The download SHALL continue from the last byte received during the previous download. If the `state` property of the `download` argument has the value `Download.DOWNLOAD_FAILED` or `Download.DOWNLOAD_STALLED` then the OITF SHALL resume the download from the last byte received during the previous download but using the new URL. If the `state` property of the `download` argument has the value `Download.DOWNLOAD_NOT_STARTED` no further action is taken until the download is started or resumed. If the `state` property of the `download` argument has the value `Download.DOWNLOAD_COMPLETED` then this method SHALL return `false`. Otherwise it SHALL return true.	
Arguments	*download*	The download object to be updated.
	newURL	The new URL from which the content can be retrieved.

7.4.3.4 Events

For the intrinsic event "`onDownloadStateChange`", a corresponding DOM level 2 event SHALL be generated, in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onDownloadStateChange`	`DownloadStateChange`	Bubbles: No Cancelable: No Context Info: `item, state, reason`

NOTE: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a `DownloadStateChange` event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `application/oipfDownloadManager` object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.4.4 The Download class

A `Download` object being made available by the `application/oipfDownloadManager` embedded object represents a content item that has either been downloaded from a remote server or is in the process of being downloaded.

If the ID of a download is a TV-Anytime CRID, then the values of the `name`, `description` and `parentalRating` properties SHALL be set by the OITF based on the metadata provided for the item matching that CRID.

In order to preserve backwards compatibility with already existing DAE content the ECMAScript `toString()` method MUST return the `Download.id` for `Download` objects.

7.4.4.1 Constants

The following constants are defined as properties of the `Download` class:

Name	Value	Use
DOWNLOAD_COMPLETED	1	The download has completed.
DOWNLOAD_IN_PROGRESS	2	The download is in progress.
DOWNLOAD_PAUSED	4	The download has been paused (either by an application or automatically by the platform)
DOWNLOAD_FAILED	8	The download has failed.
DOWNLOAD_NOT_STARTED	16	The download is queued but has not yet started.
DOWNLOAD_STALLED	32	The download has stalled due to a transient failure and the Download Manager is attempting to recuperate and re-establish the download.

7.4.4.2 Properties

`readonly Integer` **`totalSize`**

The total size (in bytes) of the download.

`readonly Integer` **`state`**

The current state of the download. When this changes, a `DownloadStateChange` event SHALL be generated. Valid values are given in Section 7.4.4.1 above.

`readonly Integer` **`reason`**

The reason property is only valid if the value of the state property is `DOWNLOAD_FAILED`.

Reason	Semantics
0	The local storage device is full.
1	The item cannot be downloaded (e.g. because it has not been purchased).
2	The item is no longer available for download.
3	The item is invalid due to bad checksum or length.
4	Other reason.

If no error has occurred, this argument SHALL take the value undefined.□

readonly Integer **amountDownloaded**
The amount of data that has been downloaded returned in bytes, or zero if no data has been downloaded.

readonly Integer **currentBitrate**
The bitrate (in bits per second) at which the download is currently transferred. This value is non-zero only when the Download object is in the DOWNLOAD_IN_PROGRESS state. If this is unknown the value of this property SHALL be undefined.

String **name**
The name of the download or undefined if this information is not present. In case the download is triggered through a Content Access Download Descriptor, this corresponds to the value for the <Title> element in the Content Access Download Descriptor. If the Content Access Download Descriptor is not specified the property may be set by the origin site. Note that the property may only be set by the site that initiated the download. The DAE application may store data related to the Download. The OITF SHALL support a minimum of 200 bytes for the property. If DAE application attempts to store a string larger than the available size the OITF SHALL set the property to NULL. The maximum length of the property value is implementation dependent.

readonly String **id**
The ID of the download as determined by the OITF.

readonly String **contentURL**
The URL the content is being fetched from, or undefined if this information is not available.

String **description**
A description of the download or undefined if this information is not present. In case the download is triggered through a Content Access Download Descriptor, this corresponds to the value for the <Synopsis> element in the Content Access Download Descriptor, or undefined if this element is not present. If the Content Access Download Descriptor is not specified the property may be set by the origin site. Note that the property may only be set by the site that initiated the download. The DAE application may store data related to the Download. The OITF SHALL support a minimum of 2000 bytes for the property. If DAE application attempts to store a string larger than the available size the OITF SHALL set the property to NULL. The maximum length of the property value is implementation dependent.

readonly ParentalRatingCollection **parentalRatings**
The parental rating collection related to the downloaded content item, or undefined if this information

is not present. In case the download is triggered through a Content Access Download Descriptor, this corresponds to the value for the <ParentalRating> element in the Content Access Download Descriptor, or undefined if this element is not present

readonly DRMControlInfoCollection **drmControl**
The DRMControlInformation object corresponding to the DRM Control information of the downloaded content item, or undefined if this information is not present. In case the download is triggered through a Content Access Download Descriptor, this corresponds to the value for the <DRMControlInformation> element associated with the same DRMSystemID of the selected <ContentURL>, or is undefined if this information is not present. The related DRMControlInformation object is defined in Section 7.4.6.

readonly Date **startTime**
The time that the download is scheduled to start (in the case of scheduled downloads) or undefined if no start time was set.

readonly Integer **timeElapsed**
The time (in seconds) that has elapsed since the download of the item was started. The elapsed time SHALL be based on the time spent in the in-progress and stalled download states. This SHALL NOT include any time the item spent queued for download.

readonly Integer **timeRemaining**
The estimated time remaining (in seconds) for the download to complete. The estimated time SHALL be based on the time spent in the in-progress and stalled download states. The estimate SHALL NOT includes any time the item spent queued for download or paused. If an estimate cannot be calculated, the value of this property SHALL be undefined.

readonly String **transferType**
In case the download was triggered through a Content Access Download Descriptor, this is the value of property TransferType of the selected <ContentURL>. In the case where the download was not triggered through a content access descriptor document, the OITF is responsible for returning either the value “playable_download” or “full_download”, based on criteria defined by the OITF.

readonly String **originSite**
In the case where the download was triggered through a Content Access Download Descriptor, this is the value of element <OriginSite>. In case the download was not triggered through a content access descriptor document, this is the FQDN of the site that initiated the download.

readonly String **originSiteName**
In case the download is triggered through a Content Access Download Descriptor, this is the value of

element <OriginSiteName>, or undefined if this information is not present. In case the download is not triggered through a content access descriptor document, this property is undefined.

String **contentID**

A unique identification of the content item relative to originSite. In case the download is triggered through a Content Access Download Descriptor, and a <ContentID> element has been defined for the given content item, this is the value of element <ContentID>. If the download is started using registerDownloadFromCRID(), this is the TV Anytime CRID. This property shall take the value undefined if no content ID is available.

If the Content Access Download Descriptor is not specified the property may be set by the originSite. Note that the property may only be set by the site that initiated the download. The DAE application may store data related to the Download. The OITF SHALL support a minimum of 2000 bytes for the property. If DAE application attempts to store a string larger than the available size the OITF SHALL set the property to NULL. The maximum length of the property value is implementation dependent.

readonly String **iconURL**

The URL of an image that provides a visual representation of the item that is being downloaded. In the case where the download was triggered a Content Access Download Descriptor, this is the value of element <IconURL>, or undefined if this element is not present. In the case where the download was not triggered through a content access descriptor document, this property is undefined.

7.4.5 The DownloadCollection class

```
typedef Collection<Download> DownloadCollection
```

The DownloadCollection class represents a collection of Download objects. See annex K for the definition of the collection template.

7.4.6 The DRMControlInformation class

A DRMControlInformation object represents the DRM Control information structure defined in §3.3.2 of [OIPF_META2].

7.4.6.1 Properties

readonly String **drmType**

URN consisting of the DRM system's DVB CASystemID value (expressed as a decimal integer), prefixed with the string "urn:dvb:casystemid:". For example, the value of this property for Marlin is "urn:dvb:casystemid:19188".

readonly String **rightsIssuerURL**

A URL used by OITF to obtain rights for this content item.

readonly String **silentRightsURL**

A URL used by OITF to obtain rights silently, e.g. a Marlin Action Token.

readonly String **drmContentID**
DRM Content ID for CoD or scheduled content item, e.g. the Marlin Content ID.

readonly String **previewRightsURL**
A URL used by OITF to obtain rights silently for preview of this content item, e.g. a Marlin Action Token.

readonly String **drmPrivateData**
Private data for the DRM scheme indicated in drmType to be applied for this content item. Private DRM Data is actually structured as an XML document whose schema is specific to the considered DRM system. One example is Marlin DRM private data schema defined in [OIPF_CSP2].

readonly Boolean **doNotRecord**
A flag indicating whether this content item is recordable or not.

readonly Boolean **doNotTimeShift**
A flag indicating if this content item is allowed for time shift play back.

7.4.7 The DRMControlInfoCollection class

```
typedef Collection<DRMControlInfo> DRMControlInfoCollection
```

The DRMControlInfoCollection class represents a collection of DRMControlInfo objects. See annex K for the definition of the collection template.

7.5 Content On Demand Metadata APIs

This section SHALL apply for OITFs that have indicated <clientMetadata> with value "true" and a type attribute with value "bcg" in the capability description and MAY apply for OITFs that have indicated <clientMetadata> with value "true" and a type attribute with value "dvb-si"

7.5.1 The application/oipfCodManager embedded object

OITFs that have indicated <clientMetadata> with value "true" and a type attribute with value "bcg" SHALL implement an "application/oipfCodManager" embedded object with the following interface.

Content is organised into catalogues, where each catalogue contains a hierarchy of folders that are used to organise individual content items. The structure of the catalogue SHALL be determined by the server managing that catalogue and SHALL be reflected in the structure of the metadata passed to the OITF.

The three types of content in a CoD catalogue are:

- Assets, represented by the `CODAsset` class. A `CODAsset` is a user-level description of a piece of CoD content, and so it is more concerned with information such as the price, rental period, description and parental rating rather than detailed technical information about the asset such as encoding format. A CoD asset MAY represent a single movie, or a bundle of movies offered for a single price.
- Folders, represented by the `CODFolder` class.
- Services, represented by the `CODService` class. `CODService` objects are a specific type of container representing subscription VoD (SVOD) services, where users purchase a group of assets which may change over time rather than a single movie or TV show.

The `CODAsset`, `CODFolder` and `CODService` classes define a type property that allows these classes to be distinguished by applications. For each class, this property SHALL take the value defined below:

Class	Value
`CODFolder`	0
`CODAsset`	1
`CODService`	2

This specification defines the mapping between the CoD API and BCG metadata. Mappings between the CoD API and other CoD metadata sources are not specified in this document.

7.5.1.1 Properties

`readonly ContentCatalogueCollection` **`catalogues`**

A collection of all available CoD catalogues, as listed in an SD&S BCG Discovery record.

`function` **`onContentCatalogueEvent`**`( Integer action )`

This function is the DOM 0 event handler for events relating to changes in a content catalogue collection. The specified function is called with the argument `action`:

- `Integer action` - The type of event. For current versions of the specification, this property SHALL always have the value 0 to indicate a change in the list of available catalogues.

`function` **`onContentAction`**`( Integer action, Integer result, Object item, ContentCatalogue catalogue )`

This function is the DOM 0 event handler for events relating to actions carried out on an item in a content catalogue. The specified function is called with the following arguments:

- `Integer action` - The type of action that the event refers to. Valid values are:

Value	Description
0	An operation to browse a content collection (e.g. getting a page from the collection).

1	Indicates that more information is available about this item (e.g. that more information has been retrieved from the server).

- `Integer result` - The result of the action. Valid values are:

Value	Description
0	The operation succeeded.
1	The item no longer exists in the catalogue.
2	The server has not responded in the timeout period.
3	Communication with the server has been interrupted.

- `Object item` - The item in the catalogue that the event refers to.
- `ContentCatalogue catalogue` - The parent catalogue of the affected object.

7.5.1.2 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onContentCatalogueEvent`	`ContentCatalogueEvent`	Bubbles: No Cancelable: No Context Info: `action`
`onContentAction`	`ContentAction`	Bubbles: No Cancelable: No Context Info: `action`, `result`, `item`, `catalogue`

NOTE: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving the events listed above during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `LocalSystem` object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.5.2 The CatalogueCollection class

```
typedef Collection<ContentCatalogue> CatalogueCollection
```

The CatalogueCollection class represents a collection of ContentCatalogue objects. See annex K for the definition of the collection template.

7.5.3 The ContentCatalogue class

A ContentCatalogue represents a content catalogue for a content on demand service.

To receive events relating to operations on items in a catalogue, applications MAY add listeners for "ContentAction" events to the application/oipfCodManager object.

7.5.3.1 Properties

readonly String **name**
The name of the content catalogue that should be displayed to the user. The value of this property is given by the Name element in the catalogue's BCG discovery record.

readonly CODFolder **rootFolder**
The root folder of the content catalogue.

7.5.3.2 Methods

CODFolder **getPurchaseHistory**()	
Description	Get the list of items that have been purchased from the catalogue by the current user, including currently active rentals. Items in this list will be derived from children of the BCG UserActionList element which have an ActionType of buy. If the ActionList element is not present, this method SHALL return null.

7.5.4 The CODFolder class

CODFolder represents a folder in a CoD catalogue. Folders may contain other folders, and an asset may be present in more than one folder.

Because a content list may contain a large number of items, the contents of the list are made available on demand using a paging model. Applications MAY request the contents of the list in 'pages' of an arbitrary size. The data SHALL be fetched from the appropriate source, and application SHALL be notified when the data is available.

Each folder is described by a GroupInformation element in the BCG Group Information Table.

7.5.4.1 Properties

readonly Integer **type**
The type of the item, used to distinguish between the types of objects that may be contained in a folder in a CoD catalogue. This SHALL always have the value 0 for folders.

readonly String **uri**
The URI used to refer to the folder. The value of this property is given by the GroupId attribute of the GroupInformation element representing this folder.

readonly String **name**
The name of the folder. The value of this property is given by the Title element that is a descendant of the GroupInformation element representing this folder.

readonly String **description**
A description of the folder, for display to an end user. The value of this property is given by the Synopsis element that is a descendant of the GroupInformation element representing this folder.

readonly String **thumbnailUri**
The URI of an image associated with this folder. For assets whose BCG description contains a RelatedMaterial element indicating a relationship of Promotional Still Image, the value of this property is given by the MediaURI element that is a descendant of that element. For assets without an appropriate RelatedMaterial element, the value of this property SHALL be undefined.

readonly Integer **length**
The number of items in the current page. If getPage() has not yet been called, the value of this property SHALL be undefined.

readonly Integer **currentPage**
The page number of the currently-available results, as specified in the last call to getPage(). If getPage() has not yet been called, the value of this property SHALL be undefined.

readonly Integer **pageSize**

The number of items that were requested from the content catalogue in a call to `getPage()`. This MAY be different form the number of items that are available (e.g. the last page in the collection). If `getPage()` has not yet been called, the value of this property SHALL be `undefined`.

`readonly Integer` **`totalSize`**
The total number of items in the folder. This MAY be `undefined` until `getPage()` has been called. The value of this property may be given by the `numOfItems` attribute of the `GroupInformation` element representing this folder.

7.5.4.2 Methods

`Object` **`item`**`( Integer index )`		
Description	Return the item at position index in the current page, or `undefined` if no item is present at that position. This function SHALL only return objects that are instances of `CODAsset`, `CODFolder`, or `CODService`. Applications SHALL be able to access items in the collection using array notation instead of calling this method directly.	
Arguments	*index*	The index into the collection.

`void` **`getPage`**`( Integer page, Integer pageSize )`		
Description	Retrieve one page of the folder's contents. The application SHALL be notified by an event targeted at the folder's parent content catalogue when the data is available. Calls to this method SHALL cancel any outstanding requests.	
Arguments	*page*	The number of the page for which data should be retrieved, indexed from zero.
	pageSize	The size of the page.

`void` **`abort`**`()`
Description	Abort the current request for a new page of folder contents. Any results for this folder SHALL be removed (i.e. the value of the `length` property will be 0 and any calls to the `item()` method SHALL return `undefined`),

7.5.5 The CODAsset class

The `CODAsset` represents a piece of CoD content that can be purchased and played. A `CODAsset` object MAY refer to a bundle of content items that are purchased together but which can only be played individually.

Some fields of a `CODAsset` object MAY not be populated until an application requests them; in this case the data MAY be fetched asynchronously from a server. Fields where the data has not been fetched from the server SHALL have a value of `undefined`. Fields for which data is not available on the server SHALL have a value of `null`.

7.5.5.1 Properties

`readonly Integer` **`type`**
The type of the item, used to distinguish between the types of objects that may be contained in a folder in a CoD catalogue. This property SHALL always have the value 1 for CoD assets.

`readonly String` **`uri`**
The CRID of the asset. The value of this property is given by the `programId` attribute of the BCG `ProgramInformation` element that describes the asset.

`readonly String` **`name`**
The title of the asset that will be displayed to the user. The value of this property is given by the BCG `Title` element that is a child of the asset's `BasicDescription` element.

`readonly String` **`description`**
A description of the asset, for display to an end user. The value of this property is given by the BCG `Synopsis` element that is a child of the asset's `BasicDescription` element.

`readonly StringCollection` **`genres`**
A collection of genres that describe this asset. Genres are represented by the values of any `Name` elements that are children of `Genre` elements in the asset's description.

`readonly ParentalRating` **`parentalRating`**
The parental rating value of the asset. This information will be read from the `ParentalGuidance` element of an asset's description, if present.

`readonly Boolean` **`blocked`**
Flag indicating whether the asset is blocked due to parental control settings (i.e. whether its parental rating value exceeds the current system threshold).

`readonly Boolean` **`locked`**
Flag indicating whether the current state of the parental control system prevents the asset from being

viewed (e.g. a correct parental control PIN has not been entered to allow the item to be viewed).

readonly String **thumbnailUri**

The URI of an image associated with this asset.

For assets whose BCG description contains a `RelatedMaterial` element indicating a relationship of `Promotional Still Image`, the value of this property is given by the `MediaURI` element that is a descendant of that element.

For assets without an appropriate `RelatedMaterial` element, the value of this property SHALL be `undefined`.

readonly String **price**

The price of the asset, in a form that can be displayed to the user. The value of this property is the concatenation of the value of the `Price` element that is a child of a `PurchaseItem` element in the asset's description and the value of the `Price` element's `currency` attribute.

For example, a `Price` element of

```
<Price currency="JPY">500</Price>
```

would give the value `500 JPY` for this field. Implementations MAY replace the currency code with the appropriate currency symbol (e.g. ¥).

readonly Integer **rentalPeriod**

The period for which the asset can be rented, in hours.

For assets descriptions containing a `Purchase` element with a `PurchaseType` of `urn:tva:metadata:cs:PurchaseTypeCS:2004:playForPeriod`, the value of this property is derived from the `QuantityUnit` and `QuantityRange` elements that are children of that `Purchase` element. If a `Purchase` element with the appropriate `PurchaseType` is not present, the value of this field SHALL be `undefined`.

readonly Integer **playCount**

The number of plays allowed for this asset when it is purchased.

For assets descriptions containing a `Purchase` element with a `PurchaseType` of `urn:tva:metadata:cs:PurchaseTypeCS:2004:playCounts`, the value of this property is derived from the `QuantityUnit` and `QuantityRange` elements that are children of that `Purchase` element. If a `Purchase` element with the appropriate `PurchaseType` is not present, the value of this field SHALL be `undefined`.

readonly Integer **duration**

The duration of the asset, in seconds. The value of this property is given by the BCG `Duration` element that is a child of the asset's `BasicDescription` element.

readonly String **previewUri**
The URI used to refer to a preview of the asset. For assets whose BCG description contains a `RelatedMaterial` element indicating a relationship of `Trailer` or `Preview`, the value of this property is given by the `MediaURI` element of the `MediaLocator` contained in that element. For assets without an appropriate `RelatedMaterial` element, the value of this property SHALL be `undefined`.

readonly BookmarkCollection **bookmarks**
A collection of the bookmarks set in a recording. If no bookmarks are set, the collection SHALL be empty.

7.5.5.2 Methods

Boolean **isReady**()	
Description	Check whether sufficient information is available to make a purchase or play the asset. Due to the asynchronous nature of CoD catalogues, not all of the information required to play or purchase a CoD asset may have been received by the OITF at any given time. If all of the required information is available, this method SHALL return `true`. Otherwise, this method SHALL request the missing information and return `false`. When the information is available, the application SHALL be notified via a `ContentActionEvent` with the reason code 1.

StringCollection **lookupMetadata**(String key)		
Description	Retrieve metadata for the asset. Metadata is stored as key/value pairs - retrieving the metadata for a specified key SHALL return all values that match that key.	
Arguments	*key*	The key for the metadata to be returned.

7.5.6 The CODService class

The `CODService` class is a subclass of `CODFolder` that represents a subscription CoD service. A subscription CoD service is similar to a folder, except that:

- The service SHALL be purchased in its entirety, rather than purchasing individual items from the service.
- Business rules may prevent browsing of the content within a service unless the service has already been purchased.

A `CODService` MAY contain a number of assets, folders and services.

7.5.6.1 Properties

readonly Integer **length**

The number of items in the current page of the service.

`readonly Integer` **`currentPage`**

The page number of the currently-available results, as specified in the last call to `getPage()`. If `getPage()` has not yet been called, the value of this property SHALL be `undefined`.

`readonly Integer` **`pageSize`**

The number of items that were requested from the content catalogue in a call to `getPage()`. This MAY be different from the number of items that are available (e.g. the last page in the collection).

If `getPage()` has not yet been called, the value of this property SHALL be `undefined`.

`readonly Integer` **`totalSize`**

The total number of items in the service. This MAY be `undefined` until `getPage()` has been called.

The value of this property may be given by the `numOfItems` attribute of the `GroupInformation` element representing this folder.

`readonly Integer` **`type`**

The type of the item, used to distinguish between the types of objects that may be contained in a folder in a CoD catalogue. This property SHALL always have the value 2 for a CoD service.

`readonly String` **`uid`**

An ID for the service.

Folders, CoD services and CoD assets each have an ID which is unique within their parent catalogue. The value of this property is given by the `serviceId` attribute of the BCG `ServiceInformation` element that describes the service.

`readonly String` **`uri`**

The URI used to refer to the service. The value of this property is given by the BCG `ServiceURL` element that is a child of the `ServiceInformation` element that describes the service.

`readonly String` **`name`**

The name of the service that will be displayed to the user. The value of this property is given by the BCG `Name` element that is a child of the `ServiceInformation` element describing the service.

`readonly String` **`description`**

A description of the service, for display to an end user. The value of this property is given by the BCG `ServiceDescription` element that is a child of the `ServiceInformation` element describing the service.

`readonly String` **`thumbnailUri`**
The URI of an image associated with this service. The value of this property is derived from the value of the first `Logo` element that is a child of the BCG `ServiceInformation` element describing the service. If this element specifies anything other than the URL of an image, the value of this property SHALL be `undefined`. Alternatively, for services whose BCG description contains a `RelatedMaterial` element indicating a relationship of `Promotional Still Image`, the value of this property is given by the `MediaURI` element of the `MediaLocator` contained in that element. For assets without an appropriate `RelatedMaterial` or `Logo` element, the value of this property shall be `undefined`.

`readonly String` **`previewUri`**
The URI used to refer to a preview of the content. For services whose BCG description contains a `RelatedMaterial` element indicating a relationship of `Trailer` or `Preview`, the value of this property is given by the `MediaURI` element of the `MediaLocator` contained in that element. For services without an appropriate `RelatedMaterial` element, the value of this property SHALL be `undefined`.

7.5.6.1.1 Methods

`Object` **`item`**`( Integer index )`		
Description	Return the item at position index in the current page, or `undefined` if no item is present at that position. This function SHALL only return objects that are instances of `CODAsset`, `CODFolder`, or `CODService`. Applications SHALL be able to access items in the collection using array notation instead of calling this method directly.	
Arguments	*index*	The index into the collection.

`void` **`getPage`**`( Integer page, Integer pageSize )`		
Description	Retrieve one page of the services contents. The application SHALL be notified by an event targeted at the services parent content catalogue when the data is available. Calls to this method SHALL cancel any outstanding requests.	
Arguments	*page*	The number of the page for which data should be retrieved, indexed from zero.

	pageSize	The size of the page.

`void` **`abort`**`()`	
Description	Abort the current request for a new page of contents. Any results SHALL be removed (i.e. the value of the `length` property will be 0 and any calls to the `item()` method SHALL return `undefined`),

`Boolean` **`isReady`**`()`	
Description	Check whether sufficient information is available to make a purchase. Due to the asynchronous nature of CoD catalogues, not all of the information required to play or purchase a CoD service may have been received by the OITF at any given time. If all of the required information is available, this method SHALL return `true`. Otherwise, this method SHALL request the missing information and return `false`. When the information is available, the application SHALL be notified via a `ContentActionEvent` with the action code 1.

`StringCollection` **`lookupMetadata`**`( String key )`		
Description	Retrieve metadata for the service. Metadata is stored as key/value pairs - retrieving the metadata for a specified key SHALL return all values that match that key.	
Arguments	*key*	The key for the metadata to be returned.

7.6 Content Service Protection API

The following requirements SHALL apply to OITF and/or server devices which have indicated support for DRM protection by providing one or more `<drm>` elements as specified in Section 9.3.10:

7.6.1 The application/oipfDrmAgent embedded object

An OITF SHALL support a non-visual embedded object of type "`application/oipfDrmAgent`", with the following Javascript API, to enable in-session message exchange from the web page with an underlying DRM agent.

Access to the functionality of the `application/oipfDrmAgent` embedded object SHALL adhere to the security requirements as defined in Section 10.1

Note: Annex D provides a clarification to the browser interaction model when dealing with services offering protected content

7.6.1.1 Properties

`function` **`onDRMMessageResult`**`( String msgID, String resultMsg,` `Integer resultCode )`
The function that is called when the underlying DRM agent has a result message to report to the current HTML document as a consequence of a call to `sendDRMMessage`. The specified function is

called with three arguments `msgID, resultMsg` and `resultCode` which are defined as follows:

- `String msgID` – identifies the original message which has led to this resulting message.
- `String resultMsg` – DRM system specific result message.
- `Integer resultCode` – result code. Valid values include:

Result message	Description	Semantics
0	Successful	The action(s) requested by `sendDRMMessage()` completed successfully.
1	Unknown error	`sendDRMMessage()` failed because an unspecified error occurred.
2	Cannot process request	`sendDRMMessage()` failed because the DRM agent was unable to complete the request.
3	Unknown MIME type	`sendDRMMessage()` failed, because the specified Mime Type is unknown for the specified DRM system indicated in the DRMSystemId.
4	User Consent Needed	`sendDRMMessage()` failed because user consent is needed for that action.
5	Unknown DRM system	`sendDRMMessage()` failed, because the specified DRM System in `DRMSystemId` is unknown.

`function` **`onDRMSystemStatusChange`**`( String DRMSystemID )`

The function that is called when the status of a DRM system changes.

The specified function is called with one argument `DRMSystemID` which is defined as follows:

- `String DRMSystemID` – argument that specifies the DRM System ID of the DRM system that generated the event as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2].

7.6.1.2 Methods

`String` **`sendDRMMessage`**`(String msgType, String msg, String DRMSystemID)`

Description	Send message to the DRM agent, using a message type as defined by the DRM system. Returns a unique ID to identify the message, to be passed as the '`msgID`' argument for the callback function registered through `onDRMMessageResult`. This is an asynchronous method. Applications will be notified of the results of the operation

	via events dispatched to `onDRMMessageResult` and corresponding DOM level 2 events.	
Arguments	*msgType*	A globally unique message type as defined by the DRM system, for example: `application/vnd.marlin.drm.actiontoken+xml` (i.e. MIME type of Marlin Action Token). Valid values for the msgType parameter include the MIME types described in Annex C of[OIPF_CSP2].
	msg	The message to be provided to the underlying DRM agent formatted according to the message type as indicated by attribute `msgType`. Valid format for the `msg` parameter are message formats described in Annex C of [OIPF_CSP2].
	DRMSystemID	DRMSystemID as defined by element DRMSystemID in Table 9 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is "`urn:dvb:casystemid:19188`". In the case that parameter `msgType` indicates a CSPG-CI+ message as described in Section 4.2.3.4.1.1.2 of [OIPF_CSP2], the `DRMSystemID` parameter SHALL be specified. Otherwise, the value may be `null`.

`Integer` **`DRMSystemStatus`**`( String DRMSystemID )`

Returns the status of the indicated DRM system.

The specified function is called with one argument `DRMSystemID`, which is defined as follows:

- `String DRMSystemID` –argument that specifies the DRM System ID of the DRM system that is being queried as defined by the element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is "urn:dvb:casystemid:19188".

The value returned shall be as defined below:

Value	Description	Semantics
0	READY	The DRM system is fully initialised and ready.
1	UNKNOWN	Unknown DRM system.
2	INITIALISING	The DRM system is initialising and not ready to start communicating with the application.
3	ERROR	There is a problem with the DRM system. It may be possible to communicate with it to obtain more information.

7.6.1.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onDRMMessageResult`	`DRMMessageResult`	Bubbles: No Cancelable: No Context Info: `msgID`, `resultMsg`, `resultCode`
`onDRMSystemStatusChange`	`DRMSystemStatusChange`	▪ Bubbles: No ▪ Cancelable: No ▪ Context Info: `DRMSystemID`

NOTE: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. The `addEventListener()` method SHOULD be called on the `application/oipfDrmAgent` object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.7 Gateway Discovery and Control APIs

The `application/oipfGatewayInfo` object SHALL provide the information of the gateway and subsequently interact with the gateway (e.g. IMS Gateway, Application Gateway, CSPG CI+ Gateway and CSPG-DTCP Gateway) as defined in Section 4.2. The OITF SHALL support the gateway discovery and control though the use of the following non-visual embedded object:

```
<object id="gatewayinfo" type="application/oipfGatewayInfo" />
```

Access to the functionality of the `application/oipfGatewayInfo` embedded object is privileged and SHALL adhere to the security requirements defined in Section 10.1.

7.7.1 The application/oipfGatewayInfo embedded object

7.7.1.1 Properties

`readonly Boolean` **`isIGSupported`**
Indicates whether an IMS Gateway is supported or not.

`readonly Boolean` **`isAGSupported`**
Indicates whether an Application Gateway is supported or not.

readonly Boolean **isCSPGCIPlusSupported**
Indicates whether an CSPG-CI+ Gateway is supported or not.

readonly Boolean **isCSPGDTCPSupported**
Indicates whether an CSPG-DTCP Gateway is supported or not.

readonly Boolean **isIGDiscovered**
Indicates whether an IMS Gateway is discovered or not. Note: This property was formerly referred to as **IGDiscovery**.

readonly Boolean **isAGDiscovered**
Indicates whether an Application Gateway is discovered or not. Note: This property was formerly referred to as **AGDiscovery**.

readonly Boolean **isCSPGCIPlusDiscovered**
Indicates whether an CSPG-CI+ Gateway is discovered or not.

readonly Boolean **isCSPGDTCPDiscovered**
Indicates whether an CSPG-DTCP Gateway is discovered or not. Note: This property was formerly referred to as **cspGatewayDiscovery**. The former **cspGatewayDiscovery** property is now replaced with **isCSPGCIPlusDiscovered** for CSPG-CI+ case and **isCSPGDTCPDiscovered** for CSPG-DTCP case.

readonly String **igURL**
The URL of the IMS Gateway.

readonly String **agURL**
The URL of the Application Gateway.

readonly String **cspgDTCPURL**
The URL of the CSPG DTCP gateway.

Note: This property was formerly referred to as **cspGatewayURL** which was relevant for CSPG-DTCP case only.

`Integer` **interval**

The periodic interval time (in seconds) to discover the gateways. When the interval property is set, a UPnP Discovery mechanism is executed.

`readonly StringCollection` **CSPGCIPlusDRMType**

Indicates the list of CA Systems supported by the CSPG-CI+ Gateway under the form of URN with the DVB CASystemID (16 bit number) in there. Each element of CSPGCIPlusDRMType shall be signalled by prefixing the decimal number format of CA_System_ID with "`urn:dvb:casystemid:`".

`function` **onDiscoverIG**()

The function that SHALL be called when an IMS Gateway is discovered or lost by the OITF which uses a UPnP Discovery mechanism described in [OIPF_PROT2] Section 10.1.1.1. The actual status of the gateway (discovered or not) can be determined by reading the isIGDiscovered property.

The specified function is called with no arguments.

`function` **onDiscoverAG**()

The function that SHALL be called when an Application Gateway is discovered or lost by the OITF which uses a UPnP Discovery mechanism described in [OIPF_PROT2] Section 10.1.1.2. The actual status of the gateway (discovered or not) can be determined by reading the isAGDiscovered property.

The specified function is called with no arguments.

`function` **onDiscoverCSPGCIPlus**()

The function that SHALL be called when a CSPG-CI+ Gateway is discovered or lost by the OITF (including any change to the DRM systems supported by that gateway). The CSPG-CI+ Gateway SHALL be discovered as defined in [OIPF_CSP2]. The actual status of the gateway (discovered or not) can be determined by reading the isCSPCIPlusGDiscovered property.

The specified function is called with no arguments.

`function` **onDiscoverCSPGDTCP**()

The function that SHALL be called when a CSPG-DTCP Gateway is discovered or lost by the OITFwhich uses a UPnP Discovery mechanism described in [OIPF_PROT2] Section 10.1.1.3. The actual status of the gateway (discovered or not) can be determined by reading the isCSPGDTCPDiscovered property.

The specified function is called with no arguments.

Note: This property was formerly referred to as **onDiscoverCSPG**. The former **onDiscoverCSPG**

property is now replaced with **onDiscoverCSPGCIPlus** for CSPG-CI+ case and **onDiscoverCSPGDTCP** for CSPG-DTCP case.

7.7.1.2 Methods

Boolean **isIGSupportedMethod**(String MethodName)	
Description	Shall return `true` when the IG supports the method named `MethodName`. If the function returns `false`, it indicates that IG does not support the specified method.

7.7.1.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 even t	DOM 2 Event properties
onDiscoverIG	DiscoverIG	Bubbles: No Cancelable: No
onDiscoverAG	DiscoverAG	Bubbles: No Cancelable: No
onDiscoverCSPGCIPlus	DiscoverCSPGCIPlus	Bubbles: No Cancelable: No
onDiscoverCSPGDTCP	DiscoverCSPGDTCP	Bubbles: No Cancelable: No

NOTE: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a `DiscoverIG`, `DiscoverAG`, `DiscoverCSPGCIPlus` and `DiscoverCSPGDTCP` event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `application/oipfGatewayInfo` object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.8 IMS Related APIs

If an OITF has indicated support for the control of its IMS functionality by a server by stating `<ims>true</ims>` as defined in Section 9.3.9 in its capability description, the OITF SHALL support IMS through the use of the following non-visual object:

```
<object type="application/oipfIMS"/>
```

The IMS API provides the necessary javascript methods to register new users in the IMS network. It also provides methods to register users (`registerUser`), along with the supported feature tags, IMS Communication Service Identifier (ICSI) and IMS Application Reference Identifier (IARI), and de-register users (`deRegisterUser`). A method `getRegisteredUsers` is also defined to view all the registered users. A method `getAllUsers` retrieves all users provisioned in the IG. Once registered it is possible to switch users for using IMS services by using method `setUser`.

A property is defined to view the current user to be used for a service (`currentUser`).

In order to handle the out-of-session IMS notifications, namely, the new dialogues, there is a method for subscribing to these events (`subscribeIMSNotification`). All new dialogues are communicated through a callback function (`onIMSNotification`) to the application instance performing the subscription.

The IMS APIs apply only to privileged applications and SHALL adhere to the security model as defined in Section 10.

7.8.1 The application/oipfIMS embedded object

7.8.1.1 Constants

The following constants are defined as properties on the `application/oipfIMS` embedded object:

Name	Value	Use
`STATE_REGISTERED`	0	Specifies that the user has been successfully registered (not subscribed to registration event). This also represents the state when the registration event subscription has been terminated for some reason by network.
`STATE_REGISTERED_SUBSCRIPTION_PENDING`	1	Indicates that user is registered successfully but the subscription-state for the registration event indicates a status of "pending".
`STATE_REGISTERED_SUBSCRIPTION_ACTIVE`	2	Specifies that the user has been successfully registered and subscribed to registration event (i.e. subscription-state for registration event indicates a status of "active").
`STATE_DEREGISTERED`	3	Specifies that the user has been successfully deregistered. This can be result of network initiated/locally initiated deregistration request.
`STATE_FAILURE`	4	Represents a failure condition.

7.8.1.2 Properties

`function onIMSNotification( String responseHeaders, String msgText, Document msgXML )`
This function is called on the application which called `subscribeIMSNotification` when an unsolicited IMS notification arrives. The application will be notified of all IMS notifications corresponding to any of the subscribed-to feature tags regardless of which application subscribed to it. The specified function is called with 3 arguments. • `String responseHeaders` – The concatenated list of all HTTP headers, as a single string, with each header line separated by a U+000D (CR) U+000A (LF) pair excluding the status line. In

absence of HNI-IGI interface, the `responseHeaders` will be a concatenated list all SIP headers, as a single string, with each header line separated by a U+000D (CR) U+000A (LF) pair excluding the status line.

- `String msgText` – the response entity body as a string, as defined in [XHR].
- `Document msgXML` – the response entity body as a Document, as defined in [XHR].

`function` **`onNotificationResult`**`( Integer resultMsg )`

This function is called with return result from the `subscribeIMSNotification` method.

This function is not invoked in the case when there is no re-registration as part of `subscribeIMSNotification`.

The specified function is called with a single argument – `resultMsg`.

- `Integer resultMsg` – result message from performing `subscribeIMSNotification` method.

Result message	Description	Semantics
0	Successful	The action performed by the underlying functionality was successful.
1	Unknown error	The action performed by the underlying functionality failed because an unspecified error occurred.
2	Wrong user credentials	The user credentials was not accepted by the server.
3	The user doesn't exist.	The user id doesn't exist in the local user table.

`function` **`onRegistrationContextUpdate`**`( String user, Integer state, Integer errorCode )`

This function is called with return result from the methods `registerUser` and `deRegisterUser`. In addition, the function is also called whenever there is an update to the registration status of specified user.

The specified function is called with 3 arguments – `user`, `state` and `errorCode`.

- `String user` – The IMPU of the user.
- `Integer state` – The current state of the registration as indicated using the constant values defined in 7.8.1.1.
- `Integer errorCode` – In case of STATE_FAILED state, provides more information on reason for failure.

errorCode	Description	Semantics

1	Unknown error	The action performed by the underlying functionality failed because an unspecified error occurred.
2	Wrong user credentials	The user credentials were not accepted by the server. The DAE may request from the user a new PIN which can then be used to call the `registerUser()` method.
3	The user doesn't exist.	The user id doesn't exist in the local user table.

`readonly UserData` **`currentUser`**
The current user property represents the public user identity which is being used or shall be used for HNI-IGI communication. If not set then the default user shall be used or indicated. It shall be set to the default user if a user has not been explicitly set using the `setUser()` method.

7.8.1.3 Methods

`UserDataCollection` **`getRegisteredUsers`**`()`	
Description	Return all the users that are currently registered with the IG.

`Void` **`registerUser`**`( String userId, String pin )`		
Description	This method performs user registration to the IMS network. Results from this method is sent to the callback method `onRegistrationContextUpdate.`	
Arguments	*userId*	The user identifier represents the public user identity or IMPU.
	pin	The pin is optional and carries the password to be used towards the IG in case of HTTP Digest over HNI-IGI interface or SIP Digest if there is no HNI-IGI. If pin is not specified then the default user password shall be used. The pin used for digest authentication is limited to the HNI-IGI interface with the IG and SHALL NOT impact the HTTP Digest requests from within the DAE application. Note that for non-native HNI-IGI support is not applicable.

`void` **`deRegisterUser`**`( String userId )`		
Description	The indicated user is de-registered from IMS. Any sessions that may be open are closed. De-registration of default user has no effect nor de-registration of any users registered from a native application in the OITF. Results from this method is sent to the callback method `onRegistrationContextUpdate.`	
Arguments	*userId*	The user identifier represents the public user identity or IMPU.

UserDataCollection **getAllUsers**()	
Description	Return all the users that are currently provisioned in the IG. The first entry in the collection is the default user. The users are retrieved according to [OIPF_PROT2] Section 5.3.6.3.

Boolean **setUser**(String userId)		
Description	When invoked, any ongoing sessions for the current user shall be closed. If setUser is unsuccessful due to user not being registered, it is necessary to first register the user and wait for a successful response to the onRegistrationContextUpdate callback function. If the user gets deregistered (either by the local application or by the network), any ongoing sessions for the user shall be closed .The default user shall be automatically assumed for all IMS services until overridden again by setUser method.	
Argument	*userId*	The user identifier represents the public user identity or IMPU.

void **subscribeIMSNotification**(FeatureTagCollection featureTagCollection, Boolean performUserRegistration)		
Description	This method subscribes for new IMS out-of-session dialogues for the indicated application for the currently active user. The notification shall be notified using onIMSNotification callback method. If the application that made the subscription closes then there is an automatic un-subscription to new notifications. Otherwise it is possible to perform unsubscribeIMSNotification. Any new dialogues shall be notified over the callback method onIMSNotification.	
Arguments	*featureTagCollection*	The FeatureTagCollection object of the DAE application. The inclusion of featureTag values other than null indicates which dualogues are reported. The dialogues that match the featureTag are to be reported. The inclusion of null indicates that all dialogues are to be reported.
	performUserRegistration	If this is true a new user registration is required. SHOULD be set to false if it is know that other applications will be registered shortly This parameter is ignored in the case when the filtering of IMS notifications is done locally. In this case, the initial registration for active user will include all feature tags.

void **unsubscribeIMSNotification**()	
Description	The DAE application calling this method will be de-registered for IMS notifications. Associated feature tag(s) for the DAE application are removed from the featureTagCollection object for the user. A re-registration will be performed for the

	corresponding user if IMS notifications are not locally filtered. Results from this method is sent to the callback method `onNotificationResult`.

7.8.1.4 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onNotificationResult`	`NotificationResult`	Bubbles: No Cancelable: No Context Info: `resultMsg`
`onIMSNotification`	`IMSNotification`	Bubbles: No Cancelable: No Context Info: `callId`, `contact`, `from`, `to`
`onRegistrationContextUpdate`	`RegistrationContextUpdate`	Bubbles: No Cancelable: No Context Info: `user`, `state`, `errorCode`

7.8.2 Extensions to application/oipfIMS for communication services

If a client has indicated support for the control of its Communication Services functionality by a server by stating `<communication_services>true</communication_services>` as defined in Section 9.3.9 in its capability description, the client SHALL support IMS through the use of the following non-visual embedded object:

```
<object type="application/oipfIMS"/>
```

The Communication Services API provides for instant messaging, presence and contact list services. The messages can either be in a chat session using MSRP (see [OIPF_PROT2]) or page mode messages sent without a session. The support of Communication Services SHALL follow the OMA specification [PRES], [IM].

The Communication Services API SHALL be supported in combined OITF and IG deployment cases. It MAY be supported in other deployment cases. The use of the HNI-IGI interface is OPTIONAL between the OITF and IG when these are co-deployed.

7.8.2.1 Properties

`function` **`onIncomingMessage`**`( String fromURI, String msg, Integer cid )`
The function that is called when an incoming chat message is received for the active user.

The specified function is called with 3 arguments:

- `String fromURI` – The sender address of the message.
- `String msg` – The text message sent by the remote peer.
- `Integer cid` – chat session identifier. This may be an identifier returned by a call to the `openSession()` method if the session was initiated by the current application, a different value if the session was started by a remote peer. If the message is to be sent outside a chat session, the value of this argument SHALL be `undefined`.

`function` **onContactStatusChange**`( String remoteURI, Integer state )`
This function is called when status has changed for a contact in the contact list or a user used with the method subscribeToStatus. The specified function is called with 2 arguments: • `String remoteURI` – The user address for which the status has changed. • `Integer state` – Set to 1 if the user is present, and 0 if not. Other values may be defined in the future.

`function` **onNewWatcher**`( String remoteURI )`
This function is called when a remote URI is requesting watcher authorization of the local user's presentity. The specified function is called with one argument: • `String remoteURI` – The remote user address which requested watcher authorization.

7.8.2.2 Methods

`Integer` **openChatSession**`( String toURI )`		
Description	Opens a chat session with a remote user. Returns an integer identifier for the chat session to be used when a message is sent in the chat session or to match when incoming message is received.	
Arguments	*toURI*	The address of the remote chat user.

`void` **sendMessageInSession**`( Integer cid, String msg )`	
Description	Sends a new text message in a chat session. The chat can either be started by the user by calling the method `openChatSession()` or can be a session received in the `onIncomingMessage` callback function.

Arguments	*cid*	The chat session identifier.
	msg	Text message to send.

`void` **`closeChatSession`**`( Integer cid )`		
Description	Closes a chat session.	
Arguments	*cid*	The chat session identifier.

`void` **`sendMessage`**`( String toURI, String msg )`		
Description	Sends a new text message to a remote peer without starting a session.	
Arguments	*toURI*	The address of the remote chat user.
	msg	Text message to send.

`void` **`setStatus`**`( Integer state )`		
Description	Sets the presence state of the local user.	
Arguments	*state*	Set to 1 if the user is present, and 0 if not. Other values may be defined in the future.

`void` **`subscribeToStatus`**`( String remoteURI )`		
Description	Subscribe to status for a remote user.	
Arguments	*remoteURI*	The address of the remote user.

`ContactCollection` **`getContacts`**`()`
Description	Get the users contact list.

`void` **`allowContact`**`( String remoteURI )`		
Description	Allows the watcher authorization to subscribe to the local user's presentity.	
Arguments	*remoteURI*	The address of the remote user.

`void` **`blockContact`**`( String remoteURI )`		
Description	Blocks the watcher authorization to subscribe to the local user's presentity.	
Arguments	*remoteURI*	The address of the remote user.

Boolean **createContactList**(String contactListUri, ContactCollection contacts)		
Description	Creates a contact list	
Arguments	*contactListUri*	The public user identity or IMPU of the contact list.
	contacts	The collection of contact objects representing the members of the list.

ContactCollection **getContacts**(String contactListUri)		
Description	Get the users in the specified contact list	
Arguments	*contactListUri*	The public user identity or IMPU of the contact list.

Boolean **addToContactList**(String contactListUri, Contact member)		
Description	Updates the specified contact list by adding a new member to that list	
Arguments	*contactListUri*	The public user identity or IMPU of the contact list to be updated.
	member	The new contact to be added to the list.

Boolean **removeFromContactList**(String contactListUri, Contact member)		
Description	Updates the specified contact list by removing specified member from that list	
Arguments	*contactListUri*	The public user identity or IMPU of the contact list to be updated.
	member	The new contact to be removed from the list

Boolean **deleteContactList**(String contactListUri)		
Description	Deletes the specified contact list	
Arguments	*contactListUri*	The public user identity or IMPU of the contact list to be deleted

void **allowAllContacts**(String domain)		
Description	Allows all watchers belonging to specified domain authorization to subscribe to local user's presentity. If `null`, then all contacts will be allowed.	
Arguments	*domain*	Watchers belonging to this domain are authorized to subscribe. If `null`, then all watchers are authorized to subscribe irrespective of domain.

void **blockAllContacts**(String domain)		
Description	Blocks all watchers belonging to specified domain from subscribing to local user's presentity. If null, then all contacts will be blocked.	
Arguments	*domain*	Watchers belonging to this domain are denied authorization to subscribe. If null, then all watchers are blocked from subscribing irrespective of domain.

7.8.2.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onIncomingMessage	IncomingMessage	Bubbles: No Cancelable: No Context Info: fromURI,msg,cid
onContactStatusChange	ContactStatusChange	Bubbles: No Cancelable: No Context Info: remoteURI,present
onNewWatcher	NewWatcher	Bubbles: No Cancelable: No Context Info: remoteURI

7.8.3 The UserData class

7.8.3.1 Properties

readonly String **userId**
The user identifier represents the public user identity or IMPU.

readonly FeatureTagCollection **featureTags**
The FeatureTag data is optional (may have a value of null) and carries a collection of feature tag objects associated to an application. For example the feature tag may be an ICSI or IARI or a feature tag identifying the service for. an incoming instant messages. The object includes feature tags related

to both DAE and native applications in OITF.

readonly String **friendlyName**
The friendly name for the user. Used as display name in outgoing messages.

7.8.4 The UserDataCollection class

```
typedef Collection<UserData> UserDataCollection
```

The `UserDataCollection` class represents a collection of `UserData` objects. See annex K for the definition of the collection template.

7.8.5 The FeatureTag class

7.8.5.1 Properties

readonly String **featureTag**
A string containing a featureTag value associated to an application. The featureTag value may have a value of `null` when used with the `subscribeImsNotification()` method on the `application/oipfIMS` object. This indicates that all dialogues are reported. The feature tag SHALL populate the X-OITF- headers as specified in [TISPAN] Section 5.6.2, [IM], [3GPP TS 24.229], [RFC3840] and [RFC3841].

7.8.6 The FeatureTagCollection class

```
typedef Collection<FeatureTag> FeatureTagCollection
```

The `FeatureTagCollection` class represents a collection of `FeatureTag` objects. See annex K for the definition of the collection template.

7.8.7 The Contact class

7.8.7.1 Properties

String **contactId**
The contact identifier represents the public user identity or IMPU used in communication with the contact.

String **friendlyName**
The friendly name for the user. Used as display name in outgoing messages.

7.8.8 The ContactCollection class

```
typedef Collection<Contact> ContactCollection
```

The `ContactCollection` class represents a collection of `Contact` objects. See annex K for the definition of the collection template.

In addition to the methods and properties defined for generic collections, the `ContactCollection` class supports the additional methods defined below.

7.8.8.1 Methods

`Boolean remove( String contactId )`		
Description	Removes the contact represented by `contactId` from the users IMS contact list. Returns `true` on success.	
Arguments	*contactId*	Contact identifier of the user in the IMS contact list.

`Boolean add( Contact contact )`		
Description	Adds the contact represented by the `Contact` object to the users IMS contact list. Returns `true` on success.	
Arguments	*contact*	Contact object to be added from users IMS contact list.

7.9 Parental rating and parental control APIs

This section defines APIs related to parental ratings and parental control.

Sections 7.9.1 through 7.9.3 define a new Javascript embedded object `application/oipfParentalControlManager` and the related `ParentalRatingScheme` and `ParentalRatingSchemeCollection` objects, which allows applications to construct a new parental rating scheme (and a parental rating value using that scheme), and to temporarily enable or disable viewing of a content item. These APIs SHALL be supported if an OITF supports parental controls as indicated by value "`true`" for element `<parentalcontrol>` (as defined by Section 9.3.5) in its capability profile.

Sections 7.9.4 and 7.9.5 define the `ParentalRating` and `ParentalRatingCollection` objects. These objects are used/referenced by various other objects, such as the `Programme` object as defined in Section 7.16.2 to indicate a particular parental rating. The support for these objects depends on the support for the sections in which these are used.

7.9.1 The application/oipfParentalControlManager embedded object

If an OITF supports parental controls as indicated by value "`true`" for element `<parentalcontrol>` (as defined by Section 9.3.5) in its capability profile, the OITF SHALL support the `application/oipfParentalControlManager` object with the following interface

The following example shows a possible usage scenario for the `application/oipfParentalControlManager`, i.e. to add a new parental rating scheme to the `parentalRatingSchemes` collection:

```
//get a reference to the parental control manager object
var pcManager = document.getElementById("pcmanager");
```

```
// add a new rating scheme – in this case, the MPAA rating scheme
pcManager.parentalRatingSchemes.addParentalRatingScheme(
    "urn:mpeg:mpeg7:cs:MPAAParentalRatingCS:2001", "G,PG,PG-13,R,NC-17,NR" );
```

The following example shows a possible usage scenario for the application/oipfParentalControlManager, i.e. to temporarily unblock a blocked content item (e.g. after asking the user to enter the parental control pin):

```
// If a content item is blocked, the event "onParentalRatingChange" can be captured,
// and the setParentalControlStatus() method can be used to temporarily unblock the
// content (e.g. after asking the user to enter the parental control pin)

function askForPin() { … }

...

//get a reference to the A/V player object
var avPlayer = document.getElementById("avPlayer");

avPlayer.onParentalRatingChange = function() {var
pin=askForPin();pcManager.setParentalControlStatus(pin, false)};
```

7.9.1.1 Properties

readonly ParentalRatingSchemeCollection **parentalRatingSchemes**
A reference to the collection of rating schemes known by the OITF.

readonly Boolean **isPINEntryLocked**
The lockout status of the parental control PIN. If the incorrect PIN has been entered too many times in the configured timeout period, parental control PIN entry SHALL be locked out for period of time determined by the OITF.

7.9.1.2 Methods

Integer **setParentalControlStatus**(String pcPIN, Boolean enable)	
Description	As defined in [OIPF_CSP2], the OITF shall prevent the consumption of a programme when its parental rating doesn't meet the parental rating criterion currently defined in the OITF. Calling this method with `enable` set to `false` will temporarily allow the consumption of any blocked programme. Setting the parental control status using this method SHALL set the status until the consumption of any of all the blocked programmes terminates (e.g. until the content item being played is changed), or another call to the `setParentalControlStatus()` method is made. Setting the parental control status using this method has the following effect :for the `Programme` and `Channel` objects as defined in Sections 7.16.2 and 7.13.12, the `blocked` property of a programme or channel SHALL be set to `true` for programmes whose parental rating does not meet the applicable parental rating criterion, but the `locked` property SHALL be set to `false`. This operation to temporarily disable parental rating control SHALL be protected by the parental control PIN (i.e. through the `pcPIN` argument). The return value indicates the success of the operation, and SHALL take one of the following values:

	Value	**Description**
	0	The PIN is correct.
	1	The PIN is incorrect.
	2	PIN entry is locked because an invalid PIN has been entered too many times. The number of invalid PIN attempts before PIN entry is locked is outside the scope of this specification.
Arguments	*pcPIN*	The parental control PIN.
	enable	Flag indicating whether parental control should be enabled.

`Boolean` **`getParentalControlStatus`**`()`	
Description	Returns a flag indicating the temporary parental control status set by `setParentalControlStatus()`. Note that the returned status covers parental control functionality related to all rating schemes.

`Boolean` **`getBlockUnrated`**`()`	
Description	Returns a flag indicating whether or not the OITF has been configured by the user to block content for which a parental rating is absent.

`Integer` **`setParentalControlPIN`**`( String oldPcPIN, String newPcPIN )`		
Description	Set the parental control PIN. This operation SHALL be protected by the parental control PIN (if PIN entry is enabled). The return value indicates the success of the operation, and SHALL take one of the following values:	
	Value	**Description**
	0	The PIN is correct.
	1	The PIN is incorrect.
	2	PIN entry is locked because an invalid PIN has been entered too many times. The number of invalid PIN attempts before PIN entry is locked is outside the scope of this specification.
Arguments	*oldPcPIN*	The current parental control PIN.
	newPcPIN	The new value for the parental control PIN.

`Integer` **`unlockWithParentalControlPIN`**`( String pcPIN, Object target, Integer`

<table>
<tr><td colspan="3">duration)</td></tr>
<tr><td>Description</td><td colspan="2">Unlock the object specified by target for viewing if pcPIN contains the correct parental control PIN.

This operation SHALL be protected by the parental control PIN (if PIN entry is enabled). The return value indicates the success of the operation, and SHALL take one of the following values:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>The PIN is correct.</td></tr>
<tr><td>1</td><td>The PIN is incorrect.</td></tr>
<tr><td>2</td><td>PIN entry is locked because an invalid PIN has been entered too many times. The number of invalid PIN attempts before PIN entry is locked is outside the scope of this specification.</td></tr>
</table>
</td></tr>
<tr><td rowspan="3">Arguments</td><td>pcPIN</td><td>The parental control PIN.</td></tr>
<tr><td>target</td><td>The channel or programme to be unlocked.</td></tr>
<tr><td>duration</td><td>The length of time (in seconds) for which the item SHALL be unlocked.</td></tr>
</table>

<table>
<tr><td colspan="3">Integer verifyParentalControlPIN(String pcPIN)</td></tr>
<tr><td>Description</td><td colspan="2">Verify that the PIN specified by pcPIN is the correct parental control PIN.

This method will return one of the following values:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>The PIN is correct.</td></tr>
<tr><td>1</td><td>The PIN is incorrect.</td></tr>
<tr><td>2</td><td>PIN entry is locked because an invalid PIN has been entered too many times. The number of invalid PIN attempts before PIN entry is locked is outside the scope of this specification.</td></tr>
</table>
</td></tr>
<tr><td>Arguments</td><td>pcPIN</td><td>The parental control PIN to be verified.</td></tr>
</table>

<table>
<tr><td colspan="2">Integer setBlockUnrated(String pcPIN, Boolean block)</td></tr>
<tr><td>Description</td><td>Set whether programmes for which no parental rating has been retrieved from the metadata client nor defined by the service provider should be blocked automatically by the terminal.</td></tr>
</table>

<table>
<tr><td></td><td colspan="2">This operation SHALL be protected by the parental control PIN (if PIN entry is enabled). The return value indicates the success of the operation, and SHALL take one of the following values:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>The PIN is correct.</td></tr>
<tr><td>1</td><td>The PIN is incorrect.</td></tr>
<tr><td>2</td><td>PIN entry is locked because an invalid PIN has been entered too many times. The number of invalid PIN attempts before PIN entry is locked is outside the scope of this specification.</td></tr>
</table></td></tr>
<tr><td rowspan="2">Arguments</td><td>pcPIN</td><td>The parental control PIN.</td></tr>
<tr><td>block</td><td>Flag indicating whether programmes SHALL be blocked.</td></tr>
</table>

7.9.2 The ParentalRatingScheme class

```
typedef Collection<String> ParentalRatingScheme
```

A `ParentalRatingScheme` describes a single parental rating scheme that may be in use for rating content, e.g. the MPAA or BBFC rating schemes. It is a collection of strings representing rating values, which next to the properties and methods defined below SHALL support the array notation to access the rating values in this collection. For the natively OITF supported parental rating systems the values SHALL be ordered by the OITF to allow the rating values to be compared in the manner as defined for property threshold for the respective parental rating system. Using a threshold as defined in this API may not necessarily be the proper way in which parental rating filtering is applied on the OITF, e.g. the US FCC requirements take precedence for device to be imported to the US.

The parental rating schemes supported by a receiver MAY vary between deployments.

See annex K for the definition of the collection template. In addition to the methods and properties defined for generic collections, the ParentalRatingScheme class supports the additional properties and methods defined below.

7.9.2.1 Properties

`readonly String` **`name`**
The unique name that identifies the parental rating scheme. Valid strings include: • the URI of one of the MPEG-7 classification schemes representing a parental rating scheme as defined by the `uri` attribute of one of the parental rating `<ClassificationScheme>` elements in [MPEG-7]. • the string value "`urn:oipf:GermanyFSKCS`" to represent the GermanyFSK rating scheme as defined in [META]. • the string value "`dvb-si`": this means that the scheme of a minimum recommended age encoded as per [EN 300 468], is used to represent the parental rating values.

If the value of `name` is "`dvb-si`", the `ParentalRatingScheme` remains empty (i.e. `ParentalRatingScheme.length == 0`).

`readonly ParentalRating` **`threshold`**
The parental rating threshold that is currently in use by the OITF's parental control system for this rating scheme, which is encoded as a `ParentalRating` object in the following manner: If the value of the `name` property of the `ParentalRatingScheme` object is unequal to "`dvb-si`", then: • the `value` property of the threshold object represents the value for which items with a `ParentalRating.value` greater or equal to the `value` property of the threshold object have been configured by the OITF's parental control subsystem to be blocked. • the `labels` property of the threshold object represents the bit map of zero or more flags for which items with a `ParentalRating.labels` property with any of the same flags set have been configured by the OITF's parental control subsystem to be blocked. If the value of the `name` property of the `ParentalRatingScheme` object is "`dvb-si`", the threshold indicates a minimum recommended age encoded as per [EN 300 468] at which or above which the content is being blocked by the OITF's parental control subsystem Note that the `value` property as an index into the `ParentalRating` object that defines the threshold can be 1 larger than the value of `ParentalRatingScheme.length` to convey that no content is being blocked by the parental control subsystem.

7.9.2.2 Methods

`Integer` **`indexOf`**`( String ratingValue )`		
Description	Return the index of the rating represented by attribute `ratingValue` inside the parental rating scheme string collection, or `-1` if the rating value cannot be found in the collection.	
Arguments	*ratingValue*	The case-insensitive string representation of a parental rating value. See property `name` in Section 7.9.1.1 for more information about possible values.

`String` **`iconUri`**`( Integer index )`		
Description	Return the URI of the icon representing the rating at `index` in the rating scheme, or `undefined` if no item is present at that position. If no icon is available, this method SHALL return `null`.	
Arguments	*index*	The index of the parental rating scheme.

7.9.3 The ParentalRatingSchemeCollection class

```
typedef Collection<ParentalRatingScheme> ParentalRatingSchemeCollection
```

A ParentalRatingSchemeCollection represents a collection of parental rating schemes, e.g. as returned by property parentalRatingSchemes of the "application/oipfParentalControlManager" object as defined in Section 7.9.1. Next to the properties and methods defined below a ParentalRatingSchemeCollection object SHALL support the array notation to access the parental rating scheme objects in this collection.

See annex K for the definition of the collection template. In addition to the methods and properties defined for generic collections, the ParentalRatingSchemeCollection class supports the additional properties and methods defined below.

7.9.3.1 Methods

ParentalRatingScheme **addParentalRatingScheme**(String name, String values)		
Description	Create a new ParentalRatingScheme object and adds it to the ParentalRatingSchemeCollection. Applications MAY use this method to register additional parental rating schemes with the platform. When registered, the new parental rating scheme SHALL (temporarily) be accessible through the parentalRatingSchemes property of the "application/oipfParentalControlmanager" object as defined in Section 7.9.1. The application SHALL make sure that the values are ordered in such a way to allow the rating values to be compared in the manner as defined for the threshold property for the respective parental rating system. This method returns a reference to the ParentalRatingScheme object representing the added scheme. If the value of the name parameter corresponds to an already-registered rating scheme, this method returns a reference to the existing ParentalRatingScheme object. If the newly defined rating scheme was not known to the OITF, the scheme MAY be stored persistently, and the OITF may offer a UI to set the parental rating blocking criteria for the newly added parental rating scheme. If the OITF has successfully stored (persistently or not persistently) the additional parental rating scheme, the method SHALL return a non-null ParentalRatingScheme object.	
Arguments	*name*	A unique string identifying the parental rating scheme to which this value refers. See property name in Section 7.9.1.1 for more information about possible values.
	values	A comma-separated list of the possible values in the rating scheme, in ascending order of severity. In case the rating scheme is one of the [MPEG-7] rating classification schemes, this means that the list of parental rating values contains the values as specified by the <Name> elements of the <Term> elements in the order of appearance as they are defined for the classification scheme, with the exception of the Internet Content Rating Association (ICRA) based ratings, for which the reverse order has to be applied. The values must be ordered in such a way to allow the rating values to be compared in the manner as defined for property threshold for the respective parental rating system.

ParentalRatingScheme **getParentalRatingScheme**(String name)	
Description	This method returns a reference to the ParentalRatingScheme object that is associated with the given scheme as specified through parameter name. If the value of name does not corresponds to the name property of any of the ParentalRatingScheme objects in the ParentalRatingSchemeCollection, the

	method SHALL return `undefined`.	
Arguments	*name*	The unique name identifying a parental rating scheme.

7.9.4 The ParentalRating class

A `ParentalRating` object describes a parental rating value for a programme or channel. The `ParentalRating` object identifies both the rating scheme in use, and the parental rating value within that scheme.

In case of a BCG the values of the properties in this object will be read from the `ParentalGuidance` element that is the child of a programme's BCG description.

Example usage:

```
<!-- This example shows a possible usage scenario for the ParentalRating
     datastructure, i.e. to create a new programme to record and set
     parental rating to MPAA parental rating to PG-13.
-->
...
<script type="text/javascript" language="Javascript1.5">

// get a reference to the recorder object
var recorder = document.getElementById("recorder");

// create new programme to record
var myProgramme = recorder.createProgrammeObject();

// add a new parental rating value to myProgramme, in this case the
// programme is rated PG-13 for the US using the MPAA Parental rating scheme.
myProgramme.parentalRatings.addParentalRating(
  "urn:mpeg:mpeg7:cs:MPAAParentalRatingCS:2001", ”PG-13”, 2, 0, “US”
  );

</script>
...
<object id="recorder" type="application/oipfRecordingScheduler"/>
```

7.9.4.1 Properties

`readonly String` **name**
The case-insensitive string representation of the parental rating value for the respective rating scheme denoted by property `scheme`. Valid strings include: • if the value of property `scheme` represents one of the parental rating classification scheme names identified by [MPEG-7]: the string representation of one of the parental rating values as defined by one of the `<Name>` elements. • if the value of property `scheme` is ”`urn:oipf:GermanyFSKCS`” , the string representation of one the values for the GermanyFSK rating scheme as defined in [OIPF_META2]. • if the value of property `scheme` is equal to “`dvb-si`”, the string representation of the minimum recommended age encoded as per [EN 300 468], which corresponds to rating_type 0 in [IEC62455]. An example of a valid parental rating value is “PG-13”.

`readonly String` **scheme**

Unique case-insensitive name identifying the parental rating guidance scheme to which this parental rating value refers. Valid strings include:

- the URI of one of the MPEG-7 classification schemes representing a parental rating scheme as defined by the `uri` attribute of one of the parental rating `<ClassificationScheme>` elements in [MPEG-7]
- the string value "`urn:oipf:GermanyFSKCS`" to represent the GermanyFSK rating scheme as defined in [OIPF_META2].
- the string value "`dvb-si`": this means that the scheme of a minimum recommended age encoded as per [EN 300 468], is used to represent the parental rating values.

`readonly Integer` **value**

The parental rating value represented as an index into the set of values defined as part of the `ParentalRatingScheme` identified through property `scheme`.

If an associated `ParentalRatingScheme` object can be found by calling method `getParentalRatingScheme()` on property `parentalRatingSchemes` of the `application/oipfParentalControlManager` object and the value of property `scheme` is not equal to "`dvb-si`", then the `value` property SHALL represent the index of the parental rating value inside the `ParentalRatingScheme` object, or -1 if the value cannot be found. If the value of property `scheme` is equal to "`dvb-si`", then this property SHALL be the integer representation of the string value of `ParentalRating` property `name`.

If no associated `ParentalRatingScheme` object can be found by calling method `getParentalRatingScheme` on property `parentalRatingSchemes` of the `application/oipfParentalControlManager` object, then the `value` property SHALL have value `undefined`.

`readonly Integer` **labels**

The labels property represents a set of parental advisory flags that may provide additional information about the rating.

The value of this field is a 32 bit integer value that represents a binary mask corresponding to the sum of zero or more label values defined in the table below. If no labels have been explicitly set, the value for the `labels` property SHALL be 0.

Valid labels include:

Value	Binary representation (least significant 16 bits)	Description
1	00000000 00000001	Indicates that a content item features sexual suggestive dialog.
2	00000000 00000010	Indicates that a content item features strong language.
4	00000000 00000100	Indicates that a content item features sexual situations.

8	00000000 00001000	Indicates that a content item features violence.
16	00000000 00010000	Indicates that a content item features fantasy violence.
32	00000000 00100000	Indicates that a content item features disturbing scenes.
64	00000000 01000000	Indicates that a content item features portrayals of discrimination.
128	00000000 10000000	Indicates that a content item features scenes of illegal drug use.
256	00000001 00000000	Indicates that a content item features strobing that could impact viewers suffering from Photosensitive epilepsy

`readonly String` **`region`**
The region to which the parental rating value applies as case-insensitive alpha-2 region code as defined in ISO 3166-1. Returns `undefined` if no specific region has been defined.

7.9.5 The ParentalRatingCollection class

```
typedef Collection<ParentalRating> ParentalRatingCollection
```

A `ParentalRatingCollection` represents a collection of parental rating values. See annex K for the definition of the collection template.

In addition to the methods and properties defined for generic collections, the `ParentalRatingCollection` class supports the additional properties and methods defined below.

7.9.5.1 Methods

`void` **`addParentalRating`**`( String scheme, String name, Integer value, Integer labels, String region )`		
Description	Creates a `ParentalRating` object instance for a given parental rating scheme and parental rating value, and adds it to the `ParentalRatingCollection` for a programme or channel.	
Arguments	*scheme*	A unique string identifying the parental rating scheme to which this value refers. See property `scheme` in Section 7.9.4.1 for more information about possible values.
	name	A case-insensitive string representation of the parental rating value. See property `name` in Section 7.9.4.1 for more information about possible values.
	value	The parental rating value represented as an Integer. See property `value` in Section 7.9.4.1 for more information about possible values.

	labels	A set of content rating labels that may provide additional information about the rating. See property `labels in` Section 7.9.4.1 for more information about possible values.
	region	The region to which the parental rating value applies as case-insensitive alpha-2 region code as defined in ISO 3166-1. Value must be `null` or `undefined` if no specific region has been identified.

7.10 Scheduled Recording APIs

This section describes the APIs needed to support control by a DAE application of the recording (PVR) functionality available to an OITF, including time-shift support, scheduled recording and storage of basic metadata about recorded items.

This section SHALL apply for OITFs that have indicated `<recording>` with value "`true`" as defined in Section 9.3.3 in its capability description.

7.10.1 The application/oipfRecordingScheduler embedded object

The OITF SHALL support the scheduling of recordings of broadcasts through the use of the following non-visual embedded object:

```
<object type="application/oipfRecordingScheduler"/>
```

Note that the functionality in this section SHALL adhere to the security model as specified in Section 10.1.

7.10.1.1 Methods

`ScheduledRecording` **`record`**`( Programme programme )`		
Description	Requests the scheduler to schedule the recording of the programme identified by the `programmeID` property of the programme. The other data contained in the programme object is used solely for annotation of the (scheduled) recording. If such programme metadata is provided, it is retained in the `ScheduledRecording` object that is returned if the recording of the programme was scheduled successfully, reflecting the possibility that not all relevant metadata might be available to the scheduler. If the recording could not be scheduled due to a scheduling conflict or lack of resources the value `null` is returned. Note that the actual implementation of this method should enable the scheduler to identify the domain of the service that issues the scheduling request in order to support future retrieval of the scheduled recording through the `getScheduledRecordings` method.	
Arguments	*programme*	The programme to be recorded, as defined in 7.16.2.

`ScheduledRecording` **`recordAt`**`( Integer startTime, Integer duration, Integer repeatDays, String channelID )`	
Description	Requests the scheduler to schedule the recording of the broadcast to be received over the channel identified by `channelID`, starting at `startTime` and continuing for `duration` minutes. If the recording was scheduled successfully, the resulting `ScheduledRecording` object is returned. If the recording could not be scheduled due to a scheduling conflict or lack of resources the value `null` is returned.

<table>
<tr><td></td><td colspan="2">Note that the actual implementation of this method should enable the scheduler to identify the domain of the service that issues the scheduling request in order to support future retrieval of the scheduled recording through the <code>getScheduledRecordings</code> method.</td></tr>
<tr><td>Arguments</td><td>startTime</td><td>The start of the time period of the recording measured in seconds since midnight (GMT) on 1/1/1970.</td></tr>
<tr><td></td><td>duration</td><td>The duration of the recording in seconds.</td></tr>
<tr><td></td><td>repeatDays</td><td>Bitfield indicating on which days of the week the recording SHOULD be repeated. Values are as follows:
<table>
<tr><th>Day</th><th>Bitfield Value</th></tr>
<tr><td>Sunday</td><td>0x01 (i.e. 00000001)</td></tr>
<tr><td>Monday</td><td>0x02 (i.e. 00000010)</td></tr>
<tr><td>Tuesday</td><td>0x04 (i.e. 00000100)</td></tr>
<tr><td>Wednesday</td><td>0x08 (i.e. 00001000)</td></tr>
<tr><td>Thursday</td><td>0x10 (i.e. 00010000)</td></tr>
<tr><td>Friday</td><td>0x20 (i.e. 00100000)</td></tr>
<tr><td>Saturday</td><td>0x40 (i.e. 01000000)</td></tr>
</table>
These bitfield values can be ‘OR’-ed together to repeat a recording on more than one day of a week (e.g. weekdays)

A value of 0x00 indicates that the recording will not be repeated.</td></tr>
<tr><td></td><td>channelID</td><td>The identifier of the channel from which the broadcasted content is to be recorded. Specifies either a ccid or ipBroadcastID (as defined by the <code>Channel</code> object in Section 7.13.12)</td></tr>
</table>

<table>
<tr><td colspan="2"><code>ScheduledRecordingCollection getScheduledRecordings()</code></td></tr>
<tr><td>Description</td><td>Returns a subset of all the recordings that are scheduled but which have not yet started. The subset SHALL include only scheduled recordings that were scheduled using a service from the same FQDN as the domain of the service that calls the method.</td></tr>
</table>

<table>
<tr><td colspan="2"><code>ChannelConfig getChannelConfig()</code></td></tr>
<tr><td>Description</td><td>Returns the channel line-up of the tuner in the form of a <code>ChannelConfig</code> object as defined in Section.7.13.8. This includes the favourite lists. The <code>ChannelConfig</code> object returned from this function SHALL be identical to the <code>ChannelConfig</code> object returned from the <code>getChannelConfig()</code> method on the video/broadcast object as defined in 7.13.3.</td></tr>
</table>

void **remove**(ScheduledRecording recording)		
Description	Removes a scheduled recording. As with the record method, only the programmeID property of the scheduled recording SHALL be used to identify the scheduled recording to remove. The other data contained in the scheduled recording SHALL NOT used when removing a scheduled recording.	
Arguments	*recording*	The scheduled recording to be removed.

Programme **createProgrammeObject**()	
Description	Factory method to create an instance of Programme

7.10.2 The ScheduledRecording class

The ScheduledRecording object represents a scheduled programme in the system, i.e. a recording that is scheduled but which has not yet started. . The values of the properties of a ScheduledRecording (except for startPadding and endPadding) are provided when the object is created using one of the record() methods in Section 7.10.1, for example by using a corresponding Programme object as argument for the record() method, and can not be changed for this scheduled recording object (except for startPadding and endPadding).

7.10.2.1 Constants

The following constants are defined as properties of the ScheduledRecording class:

Name	Value	Use
ID_TVA_CRID	0	Used in the programmeIDType property to indicate that the programme is identified by its TV-Anytime CRID (Content Reference Identifier).
ID_DVB_EVENT	1	Used in the programmeIDType property to indicate that the programme is identified by a DVB URL referencing a DVB-SI event as enabled by Section 4.1.3 of [OIPF_META2]. Support for this constant is OPTIONAL.

7.10.2.2 Properties

Integer **startPadding**
The amount of padding to add at the start of a scheduled recording, in seconds. This property is initialised to the value of the Configuration.pvrStartPadding property. The default OITF defined start padding MAY be changed through property pvrStartPadding of the Configuration class as defined in Section 7.3.2.

Integer **endPadding**
The amount of padding to add at the end of a scheduled recording, in seconds. This property is

initialised to the value of the `Configuration.pvrEndPadding` property. The default OITF defined end padding MAY be changed through property `pvrEndPadding` of the `Configuration` class as defined in Section 7.3.2.

`readonly Integer` **`repeatDays`**

Bitfield indicating on which days of the week the recording SHOULD be repeated. Values are as follows:

Day	Bitfield Value
Sunday	0x01 (i.e. 00000001)
Monday	0x02 (i.e. 00000010)
Tuesday	0x04 (i.e. 00000100)
Wednesday	0x08 (i.e. 00001000)
Thursday	0x10 (i.e. 00010000)
Friday	0x20 (i.e. 00100000)
Saturday	0x40 (i.e. 01000000)

These bitfield values can be 'OR'-ed together to repeat a recording on more than one day of a week (e.g. weekdays)

A value of 0x00 indicates that the recording will not be repeated.

For recordings other than those created using the `recordAt()` method, the value of this property SHALL be `undefined`.

`readonly String` **`name`**

The short name of the scheduled recording, e.g. 'Star Trek: DS9'.

`readonly String` **`longName`**

The long name of the scheduled recording, e.g. 'Star Trek: Deep Space Nine'. If the long name is not available, this property will be `undefined`.

`readonly String` **`description`**

The description of the scheduled recording, e.g. an episode synopsis. If no description is available, this property will be `undefined`.

`readonly String` **`longDescription`**

The long description of the programme. If no description is available, this property will be `undefined`.

`readonly Integer` **`startTime`**
The start time of the scheduled recording, measured in seconds since midnight (GMT) on 1/1/1970. The value for the `startPadding` property can be used to indicate if the recording has to be started before the `startTime` (as defined by the `Programme` class).

`readonly Integer` **`duration`**
The duration of the scheduled recording (in seconds). The value for the `endPadding` property can be used to indicate how long the recording has to be continued after the specified duration of the recording.

`readonly Channel` **`channel`**
Reference to the broadcast channel where the scheduled programme is available.

`readonly Boolean` **`isSeries`**
If true, then when a subsequent episode of a programme becomes available it SHOULD be added to the recording list automatically. Note: Where several episodes of a season are available, then only the latest scheduled recording will carry the `isSeries` flag.

`readonly String` **`programmeID`**
The unique identifier of the scheduled programme or series, e.g. a TV-Anytime CRID (Content Reference Identifier).

`readonly Integer` **`programmeIDType`**
The type of identification used to reference the programme, as indicated by one of the ID_* constants defined in Section 7.10.2.1.

`readonly Integer` **`episode`**
The episode number for the programme if it is part of a series. This property is `undefined` when the programme is not part of a series or the information is not available.

`readonly Integer` **`totalEpisodes`**
If the programme is part of a series, the total number of episodes in the series. This property is `undefined` when the programme is not part of a series or the information is not available.

readonly ParentalRatingCollection **parentalRating**
A collection of parental rating values for the programme for zero or more parental rating schemes supported by the OITF. The value of this property is typically provided by a corresponding "Programme" object that is used to schedule the recording and can not be changed for this scheduled recording object. If no parental rating information is available for this scheduled recording, this property is a ParentalRatingCollection object (as defined in Section 7.9.5) with length 0. Note that if the parentalRating property contains a certain parental rating (e.g. PG-13) and the broadcast channel associated with this scheduled recording has metadata that says that the content is rated PG-16, then the conflict resolution is implementation dependent.

7.10.3 The ScheduledRecordingCollection class

```
typedef Collection<ScheduledRecording> ScheduledRecordingCollection
```

The ScheduledRecordingCollection class represents a collection of ScheduledRecording objects. See annex K for the definition of the collection template.

7.10.4 Extension to application/oipfRecordingScheduler for control of recordings

The OITF SHALL support the following extensions to the application/oipfRecordingScheduler object defined in Section 7.10.1.

This subsection SHALL apply for OITFs that have indicated support for the extended PVR management functionality by adding the attribute 'manageRecordings = true' to the <recording> element in the client capability description as defined in Section 9.3.3

The functionality as described in this section is subject to the security model of Section 10.

7.10.4.1 Properties

readonly ScheduledRecordingCollection **recordings**
Provides a list of scheduled and recorded programmes in the system. This property may only provide access to a subset of the full list of recordings, as determined by the value of the manageRecordings attribute of the <recording> element in the client capability description (see Section 9.3.3). Note: Where a series is being recorded, every recorded episode SHALL exist as an independent entry. Only the scheduled recording SHALL carry the isSeries property.

readonly DiscInfo **discInfo**
Get information about the status of the local storage device. The DiscInfo class is defined in Section 7.16.4.

function **onPVREvent**(Integer state, Recording recording)
This function is the DOM 0 event handler for notification of changes in the state of recordings. See the

definition of the corresponding DOM 2 `PVREvent` in Section 1.1.1 for more details.

The specified function is called with the following arguments:

- `Integer state` – The current state of the recording. One of:

Value	Description
1	The recording has started.
2	The recording has stopped, having completed.
3	The recording sub-system is unable to record due to resource limitations.
4	There is insufficient storage space available. (Some of the recording may be available).
6	The recording has stopped before completion due to unknown (probably hardware) failure.
7	The recording has been newly scheduled.
8	The recording has been deleted (for complete or in-progress recordings) or removed from the schedule (for scheduled recordings).
9	The recording is due to start in a short time.
10	The recording has been updated.

- `ScheduledRecording recording` – The recording to which this event refers.

7.10.4.2 Methods

`Recording` **`getRecording`**`( String id )`		
Description	Returns the `Recording` object for which the value of the `Recording.id` property corresponds to the given `id` parameter. If such a `Recording` does not exist, the method returns `null`.	
Arguments	*id*	Identifier corresponding to the `id` property of a `Recording` object.

`void` **`remove`** `( ScheduledRecording recording )`		
Description	Remove a recording (either scheduled, in-progress or completed). For non-privileged applications, recordings SHALL only be removed when they are scheduled but not yet started and the recording was scheduled by the current service. If the A/V Control object is referring to the indicated recording the state in A/V Control object shall be automatically changed to 6 (the error state).	
Arguments	*recording*	The recording to be removed.

void **stop**(Recording recording)		
Description	Stop an in-progress recording. The recording SHALL NOT be deleted.	
Arguments	*recording*	The recording to be stopped.

void **refresh**()	
Description	Update the recordings property to show the current status of all recordings.

7.10.4.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onPVREvent	PVREvent	Bubbles: No Cancelable: No Context Info: state, recording

Note: the DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Remote UIs SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Remote UIs that use DOM 2 event handlers SHALL call the addEventListener() method on the application/oipfScheduledRecording object itself. The third parameter of addEventListener, i.e. "useCapture", will be ignored.

7.10.5 The Recording class

The Recording class represents an in-progress or completed recording being made available through the extended PVR management functionality as defined in Section 7.10.4. This class implements the ScheduledRecording interface (see Section 7.10.2).

Recordings MAY be "manual" in that they simply record a channel at a certain time, for a period - analogous to a traditional VCR - or alternatively recordings can be programme based.

Values of properties in the Recording object SHALL be obtained from metadata about the recorded programme and are typically copied from the Programme used for scheduling a recording by the record(Programme programme) method of the application/oipfRecordingScheduler object. See Section 7.10.4 for more information about the mapping between the properties of a Programme and the BCG metadata.

7.10.5.1 Properties

readonly Integer **state**
The state of the recording. One of:

Value	Description
1	The recording has started.
2	The recording has stopped, having completed.
3	The recording sub -system is unable to record due to resource limitations.
4	There is insufficient storage space available. (Some of the recording may be available).
5	The recording has not taken place due to unknown (probably hardware) failure.
6	The recording has only partially completed due to a clash or hardware failure. There are three possible conditions for this: 1) The end of the recording is missed. 2) The start of the recording is missed. 3) A piece from the centre of the recording is missed (e.g. due to the receiver rebooting or a transient failure of the network connection).

`readonly String` **`id`**

An identifier for this recording. This value SHALL be unique to this recording and so can be used to compare two recording objects to see if they refer to the same recording. The OITF SHALL guarantee that recording identifiers are unique in relation to download identifiers and `CODAsset` identifiers.

`readonly Boolean` **`isManual`**

If `false`, then any fields whose name matches a field in the `Programme` object contains details from the programme guide on the programme that has been recorded.

If `true`, only the channel, start time and duration of the recording are valid.

`Boolean` **`doNotDelete`**

If `true`, then this recording should not be automatically deleted by the system.

`Integer` **`saveDays`**

The number of days for which an individual or manual recording SHOULD be saved. Recordings older than this value MAY be deleted. This property is initialised to the value of the `Configuration.pvrSaveDays` property.

`Integer` **`saveEpisodes`**

The number of episodes of a series-link that SHOULD be saved. Older episodes MAY be deleted. This is only valid when set on the latest scheduled recording in the series. This property is initialised to the value of the `Configuration.pvrSaveEpisodes` property.

`readonly Boolean` **`blocked`**

Flag indicating whether the programme is blocked due to parental control settings or conditional access restrictions.

`readonly ParentalRatingCollection` **`parentalRatings`**

A collection of parental rating values for the programme for zero or more parental rating schemes supported by the OITF, defined using the `ParentalRatingCollection` object as specified in Section 7.9.5. If no parental rating information is available for this scheduled recording, this property is a `ParentalRatingCollection` object with length 0.

Note that if the `parentalRatings` property contains a certain parental rating (e.g. PG-13) and the broadcast channel associated with this scheduled recording has metadata that says that the content is rated PG-16, then the conflict resolution is implementation dependent.

`readonly Integer` **`showType`**

Flag indicating the type of show. This field SHALL take one of the following values:

Value	Description
0	The show is live.
1	The show is a first-run show.
2	The show is a rerun.

`readonly Boolean` **`subtitles`**

Flag indicating whether subtitles or closed-caption information is available.

`readonly StringCollection` **`subtitleLanguages`**

Supported subtitle languages, indicated by iso639 language codes.

`readonly Boolean` **`isHD`**

Flag indicating whether the programme has high-definition video.

readonly Boolean **isWidescreen**
Flag indicating whether the programme is broadcast in widescreen.

readonly Integer **audioType**
Bitfield indicating the type of audio that is available for the programme. Since more than one type of audio may be available for a given programme, the value of this field SHALL consist of one or more of the following values ORed together:

Value	Description
1	Mono audio
2	Stereo audio
4	Multi-channel audio

readonly Boolean **isMultilingual**
Flag indicating whether more than one audio language is available for this recording.

readonly StringCollection **audioLanguages**
Supported audio languages, indicated by iso639 language codes.

readonly StringCollection **genres**
A collection of genres that describe this programme.

readonly Integer **recordingStartTime**
The actual start time of the recording, including any padding. This MAY not be the same as the scheduled start time of the recorded programme (e.g. due to a recording starting late, or due to start/end padding).

readonly Integer **recordingDuration**
The actual duration of the recording, including any padding. This MAY not be the same as the scheduled duration of the recording (e.g. due to a recording finishing early, or due to start/end padding).

readonly BookmarkCollection **bookmarks**
A collection of the bookmarks set in a recording. If no bookmarks are set, the collection SHALL be

empty.

readonly Boolean **locked**
Flag indicating whether the current state of the parental control system prevents the recording from being viewed (e.g. a correct parental control PIN has not been entered to allow the recording to be viewed).

7.10.6 The RecordingCollection class

```
typedef Collection<Recording> RecordingCollection
```

The `RecordingCollection` class represents a collection of `Recording` objects. See annex K for the definition of the collection template.

7.10.7 The Bookmark class

The `Bookmark` class represents a bookmark or chapter mark in a recording or CoD asset. This is not a web bookmark – instead, it is a point from which the viewer may want to resume playback of a piece of content. These MAY be set implicitly without user intervention (e.g. at the point where a user stops watching a recording, in order to allow them to resume from that point later) or explicitly by the user (e.g. at the start of a favourite scene).

7.10.7.1 Properties

readonly Integer **time**
The time at which the bookmark is set, in seconds from the start of the content item.

readonly String **name**
The name of the bookmark.

7.10.8 The BookmarkCollection class

```
typedef Collection<Bookmark> BookmarkCollection
```

A `BookmarkCollection` is a collection of bookmarks, ordered by time. See annex K for the definition of the collection template. In addition to the methods and properties defined for generic collections, the `BookmarkCollection` class supports the additional properties and methods defined below.

NOTE: In principle bookmarks MAY be stored on in the network however the protocol for communicating bookmarks between the OITF and the network is not defined in the present document.

7.10.8.1 Methods

Bookmark **addBookmark**(Integer time, String name)

Description	Add a new bookmark to the collection. If the bookmark cannot be added (e.g. because the value given for time lies outside the length of the recording), this method SHALL return `null`.	
Arguments	*time*	The time at which the bookmark is set, in seconds since the start of the recording.
Arguments	*name*	The name of the bookmark.

`void` **`removeBookmark`**`( Bookmark bookmark )`		
Description	Remove a bookmark from the collection.	
Arguments	*bookmark*	The bookmark to be removed.

7.11 Remote Management APIs

This section defines interfaces to perform remote diagnostics and management of the device.

Browser based remote management SHALL be supported by OITFs that have indicated `<remote_diagnostics>true</remote_diagnostics>` in their capability profile (as defined in Section 9.3.12)

7.11.1 The application/oipfRemoteManagement embedded object

The `application/oipfRemoteManagement` embedded object has the following properties and methods.

Access to the functionality of the `application/oipfRemoteManagement` embedded object SHALL adhere to the security requirements as defined in Section 10.

7.11.1.1 Properties

`readonly String` **`vendorName`**
String identifying the vendor name of the device. The value of this property SHALL be the same as the value of the `LocalSystem.vendorName` property (see section 7.3.3.2).

`readonly String` **`modelName`**
String identifying the model name of the device. The value of this property SHALL be the same as the value of the `LocalSystem.modelName` property (see section 7.3.3.2).

`readonly String` **`softwareVersion`**
String identifying the version number of the platform firmware. The value of this property SHALL be the same as the value of the `LocalSystem.softwareVersion` property (see section 7.3.3.2).

`readonly String` **`hardwareVersion`**
String identifying the version number of the platform hardware. The value of this property SHALL be

the same as the value of the `LocalSystem.hardwareVersion` property (see section 7.3.3.2).

7.11.1.2 Methods

`String` **`getParameter`**`( String parameterName )`		
Description	Returns the requested parameter.	
Arguments	*parameterName*	"`SAMPLE_PACKET_LOSS`": This queries the RTP packet loss since the last call to this function, or the start of the current RTP content item, whichever is more recent. The returned string is of the format "<time in milliseconds since the last sample> <fraction lost> <number of packets lost>". These fields (i.e. <xxx>) are defined as described in Section 6.4.2 of [RFC3550] and are decimal numbers (encoded as strings). If no content item is playing an empty string is returned. "`SAMPLE_DECODER_ERRORS`": This queries the decoder errors since the last call to this function, or the start of the current RTP content item, whichever is more recent. The returned string is of the format "<time in milliseconds since the sample> <total number of frames decoded> <total number of errors>". These fields are decimal numbers (encoded as strings). If no content item is playing an empty string is returned. "`CUMULATIVE_PACKET_LOSS`": This queries the RTP packet loss since the start of the current RTP content item. The returned string is of the format "<time in milliseconds of this sample within the content> <fraction lost> <number of packets lost>". These fields (i.e. <xxx>) are defined as described in Section 6.4.2 of [RFC3550] and are decimal numbers (encoded as strings). If no content item is playing an empty string is returned. "`CUMULATIVE_DECODER_ERRORS`": This queries the decoder errors since the start of the current RTP content item, whichever is more recent. The returned string is of the format "<time in milliseconds of this sample within the content> <total number of frames decoded> <total number of errors>". These fields are decimal numbers (encoded as strings). If no content item is playing an empty string is returned. Optionally, further vendor specific parameters may be supported. In the case that a parameter is requested that a device does not support, it SHALL return an empty string.

`String` **`setParameter`**`( String parameterName, String value )`		
Description	Sets the requested parameter. Support for this API is optional.	
Arguments	*parameterName*	The name of the parameter.
	value	The value of the parameter.

<table>
<tr><td colspan="4">Integer triggerSoftwareUpdate(String token)</td></tr>
<tr><td>Description</td><td colspan="3">Triggers an OITF to start its software update process. The process itself and any user involvement (e.g. to confirm agreement for a software update) is not defined. The method SHALL block until it can be determined if there's an update which will be applied. The mechanism by which this is determined is outside the scope of this specification. Downloading and application of any updates SHALL be performed asynchronously. The returned integer is a result code that can take the following values:
<table>
<tr><th>Result message</th><th>Description</th><th>Semantics</th></tr>
<tr><td>0</td><td>Successful</td><td>The request is successful and the device software will be updated.</td></tr>
<tr><td>1</td><td>Unknown error</td><td>triggerSoftwareUpdate() failed because an unspecified error occurred.</td></tr>
<tr><td>2</td><td>Invalid token</td><td>triggerSoftwareUpdate() failed because the token is not valid.</td></tr>
<tr><td>3</td><td>No update available</td><td>triggerSoftwareUpdate() failed, because no update exists.</td></tr>
</table></td></tr>
<tr><td>Arguments</td><td>token</td><td colspan="2">An optional token string used to assist in the software update process. The token may be used to transfer credentials information to control the software update.</td></tr>
</table>

7.12 Metadata APIs

This section defines the Javascript APIs used by DAE applications for reading and searching metadata about programmes and channels. This API does not specify whether these query operations are carried out on the OITF or whether they require communication with a server.

The metadata API provides DAE applications with high-level access to metadata about programmes and channels. This document describes the mapping between this API and CoD and programme guide metadata. Mappings between this API and other metadata sources are not specified in this document.

This section SHALL apply for OITFs that have indicated <clientMetadata> with value "true" and a type attribute with value "bcg" or "dvb-si" as defined in Section 9.3.7 in their capability profile.

Note that in order to access the metadata of programmes and channels several extensions to the Programme and Channel classes have been defined when the OITF has indicated support for <clientMetadata>. See sections 7.16.2.3 and 7.13.12.3 for more information).

The functionality as described in this section is subject to the security model of Section 10 (in particular Section 10.1.4.6).

7.12.1 The application/oipfSearchManager embedded object

OITFs SHALL implement the "application/oipfSearchManager" embedded object. This object provides a mechanism for applications to create and manage metadata searches.

The following example shows how a metadata search can be constructed and executed:

```
// Event handler function for asynchronous search results
```

```
function handleSearchResults() {
  if ((state == 0) || (state ==1)) {
    //more results are available, or our search has finished

    // update the results.  Doing this asynchronously means
    // that if we're working with the current set of results,
    // we get the new results  when it suits the application.
    search.results.update();

    // do stuff with the results
    var myResult = search.results[0];

             //get the next page of results
    search.results.getResults(10, 20);
  }
}

// Function that creates and starts a search
function doSearch() {

  // create a new search for on-demand content
  mySearchManager = document.getElementById("searchManager");
  mySearch = mySearchmanager.createSearch(2);

  // search for any programme with "space" in the title as a word
  // or part of a word
  myQuery = mySearch.createQuery(
      "urn:tva:transport:fieldIDs:2002:Title",
      6,
      "space");
  mySearch.setQuery(myQuery);

  // return results alphabetically by title
  mySearch.orderBy("urn:tva:transport:fieldIDs:2002:Title", true);

  mySearchManager.onMetadataSearch = handleSearchResults;

  if (mySearch.results.getResults(0, 10)) {
    // some results are available immediately, e.g. because
    // they were cached

    // do stuff with the results
    var myResult = mySearch.results[0];
  }
}
```

7.12.1.1 Properties

readonly Integer **guideDaysAvailable**
The number of days for which guide data is available. A value of -1 means that the amount of guide data available is unknown.

function **onMetadataUpdate(Integer action, Integer info, Object object)**
This function is the DOM 0 event handler for events indicating changes in metadata. This SHALL be raised when changes to the parental control settings change the lock status of an item, or when a new version of the metadata becomes available. The specified function is called with the arguments action, info and object. These arguments are defined as follows: Number action – the type of update that has taken place. This field will take one of the following values:

Value	Description
1	A new version of metadata is available (see clause 4.1.2.1.2 of [META]) or and applications SHOULD discard all references to

	Programme objects immediately and re-acquire them.
2	A change to the parental control flags for a content item has occurred (e.g. the user has unlocked the parental control features of the receiver, allowing a blocked item to be played).
3	A flag affecting the filtering criteria of a channel has changed. Applications MAY listen for events with this action code to update lists of favourite channels, for instance.

Number info – extended information about the type of update that has taken place. If the action argument is set to the value 4, the value of this field SHALL be one or more of the following:

Value	Description
1	The list of blocked channels has changed.
2	A list of favourite channels has changed.
4	The list of hidden channels has changed.

If the action argument is set to the value 3, the value of this field SHALL be one or more of:

Value	Description
1	The block status of a content item has changed.
2	The lock status of a content item has changed.

This field is treated as a bitfield, so values MAY be combined to allow multiple reasons to be passed.

Object object – the affected channel, programme, or CoD asset. If more than one is affected, then this argument SHALL take the value null.

`function` **`onMetadataSearch( MetadataSearch search, Integer state )`**

This function is the DOM 0 event handler for events relating to metadata searches. The specified function is called with the arguments search and state. These arguments are defined as follows:

MetadataSearch search – the affected search

Number state – the new state of the search

Value	Description
0	Search has finished. This event SHALL be generated when a search has completed or been aborted.
1	More search results are available. Calling `update()` on the `SearchResults` object SHALL update the list of results to include the newly-retrieved data.

	2	The data returned by the search is no longer valid, e.g. because of a change in the metadata. Applications that still require the data SHALL repeat the search.	

7.12.1.2 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onMetadataSearch`	`MetadataSearch`	Bubbles: No Cancelable: No Context Info: `search`, `state`
`onMetadataUpdate`	`MetadataUpdate`	Bubbles: No Cancelable: No Context Info: `action`, `info`, `object`

These events are targeted at the `application/oipfSearchManager` object.

7.12.1.3 Methods

`MetadataSearch` **`createSearch`**`( Integer searchTarget )`		
Description	Create a `MetadataSearch` object that can be used to search the metadata.	
Arguments	*searchTarget*	The metadata that should be searched. Valid values of the `searchTarget` parameter are: **Value** / **Description** 1 – Metadata relating to scheduled content shall be searched. 2 – Metadata relating to content on demand shall be searched. These values are treated as a bitfield, allowing searches to be carried out across multiple search targets.

`ChannelConfig` **`getChannelConfig`**`()`	
Description	Returns the channel line-up of the tuner in the form of a `ChannelConfig` object as

	defined in Section 7.13.8. This includes the favourite lists. The `ChannelConfig` object returned from this function SHALL be identical to the `ChannelConfig` object returned from the `getChannelConfig()` method on the video/broadcast object as defined in 7.13.3.

7.12.2 The MetadataSearch class

A `MetadataSearch` object represents a query of the BCG and SD&S metadata about available programmes. Applications can create `MetadataSearch` objects using the `createSearch()` method on the `application/oipfSearchManager` object. When metadata queries are performed on a remote server, the protocol used is defined in section 4.1.2.2 of [OIPF_META2].

Changes to constraints or the ordering of search results SHALL be applied when the `getResults()` method on the corresponding `SearchResults` object is called.

Due to the nature of metadata queries, searches are asynchronous and events are used to notify the application that results are available. `MetadataSearchEvents` SHALL be targeted at the `application/oipfSearchManager` object.

To minimise race conditions, results are updated on request rather than dynamically. Upon receipt of a `MetadataSearchEvent` indicating that more results are available, applications SHALL call `update()` on the corresponding `SearchResults` object to get the new results.

7.12.2.1 Properties

`readonly Integer` **`searchTarget`**

The target(s) of the search. Valid values of the `searchTarget` parameter are:

Value	Description
1	Metadata relating to scheduled content SHALL be searched.
2	Metadata relating to on-demand content SHALL be searched.

These values SHALL be treated as a bitfield, allowing searches to be carried out across multiple search targets.

`Query` **`query`**

The query that will be carried out by this search.

`readonly SearchResults` **`result`**

The results found so far.

The values within result MAY change after subsequent calls to its `update()` method.

7.12.2.2 Methods

`void` **`setQuery`**`( Query query )`

Description	Set the query terms to be used for this search, discarding any previously-set query terms. Calling this method when a search is in progress SHALL: 1. Abort any outstanding requests for results (equivalent to calling `results.abort()`). 2. Invalidate any existing search results and dispatch a `MetadataSearch` event with a value of 2 for the state argument.	
Arguments	*Query*	The query terms to be used

`void` **`addRatingConstraint`**`( ParentalRatingScheme scheme, Integer threshold )`		
Description	Constrain the search to only include results whose parental rating value is below the specified threshold.	
Arguments	*scheme*	The parental rating scheme upon which the constraint SHALL be based. If the value of this argument is `null`, any existing parental rating constraints SHALL be cleared.
	threshold	The threshold above which results SHALL NOT be returned. If the value of this argument is `null`, any existing constraint for the specified parental rating scheme SHALL be cleared.

`void` **`addCurrentRatingConstraint`**`()`	
Description	Constrain the search to only include results whose parental rating value is below the threshold currently set by the user.

`void` **`addChannelConstraint`**`( ChannelList channels )`		
Description	Constrain the search to only include results from the specified channels. If a channel constraint has already been set, subsequent calls to `addChannelConstraint()`SHALL add the specified channels to the list of channels from which results should be returned. For CoD searches, adding a channel constraint SHALL have no effect.	
Arguments	*channels*	The channels from which results SHALL be returned. If the value of this argument is `null`, any existing channel constraint SHALL be removed.

`void` **`addChannelConstraint`**`( Channel channel )`	
Description	Constrain the search to only include results from the specified channel. If a channel constraint has already been set, subsequent calls to `addChannelConstraint()` SHALL add the specified channel to the list of channels from which results should be returned. For CoD searches, adding a channel constraint SHALL have no effect.

Arguments	*channel*	The channel from which results SHALL be returned. If the value of this argument is `null`, any existing channel constraint SHALL be removed.

`void` **`orderBy`**`( String field, Boolean ascending )`		

Description	Set the order in which results SHOULD be returned in future. Any existing search results SHALL not be re-ordered. Subsequent calls to `orderBy()` will apply further levels of ordering within the order defined by previous calls. For example:
	`orderBy("ServiceName", true);`
	`orderBy("PublishedStart", true);`
	will cause results to be ordered by service name and then by start time for results with the same channel number.	

Arguments	*field*	The name of the field by which results SHOULD be sorted. A value of `null` indicates that any currently-set order SHALL be cleared and the default sort order should be used.
	ascending	Flag indicating whether the results SHOULD be returned in ascending or descending order.

`Query` **`createQuery`**`( String field, Integer comparison, String value )`		
Description	Create a metadata query for a specific value in a specific field within the metadata. Simple queries MAY be combined to create more complex queries. Applications SHALL follow the ECMAScript type conversion rules to convert non-string values into their string representation, if necessary.	
Arguments	*field*	The name of the field to compare. Fields are identified by the fieldIDs defined in annex B.2 of [TVA-BID], or using simplified XPath notation. The '/' operator is the only permitted XPath operator.
	comparison	The type of comparison. One of:

Value	Description
0	True if the specified value is equal to the value of the specified field.
1	True if the specified value is not equal to the value of the specified field.
2	True if the value of the specified field is greater than the specified value.
3	True if the value of the specified field is greater than or equal to the specified value.
4	True if the value of the specified field is less than the specified value.
5	True if the value of the specified field is less than or equal to the specified value.

		6	True if the string value of the specified field contains the specified value. This operation SHALL be case insensitive, and SHALL match parts of a word as well as whole words (e.g. a value of “term” will match a field value of “Terminator”).
		7	True if the specified field exists.
	value	The value to check. Applications SHALL follow the ECMAScript type conversion rules to convert non-string values into their string representation, if necessary	

`void` **`findProgrammesFromStream`**`( Channel channel, Integer startTime, Integer count )`		
Description	Retrieve guide data for a specified number of programmes from a given channel from metadata contained in the stream as defined in section 4.1.3 of [OIPF_META2]. Searches made using this method will implicitly remove any existing constraints, ordering or queries created by prior calls to methods on this object.	
Arguments	*channel*	The channel for which programme information should be found.
	startTime	The start of the time period for which results should be returned measured in seconds since midnight (GMT) on 1/1/1970. The start time is inclusive; any programmes starting at the start time, or which are showing at the start time, will be included in the search results. If `null`, the search will start from the current time.
	count	The number of programmes for which information should be returned.

7.12.3 The Query class

The `Query` class represents a metadata query that the user wants to carry out. This may be a simple search, or a complex search involving Boolean logic. Queries are immutable; an operation on a query SHALL return a new `Query` object, allowing applications to continue referring to the original query.

The examples below show how more complex queries can be constructed:

```
Query qa = mySearch.createQuery("Title", 6, "Terminator");
Query qb = mySearch.createQuery ("SpokenLanguage", 0, "fr-CA");
Query qc = qb.and(qa.negate());
```

7.12.3.1 Methods

`Query` **`and`**`( Query query )`		
Description	Create a query based on the logical AND of the predicates represented by the current query and the argument query.	
Arguments	*query*	The second predicate for the AND operation.

Query **or**(Query query)		
Description	Create a query based on the logical OR of the predicates represented by the current query and the argument query.	
Arguments	*query*	The second predicate for the OR operation.

Query **not**()
Description	Create a new query that is the logical negation of the current query.

7.12.4 The SearchResults class

The `SearchResults` class represents the results of a metadata search. Since the result set may contain a large number of items, applications request a 'window' on to the result set, similar to the functionality provided by the OFFSET and LIMIT clauses in SQL.

Applications MAY request the contents of the result in groups of an arbitrary size, based on an offset from the beginning of the result set. The data SHALL be fetched from the appropriate source, and application SHALL be notified when the data is available.

Next to the properties and methods defined below a `SearchResults` object SHALL support the array notation to access the results in this collection.

7.12.4.1 Properties

readonly Integer **length**
The number of items in the currently available results. If results are fetched asynchronously, the value of this property SHALL be zero until after `update()` has been called.

readonly Integer **offset**
The current offset into the total result set.

readonly Integer **totalSize**
The total number of items in the result set. If results are fetched asynchronously, the value of this property SHALL be undefined until `getResults()` has been called and a `MetadataSearchEvent` notifying the application that results are available has been dispatched.

7.12.4.2 Methods

Object **item**(Integer index)	
Description	Return the item at position index in the collection of currently available results, or

	undefined if no item is present at that position. This function SHALL only return objects that are instances of Programme when searching metadata for scheduled content, or CODAsset, CODFolder, or CODService when searching CoD metadata	
Arguments	*index*	The index into the result set.

Boolean **getResults**(Integer offset, Integer count)		
Description	Perform the search and retrieve a subset of the items that match the query. Results MAY be returned both synchronously and asynchronously, depending on whether data is available from the cache. If getResults() returns false, results are not available until the notification events have been returned and update() has been called. If getResults() returns true, results are available immediately, and the application need not wait for MetadataSearchEvents indicating that results are available or call update() to obtain the results. For results returned as a result of the same call to getResults(), the full result set may build up over time – the availability of new entries in the result set will be indicated by notification events. Subsequent calls to getResults() will clear the result set, so only results fetched for the most recent call to getResults() will be available to applications.	
Arguments	*offset*	The number of items at the start of the result set to be skipped before data is retrieved.
	count	The number of results to retrieve.

void **abort**()	
Description	Abort any outstanding request for results. Items currently in the collection SHALL be removed (i.e. the value of the length property SHALL be 0 and any calls to item()SHALL return undefined).

void **update**()	
Description	Through the update method new results are made available to applications. When a call to getResults() has returned false and results are fetched asynchronously, this method must be called after an application has received a notification event informing it that new results are available. The results may be delivered over multiple notification events. Until this method is called, results returned by asynchronous requests SHALL NOT be available to applications. This ensures that applications have a consistent view of the available results, without the result set changing asynchronously. This enables applications to (for example) iterate over the current result set and update their UI before retrieving the new results which have been returned to the OITF but are not yet available to applications.

7.13 Scheduled content and hybrid tuner APIs

This section SHALL apply to OITFs that have indicated support for tuner control (i.e. `<video_broadcast>true</video_broadcast>` as defined in Section 9.3.1) in their capability. It describes the `video/broadcast` embedded object needed to support display and control by a DAE application of scheduled content received over local tuner functionality available to an OITF, including the conveyance of the channel list to the server. The term "tuner" is used here to identify a piece of functionality to enable switching between different types of scheduled content services that are identified through logical channels. This includes IP broadcast channels, as well as traditional broadcast channels received over a hybrid tuner.

7.13.1 The video/broadcast embedded object

The OITF SHALL support the `video/broadcast` embedded object with the following properties and methods, which SHALL adhere to the tuner related security requirements in Section 10.1.4.1. The MIME type of this object SHALL be "`video/broadcast`".

7.13.1.1 State diagram for video/broadcast objects

The state diagram below shows the states that a video/broadcast object may be in. Dashed lines indicate automatic transitions between states. The `video/broadcast` object SHALL be in the `unrealized` state when it is instantiated.

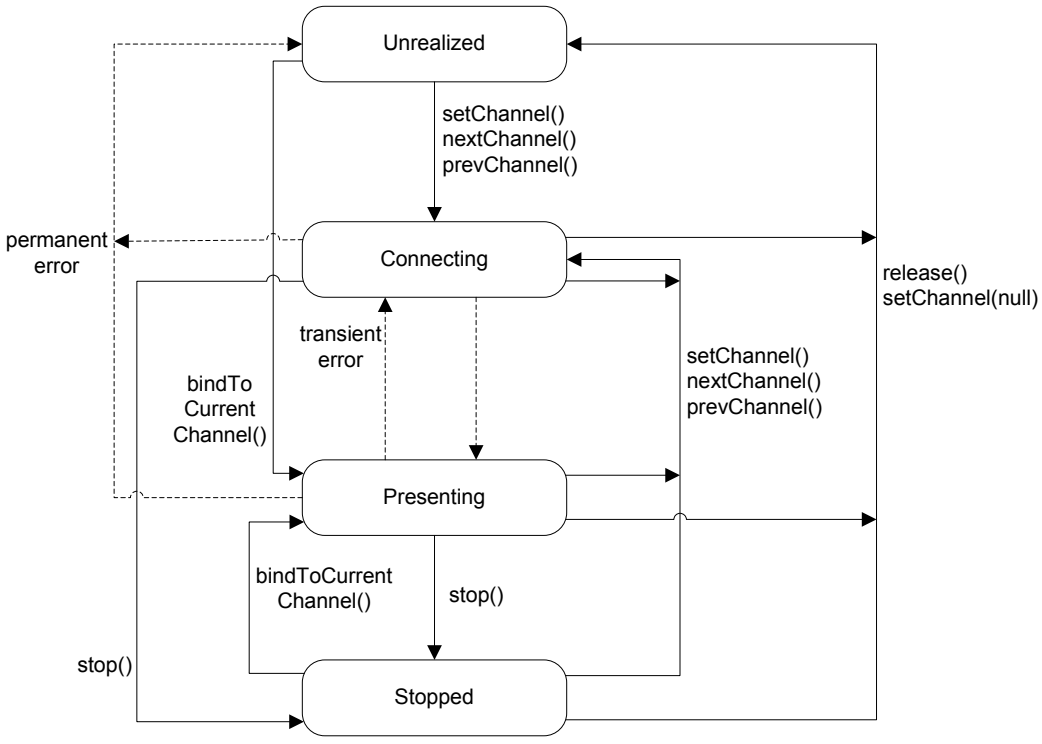


Figure 15: State diagram for embedded `video/broadcast` objects.

When the `setChannel()`, `nextChannel()` or `prevChannel()` method is called from the `unrealized`, `connecting` or `presenting` states, the object shall transition to the `connecting` state, in which the terminal attempts to connect to the broadcast stream. If `setChannel()` is called with a null parameter, the object shall transition to the `unrealized` state.

This may mean connecting to an IP multicast stream or tuning to a new transport stream and demultiplexing appropriate sub-streams. When this has completed and media is being presented, the object transitions automatically to the `presenting` state.

When the `bindToCurrentChannel()` method is called from the unrealized or stopped states, the object shall transition directly to the `presenting` state.

If the channel currently being presented changes due to an action outside the application (for example, the user pressing the CH+ key on the remote) then any `video/broadcast` object presenting that channel (e.g. as the result of a call to `bindToCurrentChannel()`) SHALL perform the same state transitions and dispatch the same events as if the channel change operation was initiated by the application.

Scarce resources such as media decoders SHALL be claimed either during the `connecting` state, or during the transition from the `connecting` to the `presenting` states. Resources SHALL be released when the `video/broadcast` object transitions to the `unrealized` state. Transitioning from the `presenting` to the `connecting` state SHOULD NOT cause scarce resources to be released.

During media presentation, transient errors (e.g. transient errors in the bitstream, temporary loss of signal or temporary halting of media decoding due to parental control issues) MAY cause the object to transition from the `presenting` state to the `connecting` state. Temporary loss of resources due to presentation being interrupted by playback of audio from memory MAY cause the object to transition from the `presenting` state to the `connecting` state. Permanent errors (e.g. loss of scarce resources or DRM errors) or calls to `release()` SHALL cause the object to transition to the `unrealized` state regardless of its current state.

Calling the `stop()` method SHALL stop video and audio presentation and cause the video/broadcast object to transition to the `stopped` state. This SHALL have no effect on access to non-media broadcast resources such as EIT information. Calling the `bindToCurrentChannel()` method while in the `stopped` state SHALL result in video and audio presentation being restarted. Calling the `setChannel()`, `nextChannel()` or `prevChannel()` methods while in the `stopped` state shall result in the terminal attempting to select the new service. Applications can use the `playState` property of the video/broadcast object to read its current state.

Applications can use the `playState` property of the `video/broadcast` object to read its current state.

The visibility of a `video/broadcast` object SHALL NOT affect its state or its use of scarce resources. A `video/broadcast` object which is hidden using one of the following techniques:

- the CSS `visibility` or `opacity` properties
- using the CSS `display:none` rule
- removed from the document's DOM (assuming that the document retains at least one other reference to the object)
- obscured by other elements
- positioned off the visible area of the screen

SHALL still be decoding video if it is in the presenting state and any audio associated with the currently presented channel will still be audible. State transitions caused by calls to methods on the `video/broadcast` object, or due to permanent or transient errors, will occur as shown above regardless of the visibility of the object.

When a video/broadcast object is destroyed (e.g. by the video/broadcast object being garbage collected), or when the `release()` method is called, control of broadcast video shall be returned to the terminal. If an application has modified the set of components being presented (e.g. changing the audio or subtitle stream being presented) then the same set of components will continue to be presented.

When a video/broadcast object is destroyed due to a page transition within an application, terminals may delay this operation until the new page is fully loaded in order to avoid display glitches if a video/broadcast object is also present in the new page. Presentation of broadcast video or audio SHALL NOT be interrupted in either case.

7.13.1.2 Properties

Integer **width**
The width of the area used for rendering the video object. This property is only writable if property `fullScreen` has value `false`. Changing the `width` property corresponds to changing the `width` property through the `HTMLObjectElement` interface, and must have the same effect as changing the width through the DOM Level 2 Style interfaces (i.e. `CSS2Properties` interface `style.width`), at least for values specified in pixels.

Integer **height**
The height of the area used for rendering the video object. This property is only writable if property `fullScreen` has value `false`. Changing the `height` property corresponds to changing the `height` property through the `HTMLObjectElement` interface, and must have the same effect as changing the height through the DOM Level 2 Style interfaces (i.e. `CSS2Properties` interface `style.height`), at least for values specified in pixels

readonly Boolean **fullScreen**
Returns `true` if this video object is in full-screen mode, `false` otherwise. The default value is `false`.

function **onChannelChangeError**(Channel channel, Number errorState)
The function that is called when a request to switch a tuner to another channel resulted in an error preventing the broadcasted content from being rendered. This function may be called either in response to a channel change initiated by the application, or a channel change initiated by the OITF (see section 7.13.1.1). The specified function is called with the arguments `channel` and `errorState`. These arguments are defined as follows: • `Channel channel` – the `Channel` object to which a channel switch was requested, but for which the error occurred. • `Number errorState` – error code detailing the type of error:

Value	Description
0	channel not supported by tuner.
1	cannot tune to given transport stream (e.g. no signal)
2	tuner locked by other object.
3	parental lock on channel.
4	encrypted channel, key/module missing.
5	unknown channel (e.g. can't resolve DVB or ISDB triplet).

6	channel switch interrupted (e.g. because another channel switch was activated before the previous one completed).
7	channel cannot be changed, because it is currently being recorded.
8	cannot resolve URI of referenced IP channel.
9	insufficient bandwidth.
10	channel cannot be changed by `nextChannel()/prevChannel()` methods because the OITF does not maintain a favourites or channel list.
11	insufficient resources are available to present the given channel (e.g. a lack of available codec resources).
12	specified channel not found in transport stream.
100	unidentified error.

`Integer` **`playState`**

The current play state of the `video/broadcast` object. Valid values are:

Value	Description
0	unrealized; the user (or application) has not made a request to start presenting a channel or has stopped presenting a channel and released any resources.
1	connecting; the receiver is connecting to the media source in order to begin playback. Objects in this state may be buffering data in order to start playback.
2	presenting; the media is currently being presented to the user. The object is in this state regardless of whether the media is playing at normal speed, paused, or playing in a trick mode (e.g. at a speed other than normal speed).
3	stopped; the terminal is not presenting media, either inside the video/broadcast object or in the logical video plane. The logical video plane is disabled. Control of media presentation is under the control of the application, as defined in Section 8.4

See section 7.13.1.1 for a description of the state model for a `video/broadcast` object.

`function` **`onPlayStateChange`**`( Number state, Number error )`

The function that is called when the play state of the `video/broadcast` object changes. This function may be called either in response to an initiated by the application, an action initiated by the OITF or an error (see section 7.13.1.1).

The specified function is called with the arguments `state` and `error`. These arguments are defined as follows:

- `Number state` – the new state of the `video/broadcast` object

Value	Description
0	unrealized; the user (or application) has not made a request to start presenting a channel or has stopped presenting a channel and released any resources. The content of the video/broadcast object is transparent. Control of media presentation is under the control of the OITF, as defined in section H.2.
1	connecting; the receiver is connecting to the media source in order to begin presenting. Objects in this state may be buffering data in order to start playback. Control of media presentation is under the control of the application, as defined in section H.2. The content of the video/broadcast object is transparent.
2	presenting; the media is currently being presented to the user. The object is in this state regardless of whether the media is playing at normal speed, paused, or playing in a trick mode (e.g. at a speed other than normal speed). Control of media presentation is under the control of the application, as defined in section H.2. The video/broadcast object contains the video being presented.

`Number error` – if the state has changed due to an error, this field contains an error code detailing the type of error. See the definition of `onChannelChangeError` above for valid values. If no error has occurred, this argument SHALL take the value `undefined`.

`function` **`onChannelChangeSucceeded`**`( Channel channel )`

The function that is called when a request to switch a tuner to another channel has successfully completed. This function may be called either in response to a channel change initiated by the application, or a channel change initiated by the OITF (see section 7.13.1.1).

The specified function is called with argument `channel`, which is defined as follows:

- `Channel channel` – the channel to which the tuner switched.

`function` **`onFullScreenChange`**`()`

The function that is called when the value of `fullScreen` changes. The default value is `null`.

The specified function is called with no arguments.

`function` **`onfocus`**`()`

The function that is called when the video object gains focus. The specified function is called with no arguments.

`function` **`onblur`**`()`

The function that is called when the video object loses focus. The specified function is called with no arguments.

7.13.1.3 Methods

ChannelConfig **getChannelConfig**()	
Description	Returns the channel line-up of the tuner in the form of a ChannelConfig object as defined in Section 7.13.8. The method SHALL return the value null if the channel list is not (partially) managed by the OITF (i.e., if the channel list information is managed entirely in the network).

Channel **bindToCurrentChannel**(Channel channel)		
Description	If a broadcast channel is being presented under the control of the OITF (i.e. the video was being presented by the OITF before the application started) then move the control of the broadcast channel presentation to the application and present the currently playing broadcast channel in the video/broadcast object. If the broadcast channel that is being presented is already under the control by an application instead of the OITF (either by the current application or by another running application) it cannot be bound using bindToCurrentChannel(). If video from exactly one channel is currently being presented by the OITF then this binds the video/broadcast object to that video. If video from more than one channel is currently being presented by the OITF then this binds the video/broadcast object to the channel whose audio is being presented. If there is no channel currently being presented, or binding to the necessary resources to play the channel through the video/broadcast object fails for whichever reason, the OITF SHALL dispatch an event to the onPlayStateChange listener(s) whereby the state parameter is given value 0 ("unrealized") and the error parameter is given the appropriate error code. Calling this method from any other state than the unrealized state SHALL have no effect. This method returns a Channel object representing the channel being presented, or null if no video is currently being presented under the control of the terminal. See state diagram in Section 7.13.1.1 for more information of its usage.	
Arguments	*channel*	Optional argument indicating the channel to be presented in this video/broadcast object. If not specified, then this is equivalent to calling setChannel() with the first entry in the bindableChannels collection (see section 7.13.7).

Channel **createChannelObject**(Integer idType, String dsd, Integer sid)	
Description	Creates a Channel object of the specified idType. This method is typically used to create a Channel object of type ID_DVB_SI_DIRECT. The Channel object can subsequently be used by the setChannel() method to switch a tuner to a channel that is not part of the channel list which was conveyed by the OITF to the server. The scope of the resulting Channel object is limited to the Javascript environment (incl. video/broadcast object) to which the Channel object is returned, i.e. it does not get added to the channellist available through method getChannelConfig(). Valid value for idType include: ID_DVB_SI_DIRECT. For other values this behaviour is not specified.

	If the channel of the given type cannot be created or the delivery system descriptor is not valid, the method SHALL return `null`. If the channel of the given type can be created and the delivery system descriptor is valid, the method SHALL return a `Channel` object whereby at a minimum the properties with the same names (i.e. `idType`, `dsd` and `sid`) are given the same value as argument `idType`, `dsd` and `sid` of the `createChannelObject` method. Whilst tuning to the given channel (i.e. using the `setChannel()` method), the OITF SHOULD fill in the values detected for properties `onid`, `tsid` and `sid`, even if an error is detected.	
Arguments	*idType*	The type of channel, as indicated by one of the `ID_*` constants defined in Section 7.13.12.1. Valid value for `idType` include: `ID_DVB_SI_DIRECT`. For other values this behaviour is not specified.
	dsd	The delivery system descriptor (tuning parameters) represented as a string whose characters shall be restricted to the ISO Latin-1 character set. Each character in the dsd represents a byte of a delivery system descriptor as defined by DVB-SI [EN 300 468] section 6.2.13, such that a byte at position "i" in the delivery system descriptor is equal the Latin-1 character code of the character at position "i" in the dsd.
	sid	The service ID.

`Channel createChannelObject( Integer idType, Integer onid, Integer tsid, Integer sid, Integer sourceID, String ipBroadcastID )`		
Description	Creates a `Channel` object of the specified `idType`. The Channel object can subsequently be used by the `setChannel` method to switch a tuner to a channel that is not part of the channel list which was conveyed by the OITF to the server. The scope of the resulting `Channel` object is limited to the Javascript environment (incl. `video/broadcast` object) to which the `Channel` object is returned, i.e. it does not get added to the channellist available through method `getChannelConfig`. If the channel of the given `idType` cannot be created or the given (combination of) arguments are not considered valid or complete, the method SHALL return `null`. If the channel of the given type can be created and arguments are considered valid and complete, the method SHALL return a `Channel` object whereby at a minimum the properties with the same names are given the same value as the given arguments of the `createChannelObject` method. The values specified for the remaining properties of the `Channel` object are set to `undefined`.	
Arguments	*idType*	The type of channel, as indicated by one of the ID_* constants defined in Section 7.13.12.1.
	onid	The original network ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type ID_DVB_* or ID_ISDB_*.
	tsid	The transport stream ID. Optional argument that MAY be specified when the `idType` specifies a channel of type `ID_DVB_*` or `ID_ISDB_*`.
	sid	The service ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_DVB_*` or `ID_ISDB_*`.
	sourceID	The source_ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_ATSC_T`.

	ipBroadcastID	The DVB textual service identifier of the IP broadcast service, specified in the format "`ServiceName.DomainName`", or the URI of the IP broadcast service. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_IPTV_SDS` or `ID_IPTV_URI`.

`void setChannel( Channel channel, Boolean trickplay, String contentAccessDescriptorURL )`	
Description	Requests the OITF to switch a (logical or physical) tuner to the channel specified by `channel` and render the received broadcast content in the area of the browser allocated for the `video/broadcast` object. If the `channel` specifies a value for ccid, and the value is not known by the OITF, the OITF SHALL ignore the request to switch channel and trigger the function specified by the `onChannelChangeError` property, specifying the value 5 ("unknown channel") for the `errorState`, and dispatch the corresponding DOM 2 event (see below). If the `channel` specifies an `idType` attribute value which is not supported by the OITF or a combination of properties that does not identify a valid channel, the OITF SHALL ignore the request to switch channel and trigger the function specified by the `onChannelChangeError` property, specifying the value 0 ("Channel not supported by tuner") for the `errorState`, and dispatch the corresponding DOM 2 event (see below). If the `channel` specifies an `idType` attribute value supported by the OITF, and the combination of properties defines a valid channel, the OITF SHALL relay the channel switch request to a local physical tuner that is currently not in use by another `video/broadcast` object and that can tune to the specified channel. If no tuner satisfying these requirements is available (i.e. all physical tuners that could receive the specified channel are in use), the OITF SHALL ignore the request and trigger the function specified by the `onChannelChangeError` property, specifying the value 2 ("tuner locked by other object") for the `errorState` and dispatch the corresponding DOM 2 event (see below). If multiple tuners satisfying these requirements are available, the OITF selects one. If the `channel` specifies an IP broadcast channel, and the OITF supports `idType` `ID_IPTV_SDS` or `ID_IPTV_URI`, the OITF SHALL relay the channel switch request to a logical 'tuner' that can resolve the URI of the referenced IP broadcast channel. If no logical tuner can resolve the URI of the referenced IP broadcast channel, the OITF SHALL ignore the channel switch request and SHOULD trigger the function specified by the `onChannelChangeError` property, specifying the value 8 ("cannot resolve URI of referenced IP channel") for the `errorState`, and dispatch the corresponding DOM 2 event. The optional attribute `contentAccessDescriptorURL` allows for the inclusion of a Content Access Streaming Descriptor (the format of which is defined in Annex E.2) to provide additional information for dealing with IPTV broadcasts that are (partially) DRM-protected. The descriptor may for example include Marlin action tokens or a previewLicense. The attribute SHALL be `undefined` or `null` if it is not applicable. If the attribute `contentAccessDescriptorURL` is present, the trickplay attribute shall take a value of either true or false. If, following this procedure, the OITF selects a tuner that was not already being used to display video inside the `video/broadcast` object, the OITF SHALL claim the selected tuner and the associated resources (e.g., decoding and rendering resources) on behalf of the `video/broadcast` object. The OITF SHALL visualize the video content received over the tuner in the area of the browser allocated for the `video/broadcast` object. If the OITF cannot visualize the video

	content following a successful tuner switch (e.g., because the channel is under parental lock), the OITF SHALL trigger the function specified by the `onChannelChangeError` property with the appropriate `channel` and `errorState` value, and dispatch a corresponding DOM 2 event (see below). If successful, the OITF SHALL trigger the function specified by the `onChannelChangeSucceeded` property with the given `channel` value, and also dispatch a corresponding DOM 2 event.	
Arguments	*channel*	The channel to which a switched is requested. If the `channel` object specifies a `ccid`, the `ccid` identifies the channel to be set. If the `channel` does not specify a `ccid`, the `idType` determines which properties of the channel are used to define the channel to be set, for example, if the channel is of type `ID_IPTV_SDS` or `ID_IPTV_URI`, the `ipBroadcastID` identifies the channel to be set.
	trickplay	Optional flag indicating whether resources SHOULD be allocated to support trick play. This argument provides a hint to the receiver in order that it may allocate appropriate resources. Failure to allocate appropriate resources, due to a resource conflict, a lack of trickplay support, or due to the OITF ignoring this hint, SHALL have no effect on the success or failure of this method. If trickplay is not supported, this SHALL be indicated through the failure of later calls to methods invoking trickplay functionality. The `timeShiftMode` property defined in section 7.13.2.2 shall provide information as to type of trickplay resources that should be allocated. If argument `contentAccessDescriptorURL` is included then the `trickplay` argument SHALL be included.
	contentAccessDescriptorURL	Optional argument containing a Content Access Streaming descriptor (the format of which is defined in Annex E.2) that can be included to provide additional information for dealing with IPTV broadcasts that are (partially) DRM-protected. The argument SHALL be `undefined` or `null` if it is not applicable.

`void` **`prevChannel`**`()`	
Description	Requests the OITF to switch the tuner that is currently in use by the `video/broadcast` object to the channel that precedes the current channel in the active favourite list, or, if no favourite list is currently selected, to the previous channel in the channel list. If it has reached the start of the favourite/channel list, it SHALL cycle to the last channel in the list. If the current channel is not part of the channel list, the result of calling this method is implementation dependent. If the previous favourite channel is a non-IP channel that cannot be received over the tuner currently used by the `video/broadcast` object, the OITF SHALL relay the channel switch request to a local physical tuner that is not in use and that can tune to the specified channel. The behaviour is defined in more detail in the description of the `setChannel` method. If an error occurs during switching to the previous channel, the OITF SHALL trigger the

	function specified by the onChannelChangeError property with the appropriate channel and errorState value, and dispatch the corresponding DOM 2 Event (see below). If the OITF does not maintain the channel list and favourite list by itself, the OITF SHALL trigger the onChannelChangeError function with the channel property having the value null, and errorState=10 ("channel cannot be changed by nextChannel()/prevChannel() methods"). If successful, the OITF SHALL trigger the function specified by the onChannelChangeSucceeded property with the appropriate channel value, and also dispatch the corresponding DOM 2 event.

void **nextChannel**()	
Description	Requests the OITF to switch the tuner that is currently in use by the video/broadcast object to the channel that succeeds the current channel in the active favourites list, or, if no favourite list is currently selected, to the next channel in the channel list. If it has reached the end of the favourite/channel list, it SHALL cycle to the first channel in the list. If the current channel is not part of the channel list, the result of calling this method is implementation dependent. If the next favourite channel is a non-IP channel that cannot be received over the tuner currently used by the video/broadcast object, the OITF SHALL relay the channel switch request to a local physical tuner that is not in use and that can tune to the specified channel. The behaviour is defined in more detail in the description of the setChannel method. If an error occurs during switching to the next channel, the OITF SHALL trigger the function specified by the onChannelChangeError property with the appropriate channel and errorState value, and dispatch the corresponding DOM 2 event (see below). If the OITF does not maintain the channel list and favourite list by itself, the OITF SHALL trigger the onChannelChangeError function with the channel property having the value null, and errorState=10 ("channel cannot be changed by nextChannel()/prevChannel() methods"). If successful, the OITF SHALL trigger the function specified by the onChannelChangeSucceeded property with the appropriate channel value, and also dispatch the corresponding DOM 2 event.

void **stop**()	
Description	Stop presenting broadcast video. If the video/broadcast object is in any state other than the unrealized state, it SHALL transition to the stopped state. and stop video and audio presentation. This SHALL have no effect on access to non-media broadcast resources such as EIT information. Calling this method from the unrealized state SHALL have no effect. See figure 11 in section 7.13.1.1 for more information of its usage.

void **setFullScreen**(Boolean fullscreen)

Description	Sets the rendering of the video content to full-screen (`fullscreen = true`) or windowed (`fullscreen = false`) mode (as per [Req. 5.7.4.f] of [CEA-2014-A]). If this indicates a change in mode, this SHALL result in a change of the value of property `fullScreen`. Changing the mode SHALL NOT affect the z-index of the video object.	
Arguments	*fullScreen*	Boolean to indicate whether video content SHOULD be rendered full-screen or not.

`Boolean` **`setVolume`**`( Integer volume )`		
Description	Adjusts the volume of the currently playing media to the volume as indicated by volume. Allowed values for the volume argument are all the integer values starting with 0 up to and including 100. A value of 0 means the sound will be muted. A value of 100 means that the volume will become equal to current “master” volume of the device, whereby the “master” volume of the device is the volume currently set for the main audio output mixer of the device. All values between 0 and 100 define a linear increase of the volume as a percentage of the current master volume, whereby the OITF SHALL map it to the closest volume level supported by the platform. The method returns `true` if the volume has changed. Returns `false` if the volume has not changed. Applications MAY use the `getVolume()` method to retrieve the actual volume set.	
Arguments	*volume*	Integer value between 0 up to and including 100 to indicate volume level.

`Integer` **`getVolume`**`()`	
Description	Returns the actual volume level set; for systems that do not support individual volume control of players, this method will have no effect and will always return 100.

`void` **`release`**`()`	
Description	Releases the decoder/tuner used for displaying the video broadcast inside the `video/broadcast` object, stopping any form of visualization of the video inside the `video/broadcast` object and releasing any other associated resources.

7.13.1.4 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onfocus`	`focus` (as specified in Section 1.6.5 of [DOM 2 Events])	Bubbles: No Cancelable: No Context Info: None

onblur	blur (as specified in Section 1.6.5 of [DOM 2 Events])	Bubbles: No Cancelable: No Context Info: None
onFullScreenChange	FullScreenChange	Bubbles: No Cancelable: No Context Info: None
onChannelChangeError	ChannelChangeError	Bubbles: No Cancelable: No Context Info: channel, errorState
onChannelChangeSucceeded	ChannelChangeSucceeded	Bubbles: No Cancelable: No Context Info: channel
onPlayStateChange	PlayStateChange	Bubbles: No Cancelable: No Context Info: state, error

Note: these DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the addEventListener() method on the video/broadcast object itself. The third parameter of addEventListener, i.e. "useCapture", will be ignored.

7.13.1.5 Styling

The OITF SHALL support the CSS properties (which MAY be changed using the DOM Level 2 Style module) for embedded video/broadcast objects: width, height, position, float, top, left, right, bottom, vertical-align, padding and padding-* properties, margin and margin-* properties, border and border-* properties, visibility, and display.

If the value of the <overlaylocaltuner> element in the capability description of the OITF is not set to none, then the OITF SHALL support overlays as defined by bullet p) of [Req. 5.2.1.a] of CEA-2014-A for broadcasts coming from the local tuner that are displayed using the video/broadcast embedded object. In this case, broadcast video objects SHALL support CSS-property z-index, in both full-screen and windowed mode. Moreover, the OITF SHALL support the CSS opacity property and CSS3 RGBA color values, for any non-video XHTML element on top of a video object. If the value of the <overlaylocaltuner> element in the capability description of the OITF is set to none, no objects SHALL overlay the video, i.e. the value of z-index for video is ignored.

If the value of the <overlayIPbroadcast> element in the capability description of the OITF is not set to none, then the OITF SHALL support overlays as defined by bullet p) of [Req. 5.2.1.a] of CEA-2014-A for IP broadcasts that are displayed using the video/broadcast embedded object. In this case, broadcast video objects SHALL support CSS-property z-index, in both full-screen and windowed mode. Moreover, the OITF SHALL support the CSS opacity property and CSS3 RGBA color values, for any non-video XHTML element on top of a video object. If the value of the <overlayIPbroadcast> element in the capability description of the OITF is set to none, no objects SHALL overlay the video, i.e. the value of z-index for video is ignored.

7.13.2 Extensions to video/broadcast for recording and time-shift

If an OITF has indicated support for recording functionality (i.e. by giving value `true` to element `<recording>` as specified in Section 9.3.3 in its capability description), the OITF SHALL support the following additional constants, properties and methods on the `video/broadcast` object, in order to start a recording and/or time-shift of a current broadcast.

Note that this functionality is subject to the security model as specified in Section 10.1.

This functionality is subject to the state transitions represented in the following state diagram:

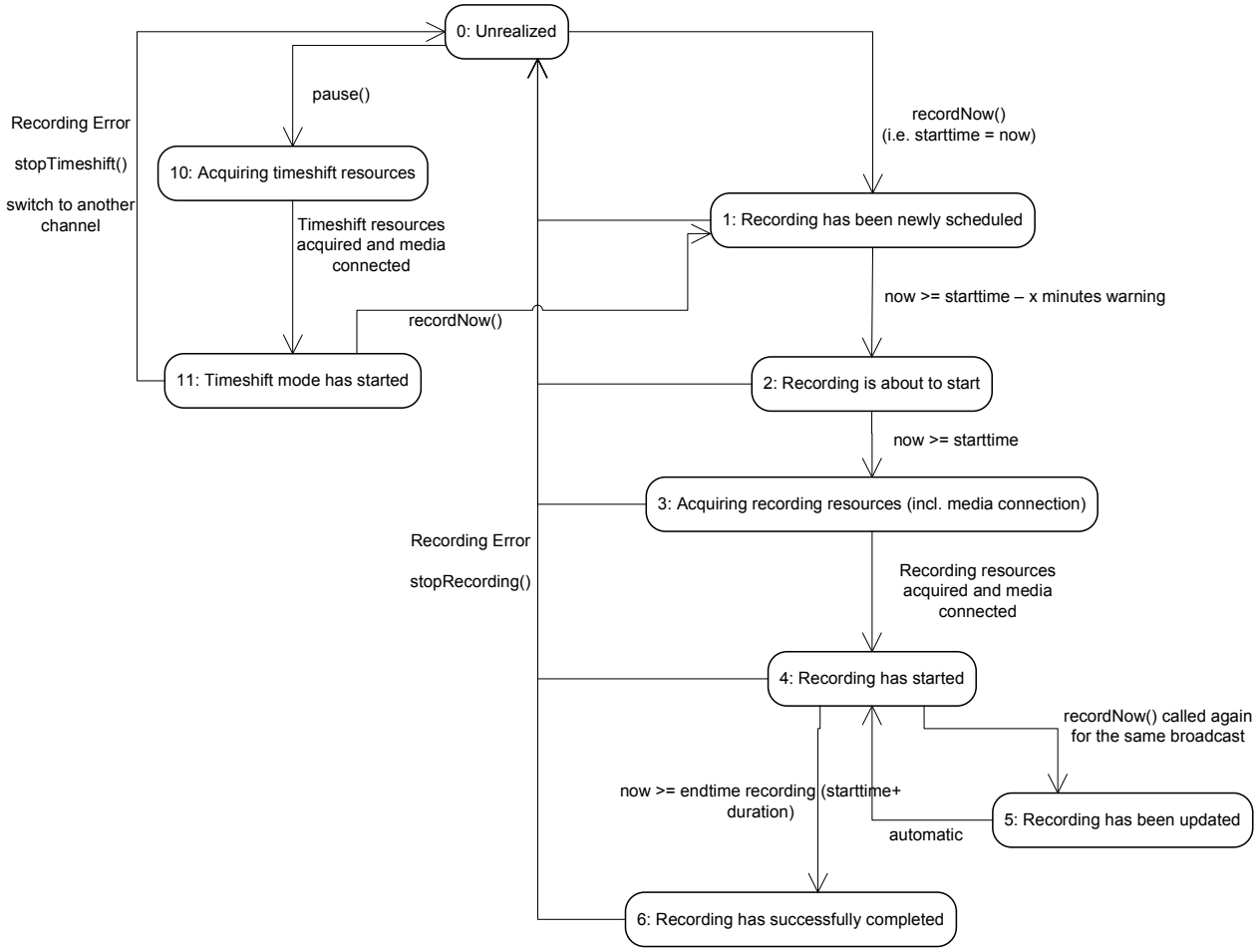


Figure 16: PVR States for `recordNow` and timeshifting using `video/broadcast`

Note that when the user switches to another channel whilst the current channel is being recorded using `recordNow` or the `video/broadcast` object gets destroyed, the conflict resolution and the release of resources is implementation dependent. The OITF MAY report a recording error using a `RecordingEvent` with value 0 (“Unrealized”) for argument `state` and with value 2 (“Tuner conflict”) for argument `error` in that case.

7.13.2.1 Additional constants for video/broadcast object

Name	Value	Use
POSITION_START	0	Indicates a playback position relative to the start of the buffered content.
POSITION_CURRENT	1	Indicates a playback position relative to the current playback position.
POSITION_END	2	Indicates a playback position relative to the end of the buffered content.

7.13.2.2 Additional properties for video/broadcast object

```
function onPlaySpeedChanged( Number speed )
```

The function that is called when the playback speed of a channel changes.

The specified function is called with one argument, `speed`, which is defined as follows:

- `Number speed` – the playback speed of the media at the time the event was dispatched.

If the playback reaches the beginning of the time-shift buffer at rewind playback speed, then the play state is changed to 2 ('paused') and a `PlaySpeedChanged` event with a speed of 0 is generated. If the playback reaches the end of the time-shift buffer at fast-forward playback speed, then the play speed is set to 1.0 and a `PlaySpeedChanged` event is generated.

```
function onPlayPositionChanged( Integer position )
```

The function that is called when change occurs in the play position of a channel due to the use of trick play functions.

The specified function is called with one argument, `position`, which is defined as follows:

- `Integer position` – the playback position of the media at the time the event was dispatched, measured from the start of the timeshift buffer. If the value of the `currentTimeShiftMode` property is 1, this is measured in milliseconds from the start of the timeshift buffer. If the value of the `currentTimeShiftMode` property is 2, this is measured in milliseconds from the start of the media item. If the play position cannot be determined, this argument takes the value `undefined`.

```
readonly Integer playbackOffset
```

Returns the playback position, specified as the positive offset of the live broadcast in seconds, in the currently rendered (timeshifted) broadcast.

When the `currentTimeShiftMode` property has the value 1, the value of this property is `undefined`.

```
readonly Integer maxOffset
```

Returns the maximum playback offset, in seconds of the live broadcast, which is supported for the currently rendered (timeshifted) broadcast. If the maximum offset is unknown, the value of this property SHALL be `undefined`.

When the `currentTimeShiftMode` property has the value 1, the value of this property is `undefined`.

`readonly Integer` **`recordingState`**

Returns the state of the OITF's timeshift and `recordNow` functionality for the channel shown in the `video/broadcast` object. One of:

Value	Description
0	Unrealized: user/application has not requested timeshift or `recordNow` functionality for the channel shown. No timeshift or recording resources are claimed in this state.
1	Recording has been newly scheduled.
2	Recording is about to start .
3	Acquiring recording resources (incl. media connection).
4	Recording has started.
5	Recording has been updated.
6	Recording has successfully completed.
10	Acquiring timeshift resources (incl. media connection).
11	Timeshift mode has started.

When the `currentTimeShiftMode` property has the value 1, the value of this property is `undefined`.

`function` **`onRecordingEvent`**`(Integer state, Integer error, String recordingId )`

This function is the DOM 0 event handler for notification of state changes of the recording functionality.

The specified function is called with the following arguments:

- `Integer state` - The current state of the recording. One of:

Value	Description
0	Unrealized: user/application has not requested timeshift or recordNow functionality for the channel shown. No timeshift or recording resources are claimed in this state.
1	Recording has been newly scheduled.
2	Recording is about to start .
3	Acquiring recording resources (including media connection).
4	Recording has started.

5	Recording has been updated.
6	Recording has successfully completed.
10	Acquiring timeshift resources (including media connection).
11	Timeshift mode has started.

- `Integer error` - If the state of the recording has changed due to an error, this field contains an error code detailing the type of error. One of:

Value	Description
0	The recording sub-system is unable to record due to resource limitations.
1	There is insufficient storage space available. (Some of the recording may be available).
2	Tuner conflict (e.g. due to conflicting scheduled recording).
3	Recording not allowed due to DRM restrictions.
4	Recording has stopped before completion due to unknown (probably hardware) failure.
10	Timeshift not possible due to resource limitations.
11	Timeshift not allowed due to DRM restrictions.
12	Timeshift ended due to unknown failure.

  If no error has occurred, this argument SHALL take the value `undefined`.

- `String recordingId` - The identifier of the recording to which this event refers.

`readonly Integer` **`playPosition`**

If the value of the `currentTimeShiftMode` property is 1, the current playback position of the media, measured in milliseconds from the start of the timeshift buffer.

If the value of the `currentTimeShiftMode` property is 2, the current playback position of the media, measured in milliseconds from the start of the media item.

`readonly Number` **`playSpeed`**

The current play speed of the media.

`readonly Number playSpeeds[ ]`

Returns the ordered list of playback speeds, expressed as values relative to the normal playback speed (1.0), at which the currently specified A/V content can be played (as a time-shifted broadcast in the `video/broadcast` object), or `undefined` if the supported playback speeds are not known or the `video/broadcast` object is not in timeshift mode..

If the `video/broadcast` object is in timeshift mode, the `playSpeeds` array SHALL always include at least values 1.0 and 0.0.

`function onplaySpeedsArrayChanged()`

The function that is called when the `playSpeeds` array values have changed. An application that makes use of the `playSpeeds` array needs to read the values of the `playSpeeds` property again.

`Integer timeShiftMode`

The time shift mode indicates the mode of operation for support of timeshift playback in the video/broadcast object. Valid values are:

Value	Description
0	Timeshift is turned off.
1	Timeshift shall use “local resource”
2	Timeshift shall use “network resources”.
3	Timeshift shall first use “local resource” when available and fallback to “network resources”.

If property is not set the default value of the property is according to `preferredTimeShiftMode` in section 7.3.2.1.

`readonly Integer currentTimeShiftMode`

When timeshift is in operation the property indicates which resources are currently being used. Valid values are:

Value	Description
0	No timeshift
1	Timeshift using “local resource”
2	Timeshift using “network resources”

7.13.2.3 Additional methods for video/broadcast object

String **recordNow**(Integer duration)		
Description	Starts recording the broadcast currently rendered in the `video/broadcast` object. If the OITF has buffered the broadcasted content, the recording starts from the current playback position in the buffer, otherwise start recording the broadcast stream as soon as possible after the recording resources have been acquired. The specified duration is used by the OITF to determine the minimum duration of the recording in seconds from the current starting point. If `recordNow()` is called while the broadcast that is currently rendered in the video/broadcast object is already being recorded, the total duration of this ongoing recording is extended by the value of the `duration` argument (i.e. the value of the `duration` argument is added onto the remaining recording time).The success or failure and the current state of the recording can be tracked using the `onRecordingEvent` intrinsic event handler as defined in Section 7.13.2.2 or by registering for the respective DOM 2 `RecordingEvent` as defined in Section 1.1.1. The method returns a `String` value representing a unique identifier to identify the recording. If the OITF provides recording management functionality through the APIs defined in section 7.10.4, this SHALL be the value of the `id` property of the associated `Recording` object defined in section 7.10.5.1. The OITF SHALL guarantee that recording identifiers are unique in relation to download identifiers and `CODAsset` identifiers. The method returns `undefined` if the given argument is not accepted to trigger a recording. If the OITF supports metadata processing in the terminal, the fields of the resulting `Recording` object MAY be populated using metadata retrieved by the terminal. Otherwise, the values of these fields SHALL be implementation-dependent	
Arguments	*duration*	The minimum duration of the recording in seconds. A value of -1 indicates that the recording SHOULD continue until `stopRecording()` is called, storage space is exhausted, or an error occurs. In this case it is essential that `stopRecording()` is called later.

void **stopRecording**()	
Description	Stops the current recording started by `recordNow`.

Boolean **pause**()	
Description	Pause playback of the broadcast. This operation may be asynchronous, and presentation of the video may not pause until after this method returns. For this reason, a `PlaySpeedChanged` event will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play speed. The action taken depends on the value of the `timeShiftMode` property.

	If the value of the `timeShiftMode` property is 0, if trick play is not supported for the channel currently being rendered, or if the current time shift mode is not supported for the type of channel being presented (e.g. attempting to use network resource to time shift a DVB or analogue channel) this method shall return false. If the timeshift mode is set to 1 or 3 (local resources) and if recording has not yet been started, this method will start recording the broadcast that is currently being rendered live (i.e., not time-shifted) in the video/broadcast object and return true. If the OITF has buffered the 'live' broadcasted content, the recording starts with the content that is currently being rendering in the video/broadcast object. Since this operation may be asynchronous if the recording started successfully, the rendering of the broadcasted content is paused, i.e. a still-image video frame is shown, and a `PlaySpeedChanged` event is generated. If the timeshift mode is set to 2 (network resources) then the OITF shall follow the procedures defined in section 8.2.3.2.3 and returns true. Since this operation is asynchronous when the procedure are executed successful the rendering of the broadcasted content is paused, i.e. a still-image video frame is shown, and `PlaySpeedChanged` event is generated. If the specified timeshift mode is not supported, this method shall return false. Otherwise, this method shall return true. Acquiring the necessary resources to enable the specified timeshift mode may be an asynchronous operation; applications may receive updates of this process by registering a listener for `RecordingEvents` as defined in section 1.1.1.

`Boolean` **`resume`**`()`	
Description	Resumes playback of the time-shifted broadcast channel that is currently being rendered in the video/broadcast object at the speed specified by `setSpeed()`. If the desired speed was not set via `setSpeed()`, playback is resumed at normal speed (i.e. speed 1.0). This operation may be asynchronous, and presentation of the video may not resume until after this method returns. For this reason, a `PlaySpeedChanged` event will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play speed. If playback is not currently paused, the OITF shall ignore the request to start playback and shall return `false`. The action taken depends on the value of the `timeShiftMode` property. If the value of the `timeShiftMode` property is 1 or 3 (local resources) then the OITF shall resume playback of the broadcast channel as specified above and return `true`. If the value of the `timeShiftMode` property is 2 (network resources) then the OITF shall follow the procedures defined in section 8.2.3.2.3 and return `true`. Since this operation is asynchronous when the procedure is successfully executed a `PlaySpeedChanged` event is generated with current speed. After initial operation of `resume()` several events may affect the operation. If during fast forward the end of stream is reached the playback SHALL resume at normal speed and a `PlaySpeedChanged` event is generated. If the end of stream is reached due to end of content the playback will automatically be paused and a `PlaySpeedChanged` event is generated. Any resources used for time-shifting SHALL

	NOT be discarded. If during rewinding the playback reaches the point that it cannot be rewound further, playback will automatically be paused (i.e. the play speed will be changed to 0) and a `PlaySpeedChanged` event is generated. If for any of these events timeShiftMode is set to 3 and local resources are not available anymore then network sources SHALL be used according to the procedures defined in section 8.2.3.2.3. The OITF SHALL perform a smooth transition of the stream between local and network resources.

`Boolean` **`setSpeed`**`( Number speed )`	
Description	Sets the playback speed of the time-shifted broadcast to the value speed. If the time-shifted broadcast cannot be played at the desired speed (specified as a value relative to the normal playback speed), the playback speed will be set to the best approximation of speed. Applications are not required to pause playback of the broadcast or take any other action before calling `setSpeed()`. Setting a speed of 0 SHALL have the same effect as calling `pause()`. If playback is paused, setting the speed SHALL NOT cause playback to resume. If the `video/broadcast` object is currently not rendering a time-shifted channel, the OITF shall ignore the request to change the playback speed and shall return `false`, otherwise `true` is returned. This operation may be asynchronous, and presentation of the video may not be affected until after this method returns. For this reason, a `PlaySpeedChanged` event will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play speed. The action taken depends on the value of the `timeShiftMode` property. If the value of the `timeShiftMode` property is 1 or 3 (local resources) then the `setSpeed()` method sets the playback speed of the time-shifted broadcast to the value speed. If the `timeShiftMode` is set to 2 (network resources) the OITF shall follow the procedures defined in section 8.2.3.2.3 and return true. Since this operation is asynchronous when the procedure is successfully executed `PlaySpeedChanged` event is generated with the new speed. After initial operation of `setSpeed()` several events may affect the operation. If during fast forward the end of stream is reached the playback SHALL resume at normal speed and a `PlaySpeedChanged` event is generated. If the end of stream is reached due to end of content the playback will automatically be paused and a `PlaySpeedChanged` event is generated. Any resources used for time-shifting SHALL NOT be discarded. If during rewinding the playback reaches the point that it cannot be rewound further, playback will automatically be paused (i.e. the play speed will be changed to 0) and a `PlaySpeedChanged` event is generated. If for any of these events if `timeShiftMode` is set to 3 and local resources are not available anymore then network sources SHALL be used according to the procedures defined in section 8.2.3.2.3. The OITF SHALL perform a smooth transition of the

	stream between local and network resources.	
Arguments	*speed*	The desired relative playback speed, specified as a float value relative to the normal playback speed of 1.0. A negative value indicates reverse playback. If the time-shifted broadcast cannot be played at the desired speed, the playback speed will be set to the best approximation.

`Boolean` **`seek`**`( Integer offset, Integer reference )`		
Description	Sets the playback position of the time-shifted broadcast that is being rendered in the video/broadcast object to the position specified by the offset and the reference point as specified by one of the constants defined in Section 7.13.2.1. Returns `true` if the playback position is a valid position to seek to, `false` otherwise. Applications are not required to pause playback of the broadcast or take any other action before calling `seek()`. This operation may be asynchronous, and presentation of the video may not be affected until after this method returns. For this reason, a `PlayPositionChanged` event will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play position. If the `video/broadcast` object is currently not rendering a time-shifted channel or if the position falls outside the time-shift buffer, the OITF shall ignore the request to seek and shall return the value false. The action taken depends on the value of the `timeShiftMode` property. If the `timeShiftMode` is set to 1 (local resources) the `seek()` method sets the playback position of the time-shifted broadcast that is being rendered in the video/broadcast object as defined above. Playback of live content is resumed if the new position equals the end of the time-shift buffer. If the `timeShiftMode` is set to 2 (network resources) the OITF shall follow the procedures defined in section 8.2.3.2.3 and return true. Since this operation is asynchronous when the procedure is successfully executed `PlayPositionChanged` event is generated with the new position. Note that if `timeShiftMode` is set to 3 then local resources are used over network resources. After initial operation of `seek()` several events may affect the operation. If during fastforward the end of stream is reached the playback SHALL resume at normal speed and a `PlaySpeedChanged` event is generated. If the end of stream is reached due to end of content the playback will automatically be paused and a `PlaySpeedChanged` event is generated. Any resources used for time-shifting SHALL NOT be discarded. If for any of these events if `timeShiftMode` is set to 3 and local resources are not available anymore then network sources SHALL be used according to the procedures defined in section 8.2.3.2.3. The OITF SHALL perform a smooth transition of the stream between local and network resources.	
Arguments	*offset*	The offset from the reference position, in seconds. This can be either a positive or negative value.

	reference	The reference point from which the offset SHALL be measured. The reference point can be either `POSITION_CURRENT`, `POSITION_START`, or `POSITION_END`.

`Boolean stopTimeshift()`	
Description	Stops rendering in time-shifted mode of the broadcast channel in the `video/broadcast` object and, if applicable, plays the current broadcast from the live point and stops time-shifting the broadcast. The OITF SHALL release all resources that were used to support time-shifted rendering of the broadcast. This operation SHALL NOT affect recording of a channel if `recordNow()` was used. Returns `true` if the time-shifted broadcast was successfully stopped and resources were released and `false` otherwise. If the `video/broadcast` object is currently not rendering a time-shifted channel, the OITF shall ignore the request to stop the time-shift and shall return the value `false`.

In addition to these methods, the OITF SHALL support an additional optional attribute `offSet` on the `setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)` method of the `video/broadcast` object as defined in Section 7.13.1.3, if the OITF has indicated support for scheduled content over IP by defining one or more `ID_IPTV_*` values as part of the transport attribute of the <video_broadcast> element in the capability description.

`void setChannel( Channel channel, Boolean trickplay, String contentAccessDescriptorURL, Integer offset )`		
Description	Requests the OITF to switch a (logical or physical) tuner to the specified `channel` and render the received broadcast content in the area of the browser allocated for the `video/broadcast` object, as specified by the `setChannel(Channel channel, Boolean trickPlay, String contentAccessDescriptorURL)` method in Section 7.13.1.3. The additional `offSet` attribute optionally specifies the desired offset with respect to the live broadcast in number of seconds from which the OITF SHOULD start playback immediately after the channel switch (whereby `offSet` is given as a positive value for seeking to a time in the past). If an OITF cannot start playback from the desired position, as indicated by the specified `offSet` (e.g. because the OITF did not, or could not, record the specified channel prior to the call to `setChannel`), if the specified `offSet` is '0', or if the `offSet` is not specified, the OITF SHALL start playback from the live position after the specified channel switch.	
Arguments	*channel*	As defined for method `setChannel()` in Section 7.13.1.3.
	trickplay	Optional flag as defined for method `setChannel()` in Section 7.13.1.3.
	contentAccessDescriptorURL	Optional attribute as defined for method `setChannel()` in Section 7.13.1.3.
	offset	The optional offset attribute MAY be used to specify the desired offset with respect to the live broadcast in number of seconds from which the OITF SHOULD start playback immediately after the channel switch (whereby offset is given as a

		positive value for seeking to a time in the past).

7.13.2.4 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onRecordingEvent`	`RecordingEvent` (as specified in Section 1.1.1)	Bubbles: No Cancelable: No Context Info: `state`, `error`, `recordingId`
`onPlaySpeedChanged`	`PlaySpeedChanged`	Bubbles: No Cancelable: No Context Info: `speed`
`onPlayPositionChanged`	`PlayPositionChanged`	Bubbles: No Cancelable: No Context Info: `position`
`onPlaySpeedsArrayChanged`	`PlaySpeedsArrayChanged`	Bubbles: No Cancelable: No Context Info: None

Note: the DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `video/broadcast` object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.13.3 Extensions to video/broadcast for access to EIT p/f

The following properties and events SHALL be added to the video/broadcast embedded object, if the OITF has indicated support for accessing DVB-SI EIT p/f information, by giving the value "`true`" to element `<clientMetadata>` and the value "`eit-pf`" or "`dvb-si`" to the `type` attribute of that element as defined in Section 9.3.7 in their capability profile.

Access to these properties SHALL adhere to the security model in Section 10. The associated permission name is "*permission_metadata*".

```
readonly ProgrammeCollection programmes
```

The collection of programmes available on the currently tuned channel. This list is a `ProgrammeCollection` as defined in Section 7.16.3 and is ordered by start time, so index 0 will always refer to the present programme (if this information is available).

If the type attribute of the `<clientMetadata>` element in the OITF's capability description has the value "`eit-pf`", this list SHALL at least provide `Programme` objects as defined in Section 7.16.2 for the present and the directly following programme on the currently tuned channel, if that information is available. In other words, the DAE application should not expect `programmes.length` to be larger than 2.

If the `video/broadcast` object is not currently tuned to a channel, or if the present/following information has not yet been retrieved (e.g. the object has just tuned to a new channel and present/following information has not yet been broadcast), or if present/following information is not available for the current channel, the length of this collection SHALL be 0.

If the type attribute of the <clientMetadata> element in the OITF's capability description has a value other than "`eit-pf`", an OITF MAY populate this field from other metadata sources described in [OIPF_META2].

The `programmes.length` property SHALL indicate the number of items that are currently known and up to date (i.e. whereby the "`startTime + duration`" is not smaller than the current time). This may be 0 if no programme information is currently known for the currently tuned channel.

In order to prevent misuse of this information, access to this property SHALL adhere to the security model in Section 10. The associated permission name is "*permission_metadata*".

`function` **`onProgrammesChanged`**`()`
The function that is called when the `programmes` property has been updated with new programme information, e.g. when the current broadcast programme is finished and a new one has started. The specified function is called with no arguments.

7.13.3.1 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onProgrammesChanged`	`ProgrammesChanged`	Bubbles: No Cancelable: No Context Info: `None`

7.13.4 Extensions to video/broadcast for playback of selected components

To support the selection of specific A/V components for playback (e.g. a specific subtitle language, audio language, or camera angle), the classes defined in Sections 7.16.5.2 – 7.16.5.5 SHALL be supported and the constants, properties and methods defined in Section 7.16.5.1 SHALL be supported on the `video/broadcast` object.

7.13.5 Extensions to video/broadcast for parental ratings errors

For parental rating related errors or changes during playback of A/V content through the "video/broadcast" object an OITF SHALL support the following intrinsic event properties and corresponding DOM 2 events for the "video/broadcast" object:

function **onParentalRatingChange**(String contentID, ParentalRating rating, String DRMSystemID, Boolean blocked)
The function that is called whenever the parental rating of the content being played inside the embedded object changes. These events may occur at the start of a new content item, or during playback of a content item (e.g. during playback of linear TV). The specified function is called with four arguments contentID, rating, DRMSystemID, and blocked which are defined as follows: • String contentID – the content ID to which the parental rating change applies. If the event is generated by the DRM system, it SHALL be the unique identifier for that content in the context of the DRM system (i.e. in the case of Marlin BB it is the Marlin contentID). Otherwise it MAY be null or undefined. • ParentalRating rating – the parental rating value of the currently playing content. The ParentalRating object is defined in Section 7.9 • String DRMSystemID – optional argument that specifies the DRM System ID of the DRM system that generated the event as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. The value SHALL be null if the parental control is not enforced by a particular DRM system. • Boolean blocked – flag indicating whether consumption of the content is blocked by the parental control system as a result of the new parental rating value.

function **onParentalRatingError**(String contentID, ParentalRating rating, String DRMSystemID)
The function that is called when a parental rating error occurs during playback of A/V content inside the embedded object, and is triggered whenever a parental rating value is discovered for a parental rating system that is not supported by the OITF. The specified function is called with three arguments contentID, rating, and DRMSystemID which are defined as follows: • String contentID – the content ID to which the parental rating change applies. If the event is generated by the DRM system, it SHALL be the unique identifier for that content in the context of the DRM system (i.e. in the case of Marlin BB it is the Marlin contentID). Otherwise it MAY be null or undefined. • ParentalRating rating – the parental rating value of the currently playing content. The ParentalRating object is defined in Section 7.9. • String DRMSystemID – optional argument that specifies the DRM System ID of the DRM system that generated the event as defined by element DRMSystemID in Table 8 of Section 3.3.2

of [OIPF_META2]. The value SHALL be `null` if the parental control is not enforced by a particular DRM system.

7.13.5.1 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onParentalRatingChange`	`ParentalRatingChange`	Bubbles: No Cancelable: No Context Info: `contentID`, `rating`, `DRMSystemID` and `blocked`
`onParentalRatingError`	`ParentalRatingError`	Bubbles: No Cancelable: No Context Info: `contentID`, `rating`, and `DRMSystemID`.

Note: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a `ParentalRatingError` event during the bubbling or the capturing phase. The Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `video/broadcast` object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.13.6 Extensions to video/broadcast for DRM rights errors

This section SHALL apply to OITF and/or server devices which have indicated support for DRM protection by providing one or more `<drm>` elements as specified in Section 9.3.10:

For notifying Javascript about DRM licensing errors during playback of DRM protected A/V content through the "`video/broadcast`" object, an OITF SHALL support the following intrinsic event property and corresponding DOM 2 event for the "`video/broadcast`" object:

`function onDRMRightsError( Integer errorState, String contentID, String DRMSystemID, String rightsIssuerURL )`
The function that is called: • Whenever a rights error occurs for the A/V content (no license, license invalid), which has led to blocking consumption of the content. • Whenever a rights change occurs for the A/V content (license valid), which leads to unblocking the consumption of the content.

This may occur during playback, recording or timeshifting of DRM protected AV content. The specified function is called with four arguments `errorState`, `contentID`, `DRMSystemID` and `rightsIssuerURL` which are defined as follows:

- `Integer errorState` – error code detailing the type of error:

  0: no license, consumption of the content is blocked.

  1: invalid license, consumption of the content is blocked.

  2: valid license, consumption of the content is unblocked.

- `String contentID` – the unique identifier of the protected content in the scope of the DRM system that raises the error (i.e. in the case of Marlin BB it is the Marlin contentID, in the case of CSPG-CI+ and CSPG-DTCP this field is empty).

- `String DRMSystemID` – DRMSystemID as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is "`urn:dvb:casystemid:19188`".

- `String rightsIssuerURL` – optional element indicating the value of the rightsIssuerURL that can be used to non-silently obtain the rights for the content item currently being played for which this DRM error is generated, in cases whereby the `rightsIssuerURL` is known. Cases whereby the `rightsIssuerURL` is known include cases whereby the `rightsIssuerURL` has been extracted from the MPEG2_TS of the protected content, retrieved from the SD&S discovery record or from the associated BCG metadata. The corresponding `rightsIssuerURL` fields are defined in Section 4.1.3.4 of [OIPF_CSP2] and in section 3.3.2 of [OIPF_META2] respectively. If different URLs are retrieved from the stream and the metadata, then the conflict resolution is implementation-dependent.

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onDRMRightsError`	`DRMRightsError`	▪ Bubbles: No ▪ Cancelable: No ▪ Context Info: `errorState, contentID, DRMSystemID, rightsIssuerURL`

Note: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a DRMRightsError event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `video/broadcast` object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.13.7 Extensions to video/broadcast for current channel information

If an OITF has indicated support for extended tuner control (i.e. by giving value `true` to element `<extendedAVControl>` as specified in Section 9.3.6 in its capability description), the OITF SHALL support the following additional properties and methods on the `video/broadcast` object.

The functionality as described in this section is subject to the security model of Section 10.1.4.8.

Note the property `onChannelScan` and methods `startScan` and `stopScan` have been moved to section 7.13.10.

7.13.7.1 Properties

`readonly ChannelList` **`bindableChannels`**
The channels currently being presented under the control of the OITF (i.e. the channels that were being presented by the OITF when the application started). If no channels are being presented under the control of the OITF, the value of this property SHALL be `null`.

`readonly Channel` **`currentChannel`**
The channel currently being presented by this embedded object if the user has given permission to share this information, possibly through a mechanism outside the scope of this specification. If no channel is being presented, or if this information is not visible to the caller, the value of this property SHALL be `null`. The value of this property is not affected during timeshift operations and SHALL reflect the value prior to the start of a timeshift operation, for both local and network timeshift resources.

7.13.8 Extensions to video/broadcast for creating channel lists from SD&S fragments

Note the method `createChannelList()` has been moved to section 7.13.10.

7.13.9 Extensions to video/broadcast for synchronization

The OITF SHALL support the following additional methods on the video/broadcast object, in order to enable synchronization to broadcast events.

`void` **`addStreamEventListener`**`( String targetURL, String eventName, function listener)`	
Description	Add a listener for the specified DSM-CC stream event. Event triggers are carried in the stream as MPEG private data sections. For robustness, the section describing a particular trigger may be repeated several times. Each section has a version number which is used to disambiguate a new trigger for the same event (which will have a different version number) from a repeated instance of a previous trigger (which will have the same version number). When OITF detects a trigger corresponding to an event for which a listener has been registered, a DOM StreamEvent SHALL be dispatched. An event shall also be dispatched in case of error.

	An OITF SHALL dispatch only one DOM StreamEvent per unique trigger detected. Repeated instances of the same trigger SHALL NOT cause a new DOM StreamEvent to be dispatched. A new trigger for the same event (i.e. an MPEG private data section for the same event but with an updated version number) SHALL cause a new DOMStreamEvent to be dispatched.	
Arguments	*targetURL*	The URL of the DSM-CC StreamEvent object or the event description file describing the event as defined in Section 8.2 of [TS 102 809].
	eventName	The name of the event (in the DSM-CC StreamEvent object) that should be subscribed to.
	listener	The listener for the event.

`void` **`removeStreamEventListener`**`( String eventURL, String eventName, function listener)`		
Description	Remove a stream event listener for the specified stream event name.	
Arguments	*targetURL*	The URL of the DSM-CC StreamEvent object or the event description file describing the event as defined in Section 8.2 of [TS 102 809].
	eventName	The name of the event (in the DSM-CC StreamEvent object) whose subscription should be removed.
	listener	The listener for the event.

7.13.9.1 The StreamEvent class

The `StreamEvent` class is a subclass of the DOM 2 `Event` class which notifies an application that a synchronisation trigger in a broadcast stream has been detected. This event also notifies an application when the event is no longer being monitored.

Instances of this event are directly dispatched to the event target, and will not bubble nor capture.

`readonly String` **`eventName`**
The name of the stream event.

`readonly String` **`data`**
Data of the DSM-CC StreamEvent's event encoded in hexadecimal. For example: "0A10B81033" (for a message 5 bytes long).

`readonly String` **`text`**
Text data of the DSM-CC StreamEvent's event as a string, assuming UTF-8 as the encoding for the DSM-CC StreamEvent's event. Characters that cannot be transcoded SHALL be skipped.

readonly String **status**
The status of the event. Equal to "trigger" when the event is dispatched in response to a trigger in the stream or "error" when an error occurred (e.g. attempting to add a listener for an event that does not exist, or when a StreamEvent object with registered listeners is removed from the carousel). An event SHALL be dispatched with an error status if: • the StreamEvent object pointed to by targetURL is not found in the carousel or via broadband • the StreamEvent object pointed to by targetURL does not contain the event specified by the eventName parameter • the carousel containing the event cannot be mounted • the elementary stream which contains the StreamEvent event descriptor is no longer being monitored (e.g. due to another monitoring request or because it disappears from the PMT) • the event description file pointed to by targetURL is not available or does not have the correct syntax. Once an error is dispatched, the listener SHALL be automatically unregistered by the OITF.

7.13.10 The ChannelConfig class

The ChannelConfig object provides the entry point for applications to get information about available channels. It can be obtained in two ways:

- By calling the method getChannelConfig() of the video/broadcast embedded object as defined in Section 7.13.1.3.
- By calling the method createChannelConfig() of the object factory API as defined in Section 7.1.1.

The availability of the properties and methods are dependent on the capabilities description as specified in section 9.3. The following table provides a list of the capabilities and the associated properties and methods. If the capability is false the properties and methods SHALL NOT be available to the application. Properties and methods not listed in the following table SHALL be available to all applications as long as the OITF has indicated support for tuner control (i.e. <video_broadcast>true</video_broadcast> as defined in Section 9.3.1) in their capability.

Capability	Properties	Methods
Element <extendedAVControl> is set to "true" as defined in Section 9.3.6.	onChannelScan	startScan() stopScan()
Element <video_broadcast type="ID_IPTV_SDS"> is set as defined in Section 9.3.6.		createChannelList()

The functionality as described in this section is subject to the security model of Section 10.1.4.8.

7.13.10.1 Properties

readonly ChannelList **channelList**
The list of all available channels. The order of the channels in the list corresponds to the channel

ordering as managed by the OITF.

SHALL return the value `null` if the channel list is not (partially) managed by the OITF (i.e., if the channel list information is managed entirely in the network).

`readonly FavouriteListCollection` **`favouriteLists`**

A list of favourite lists. SHALL return the value `null` if the favourite lists are not (partially) managed by the OITF (i.e., if the favourite lists information is managed entirely in the network).

`readonly FavouriteList` **`currentFavouriteList`**

Currently active Favourite channel list object. If `currentFavouriteList` is `undefined`, no favourite filter list is currently applied.

The OITF SHALL return the value `null` if the favourite lists are not (partially) managed by the OITF (i.e. if the favourite lists information is managed entirely in the network).

```
function onChannelScan( Integer type, Integer progress, Integer frequency,
                        Integer signalStrength, Integer channelNumber,
                        Integer channelType, Integer channelCount,
                        Integer transponderCount)
```

This function is the DOM 0 event handler for events relating to channel scanning. On IP-only receivers, setting this property SHALL have no effect.

The specified function is called with the following arguments:

- `Integer type` - The type of event. Valid values are:

Value	Description
0	A channel scan has started.
1	Indicates the current progress of the scan.
2	A new channel has been found.
3	A new transponder has been found.
4	A channel scan has completed.
5	A channel scan has been aborted.

- `Integer progress` - the progress of the scan. Valid values are in the range 0 - 100, or -1 if the progress is unknown.
- `Integer frequency` - The frequency of the transponder in kHz (for scans on RF sources only).

- `Integer signalStrength` - The signal strength for the current channel. Valid values are in the range 0 - 100, or -1 if the signal strength is unknown.
- `Integer channelNumber` - The logical channel number of the channel that has been found.
- `Integer channelType` - The type of channel that has been found. Valid values are the same as for `Channel.channelType`.
- `Integer channelCount` - The total number of channels found so far during the scan.

`Integer transponderCount` - The total number of transponders found so far during the scan (RF sources only).

`function` **onChannelListUpdate**
This function is the DOM 0 event handler for events relating to channel list updates. Upon receiving a ChannelListUpdate event, if an application has references to any Channel objects then it SHOULD dispose of them and rebuild its references. Where possible Channel objects are updated rather than removed, but their order in the ChannelConfig.all collection MAY have changed. Any lists created with ChannelConfig.createFilteredList() SHOULD be recreated in case channels have been removed.

7.13.10.2 Methods

`ChannelList` **createFilteredList**`( Boolean blocked, Boolean favourite, Boolean hidden, String favouriteListID )`	
Description	Create a filtered list of channels. Returns a subset of `ChannelConfig.channelList`. The `blocked`, `favourite` and `hidden` flags indicate whether a channel is included in the returned list. These flags correspond to the properties on `Channel` with the same names. Each flag MAY be set to one of three values:

Value	Meaning
`true`	The channel is added if and only if the corresponding property has the value true.
`false`	The channel is added if and only if the corresponding property has the value false.
`undefined`	The channel is added regardless of the state of the corresponding property.

A channel will only be added to the list if the values of all three flags allow it to be added.

The `favouriteListID` attribute is used to select a particular `favouriteList` that the `createFilteredList` method uses as a basis of the filtering process. If `favouriteListID` is the empty string (i.e. “”), then the filtering is performed on all

	available channels as defined by `ChannelConfig.channelList`.	
Arguments	*blocked*	Flag indicating whether manually blocked channels SHALL be added to the list.
	favourite	Flag indicating whether favourite channels SHALL be added to the list.
	hidden	Flag indicating whether hidden channels SHALL be added to the list.
	favouriteListID	If the value of the favourite flag is `true`, indicates which favourites list SHALL be filtered upon.

`Integer` **`startScan`**`( ChannelScanOptions options, ChannelScanParameters scanParameters )`		
Description	Start a scan for new channels on all available sources. When each source finishes scanning, an `UpdateEvent` SHALL be raised with the type `CHANNELS_INVALIDATED` and any channel lists for that source SHALL have been updated. On IP-only receivers, this method SHALL have no effect.	
Arguments	*options*	The options to the channel scan operation.
	scanParameters	The tuning parameters to be scanned. The value of this argument SHALL be one of the classes that implements the `ChannelScanParameters` interface and SHALL NOT be an instance of the `ChannelScanParameters` class.

`void` **`stopScan`**`()`	
Description	Stop a channel scan, if one is in progress. Any sources that have not finished scanning SHALL have their scans aborted and channel line-ups for SHALL NOT be changed. On IP-only receivers, this method SHALL have no effect.

`ChannelList` **`createChannelList`**`( String bdr )`		
Description	Creates a `ChannelList` object from the specified SD&S Broadcast Discovery Record. Channels in the returned channel list will not be included in the channel list that can be retrieved via calls to `getChannelConfig()`.	
Arguments	*bdr*	An XML-encoded string containing an SD&S Broadcast Discovery Record as specified in [OIPF_META2]. If the string is not a valid Broadcast Discovery Record, this method SHALL return `null`.

`Channel` **`createChannelObject`**`( Integer idType, Integer onid, Integer tsid, Integer sid, Integer sourceID, String ipBroadcastID )`

Description	Creates a `Channel` object of the specified `idType`. The Channel object can subsequently be used by the `setChannel` method to switch a tuner to a channel that is not part of the channel list which was conveyed by the OITF to the server. The scope of the resulting `Channel` object is limited to the Javascript environment (incl. `video/broadcast` object) to which the `Channel` object is returned, i.e. it does not get added to the channellist available through method `getChannelConfig`. If the channel of the given `idType` cannot be created or the given (combination of) arguments are not considered valid or complete, the method SHALL return `null`. If the channel of the given type can be created and arguments are considered valid and complete, the method SHALL return a `Channel` object whereby at a minimum the properties with the same names are given the same value as the given arguments of the `createChannelObject` method. The values specified for the remaining properties of the `Channel` object are set to `undefined`.	
Arguments	*idType*	The type of channel, as indicated by one of the ID_* constants defined in Section 7.13.12.1.
	onid	The original network ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type ID_DVB_* or ID_ISDB_*.
	tsid	The transport stream ID. Optional argument that MAY be specified when the `idType` specifies a channel of type `ID_DVB_*` or `ID_ISDB_*`.
	sid	The service ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_DVB_*` or `ID_ISDB_*`.
	sourceID	The source_ID. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_ATSC_T`.
	ipBroadcastID	The DVB textual service identifier of the IP broadcast service, specified in the format "`ServiceName.DomainName`", or the URI of the IP broadcast service. Optional argument that SHALL be specified when the `idType` specifies a channel of type `ID_IPTV_SDS` or `ID_IPTV_URI`.

7.13.10.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onChannelScan`	`ChannelScan`	▪ Bubbles: No ▪ Cancelable: No ▪ Context Info: `type, progress, frequency, signalStrength, channelNumber, channelType, channelCount, transponderCount`

onChannelListUpdate	ChannelListUpdate	▪ Bubbles: No ▪ Cancelable: No ▪ Context Info: none

Note: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the `ChannelConfig` object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.13.11 The ChannelList class

```
typedef Collection<Channel> ChannelList
```

A `ChannelList` represents a collection of `Channel` objects. See annex K for the definition of the collection template.

In addition to the methods and properties defined for generic collections, the `ChannelList` class supports the additional properties and methods defined below.

7.13.11.1 Methods

`Channel` **`getChannel`**`( String channelID )`		
Description	Return the first channel in the list with the specified channel identifier. Returns `null` if no corresponding channel can be found.	
Arguments	*channelID*	The channel identifier of the channel to be retrieved. Valid values are as defined for the `ccid` and `ipBroadcastID` properties of the `Channel` object as defined in Section 7.13.12.

`Channel` **`getChannelByTriplet`**`( Integer onid, Integer tsid, Integer sid )`		
Description	Return the first (IPTV or non-IPTV) channel in the list that matches the specified DVB or ISDB triplet (original network ID, transport stream ID, service ID). Where no channels of type ID_ISDB_* or ID_DVB_* are available, or no channel identified by this triplet are found, this method SHALL return `null`.	
Arguments	*onid*	The original network ID of the channel to be retrieved.
	tsid	The transport stream ID of the channel to be retrieved. If set to `null` the client SHALL retrieve the channel defined by the combination of onid and sid. This makes it possible to retrieve the correct channel also in case a remultiplexing took place which led to a changed tsid.
	sid	The service ID of the channel to be retrieved.

Channel **getChannelBySourceID**(Integer sourceID)		
Description	Return the first (IPTV or non-IPTV) channel in the list with the specified ATSC source ID. Where no channels of type ID_ATSC_* are available, or no channel with the specified source ID is found in the channel list, this method SHALL return `null`.	
Arguments	*sourceID*	The ATSC source_ID of the channel to be returned.

7.13.12 The Channel class

The `Channel` object represents a broadcast stream or service. It is defined as follows:

7.13.12.1 Constants

The following constants are defined as properties of the `Channel` class:

Name	Value	Use
TYPE_TV	0	Used in the `channelType` property to indicate a TV channel.
TYPE_RADIO	1	Used in the `channelType` property to indicate a radio channel.
TYPE_OTHER	2	Used in the `channelType` property to indicate that the type of the channel is unknown or known but not of type TV or radio.
ID_ANALOG	0	Used in the `idType` property to indicate an analogue channel identified by the property `freq` and optionally `cni` or `name`.
ID_DVB_C	10	Used in the `idType` property to indicate a DVB-C channel identified by the three properties `onid`, `tsid`, `sid`.
ID_DVB_S	11	Used in the `idType` property to indicate a DVB-S channel identified by the three properties `onid`, `tsid`, `sid`.
ID_DVB_T	12	Used in the `idType` property to indicate a DVB-T channel identified by the three properties `onid`, `tsid`, `sid`.
ID_DVB_SI_DIRECT	13	Used in the `idType` property to indicate a channel that is identified through its delivery system descriptor as defined by DVB-SI [EN 300 468] section 6.2.13.
ID_DVB_C2	14	Used in the `idType` property to indicate a DVB-C or DVB-C2 channel identified by the three properties `onid`, `tsid`, `sid`.
ID_DVB_S2	15	Used in the `idType` property to indicate a DVB-S or DVB-S2 channel identified by the three properties `onid`, `tsid`, `sid`.
ID_DVB_T2	16	Used in the `idType` property to indicate a DVB-T or DVB-T2 channel identified by the three properties `onid`, `tsid`, `sid`.
ID_ISDB_C	20	Used in the `idType` property to indicate an ISDB-C channel identified by the three properties: `onid`, `tsid`, `sid`.

Name	Value	Use
ID_ISDB_S	21	Used in the idType property to indicate an ISDB-S channel identified by the three properties onid, tsid, sid.
ID_ISDB_T	22	Used in the idType property to indicate an ISDB-T channel identified by the three properties onid, tsid, sid.
ID_ATSC_T	30	Used in the idType property to indicate a terrestrial ATSC channel identified by the property sourceID.
ID_IPTV_SDS	40	Used in the idType property to indicate an IP broadcast channel identified through SD&S by a DVB textual service identifier specified in the format "*ServiceName*.*DomainName*" as value for property ipBroadcastID with *ServiceName* and *DomainName* as defined in [DVB-IPTV]. This idType SHALL be used to indicate Scheduled content service defined by [OIPF_PROT2]
ID_IPTV_URI	41	Used in the idType property to indicate an IP broadcast channel identified by a DVB MCAST URI (e.g. i.e. dvb-mcast://), as value for property ipBroadcastID.

7.13.12.2 Properties

This section defines the properties of the Channel object.

Properties that do not apply in a specific circumstance (e.g. onid does not apply unless the channel is of type ID_DVB_* or ID_ISDB_*) SHALL be undefined.

readonly Integer **channelType**
The type of channel, as indicated by one of the TYPE_* constants defined above.

readonly Integer **idType**
The type of identification for the channel, as indicated by one of the ID_* constants defined above.

readonly String **ccid**
Unique identifier of a channel within the scope of the OITF. The ccid is defined by the OITF and SHALL have prefix 'ccid: e.g., 'ccid:*{tunerID.}*majorChannel*{.minorChannel}*'. Note: the format of this string is platform-dependent.

readonly String **tunerID**
Optional unique identifier of the tuner within the scope of the OITF that is able to receive the given channel.

readonly Integer **onid**
DVB or ISDB original network ID (for channels of type ID_DVB_* and ID_ISDB_*); can be undefined if stream does not contain an SDT Actual.

readonly Integer **tsid**
DVB or ISDB transport stream ID (for channels of type ID_DVB_* and ID_ISDB_*).

readonly Integer **sid**
DVB or ISDB service ID (for channels of type ID_DVB_* and ID_ISDB_*).

readonly Integer **sourceID**
ATSC source_ID value.

readonly Integer **freq**
For analogue channels, the frequency of the video carrier in kHz.

readonly Integer **cni**
For analogue channels, the VPS/PDC confirmed network identifier.

readonly String **name**
The name of the channel. Can be used for linking analog channels without CNI. Typically, it will contain the call sign of the station (e.g. 'HBO'). For channels of type ID_DVB_* the service name is to be used.

readonly Integer **majorChannel**
The major channel number, if assigned. Value undefined otherwise. Typically used for channels of type ID_ATSC_*. For channels of type ID_IPTV_SDS the major channel represents the logical channel number. The number is populated during SD&S and the LogicalChannelNumber element in the Package Discovery Record[DVB-IPTV].

readonly Integer **minorChannel**
The minor channel number, if assigned. Value undefined otherwise. Typically used for channels of type ID_ATSC_*.

readonly String **dsd**
For channels of type ID_DVB_SI_DIRECT created through createChannelObject(), this property defines the delivery system descriptor (tuning parameters) as defined by DVB-SI [EN 300 468] section 6.2.13. The dsd property provides a string whose characters shall be restricted to the ISO Latin-1 character set. Each character in the dsd represents a byte of a delivery system descriptor as defined by DVB-SI [EN 300 468] section 6.2.13, such that a byte at position "i" in the delivery system descriptor is equal the Latin-1 character code of the character at position "i" in the dsd. Described in the syntax of ECMAScript: let sdd[] be the byte array of a system delivery descriptor, in which sdd[0] is the descriptor_tag, then, dsd is its equivalent string, if : dsd.length==sdd.length and for each integer i : 0<=i<dsd.length holds: sdd[i] == dsd.charCodeAt(i).

readonly Boolean **favourite**
Flag indicating whether the channel is marked as a favourite channel or not in one of the favourite lists as defined by the property favIDs.

readonly StringCollection **favIDs**
The names of the favourite lists to which this channel belongs (see the favouriteLists property on the ChannelConfig class).

readonly Boolean **locked**
Flag indicating whether the current state of the parental control system prevents the channel from being viewed (e.g. a correct parental control pin has not been entered). Note that this property supports the option of client-based management of parental control without excluding server-side implementation of parental control.

readonly Boolean **manualBlock**
Flag indicating whether the user has manually blocked viewing of this channel. Manual blocking of a channel will treat the channel as if its parental rating value always exceeded the system threshold. Note that this property supports the option of client-based management of manual blocking without excluding server-side management of blocked channels.

readonly String **ipBroadcastID**
If the Channel has idType ID_IPTV_SDS, this element denotes the DVB textual service identifier of the IP broadcast service, specified in the format “ServiceName.DomainName” with the ServiceName and DomainName as defined in [DVB-IPTV].

If the Channel has idType ID_IPTV_URI, this element denotes a URI of the IP broadcast service.

readonly Integer **channelMaxBitRate**
The MaxBitRate associated to the channel is returned through this property. The MaxBitRate is provided through SD&S as defined in section 3.2.2 of [OIPF_META2]. The property is only related to IP based broadcast of type `ID_IPTV_SDS`. If the field does not exist, this method SHALL return `undefined`.

readonly Integer **channelTTR**
The TTR (TimeToRenegotiate) associated to the channel is returned through this property. The MBR is provided through SD&S as defined in section 3.2.2 of [OIPF_META2]. The property is only related to IP based broadcast of type `ID_IPTV_SDS`. If the field does not exist, this method SHALL return `undefined`.

7.13.12.3 Metadata extensions to Channel

This subsections SHALL apply for OITFs that have indicated `<clientMetadata>` with value "`true`" and a `type` attribute with values "`bcg`", "`sd-s`", "`eit-pf`" or "`dvb-si`" as defined in Section 9.3.7 in their capability profile.

The OITF SHALL extend the `Channel` class with the properties and methods described below.

The values of many of these properties are derived from elements in the BCG metadata. For optional elements that are not present in the metadata, the default value of any property that derives its value from one of those elements SHALL be `undefined`.

7.13.12.3.1 Properties

readonly String **longName**
The long name of the channel. If both short and long names are being transmitted, this property SHALL contain the long name of the station (e.g. 'Home Box Office'). If the long name is not available, this property SHALL be `undefined`. The value of this property is derived from the `Name` element that is a child of the BCG `ServiceInformation` element describing the channel, where the `length` attribute of the `Name` element has the value 'long'.

readonly String **description**
The description of the channel. If no description is available, this property SHALL be `undefined`. The value of this field is taken from the `ServiceDescription` element that is a child of the BCG `ServiceInformation` element describing this channel.

readonly Boolean **authorised**

Flag indicating whether the receiver is currently authorised to view the channel. This describes the conditional access restrictions that may be imposed on the channel, rather than parental control restrictions.

`readonly StringCollection` **`genre`**
A collection of genres that describe the channel. This field contains the values of. any `ServiceGenre` elements that are children of the BCG `ServiceInformation` element describing the channel

`Boolean` **`hidden`**
Flag indicating whether the channel is included in the default channel list. A value of true means that the OITF SHALL exclude this channel from the default channel list.

`string` **`logoURL`**
The URL for the default logo image for this channel. The value of this field is derived from the value of the first `Logo` element that is a child of the BCG `ServiceInformation` element describing the channel. If this element specifies anything other than the URL of an image, the value of this filed SHALL be `undefined`.

7.13.12.3.2 Methods

`String` **`getField`**`( String fieldId )`		
Description	Get the value of the field referred to by `fieldId` that is contained in the BCG metadata for this channel. If the field does not exist, this method SHALL return `undefined`.	
Arguments	*fieldId*	The name of the field whose value SHALL be retrieved.

`String` **`getLogo`**`( Integer width, Integer height )`	
Description	Get the URI for the logo image for this channel. The width and height parameters specify the desired width and height of the image; if an image of that size is not available, the URI of the logo with the closest available size not exceeding the specified dimensions SHALL be returned. If no image matches these criteria, this method SHALL return `null`. The URI returned SHALL be suitable for use as the `SRC` attribute in an HTML `IMG` element or as a background image. The URIs returned by this method will be derived from the values of the `Logo` elements that are children of the BCG `ServiceInformation` element describing the channel .

Arguments	*width*	The desired width of the image
	height	The desired height of the image

7.13.13 The FavouriteListCollection class

```
typedef Collection<FavouriteList> FavouriteListCollection
```

The `FavouriteListCollection` class represents a collection of `FavouriteList` objects. See annex K for the definition of the collection template. In addition to the methods and properties defined for generic collections, the `FavouriteListCollection` class supports the additional methods defined below.

7.13.13.1 Methods

`FavouriteList` **`getFavouriteList`**`( String favID )`		
Description	Return the first favourite list in the collection with the given `favListID`.	
Arguments	*favID*	The ID of a favourite list.

7.13.13.2 Extensions to FavouriteListCollection

If an OITF has indicated support for extended tuner control (i.e. by giving value `true` to element `<extendedAVControl>` as specified in Section 9.3.6 in its capability description), the OITF SHALL support the following additional constants and methods on the `FavouriteListCollection` object.

The functionality as described in this section is subject to the security model of Section 10.1.4.8.

`FavouriteList` **`createFavouriteList`**`(String name)`		
Description	Create a new favourite list and add it to the collection. The ID of the new favourite list SHALL be returned.	
Arguments	name	The name to be associated to the new favourite list.

`Boolean` **`remove`**`( Integer index )`		
Description	Remove the list at the specified index from the collection. This method SHALL return `true` of the operation succeeded, or `false` if an invalid index was specified.	
Arguments	*index*	The index of the list to be removed.

`Boolean` **`commit`**`()`
Description: Commit any changes to the collection to persistent storage. This method SHALL return `true` of the operation succeeded, or `false` if it failed (e.g. due to insufficient space to store the collection).

	If a server has indicated that it requires control of the tuner functionality of an OITF in the server capability description for a particular service, then the OITF SHOULD send an updated Client Channel Listing to the server using HTTP POST over TLS as described in section 4.8.1.1.

Boolean **activateFavouriteList**(string favID)		
Description	Active the favourite list from the collection. This method SHALL return `true` if the operation succeeded, or `false` if an invalid index was specified. A newly created favourite list has to be committed before it can be activated.	
Arguments	*favID*	The ID of a favourite list.

7.13.14 The FavouriteList class

```
typedef Collection<Channel> FavouriteList
```

The `FavouriteList` class represents a list of favourite channels. See annex K for the definition of the collection template. In addition to the methods and properties defined for generic collections, the `FavouriteList` class supports the additional properties and methods defined below.

In order to preserve backwards compatibility with already existing DAE content the ECMAScript `toString()` method SHALL return the `FavouriteList.id` for `FavouriteList` objects.

7.13.14.1 Properties

readonly String **favID**
A unique identifier by which the favourite list can be identified

String **name**
A descriptive name given to the favourite list

7.13.14.2 Methods

Channel **getChannel**(String channelID)		
Description	Return the first channel in the favourite list with the specified channel identifier. Returns `null` if no corresponding channel can be found.	
Arguments	*channelID*	The channel identifier of the channel to be retrieved, which is a value as defined for property `ccid` of the `Channel` object or a value as defined for property `ipBroadcastID` of the `Channel` object as defined in Section 7.13.12.

Channel **getChannelByTriplet**(Integer onid, Integer tsid, Integer sid)		
Description	Return the first (IPTV or non-IPTV) channel in the list that matches the specified DVB or ISDB triplet (original network ID, transport stream ID, service ID). Where no channels of type ID_ISDB_* or ID_DVB_* are available, or no channel identified by this triplet are found, this method SHALL return `null`.	
Arguments	*onid*	The original network ID of the channel to be retrieved.
	tsid	The transport stream ID of the channel to be retrieved. If set to `null` the client SHALL retrieve the channel defined by the combination of `onid` and `sid`. This makes it possible to retrieve the correct channel also in case a remultiplexing took place which led to a changed tsid.
	sid	The service ID of the channel to be retrieved.

Channel **getChannelBySourceID**(Integer sourceID)		
Description	Return the first (IPTV or non-IPTV) channel in the list with the specified ATSC source ID. Where no channels of type ID_ATSC_* are available, or no channel with the specified source ID is found in the channel list, this method SHALL return `null`.	
Arguments	*sourceID*	The ATSC source_ID of the channel to be returned.

7.13.14.3 Extensions to FavouriteList

If an OITF has indicated support for extended tuner control (i.e. by giving value `true` to element `<extendedAVControl>` as specified in Section 9.3.6 in its capability description), the OITF SHALL support the following additional constants and methods on the `FavouriteList` object.

When the `FavouriteList` object is updated with new or removed channels it does not take effect until the object is committed. Only after `commit()` will the updates of a `FavouriteList` object become available to other DAE applications.

The `name` property of the `FavouriteList` object SHALL be read/write for OITFs which are controlled by a service provider. The following methods SHALL also be supported:

Boolean **insertBefore**(Integer index, String ccid)		
Description	Insert a new favourite into the favourites list at the specified index. In order to add a ccid at the end of the favourite list the index shall be equal to `length`. This method SHALL return `true` of the operation succeeded, or `false` if an invalid index was specified (e.g. `index` > `length`).	
Arguments	*index*	The index in the list before which the favourite should be inserted.
	ccid	The ccid of the channel to be added.

Boolean **remove**(Integer index)

Description	Remove the item at the specified index from the favourites list. Returns `true` of the operation succeeded, or `false` if an invalid index was specified.	
Arguments	*index*	The index of the item to be removed.

`Boolean` **`commit`**`()`
Description: Commit any changes to the favourites list to persistent storage. This method SHALL return `true` of the operation succeeded, or `false` if it failed (e.g. due to insufficient space to store the list on the OITF). If a server has indicated that it requires control of the tuner functionality of an OITF in the server capability description for a particular service, then the OITF SHOULD send an updated Client Channel Listing to the server using HTTP POST over TLS as described in section 4.8.1.1.

7.13.15 The ChannelScanOptions class

The `ChannelScanOptions` class defines the options that should be applied during a channel scan operation. This class does not define parameters for the channel scan itself.

7.13.15.1 Properties

`Integer` **`channelType`**
The types of channel that should be discovered during the scan. Valid values are `TYPE_RADIO`, `TYPE_TV` or `TYPE_OTHER` as defined in section 7.13.12.1.

`Boolean` **`replaceExisting`**
If true, any existing channels in the channel list managed by the OITF SHALL removed and the new channel list SHALL consist only of channels found during the channel scan operation. If false, any channels discovered during the channel scan SHALL be added to the existing channel list.

7.13.16 The ChannelScanParameters class

This is an empty class that acts as the base interface for channel scan parameters specific to certain types of broadcast network.

7.13.17 The DVBTChannelScanParameters class

The `DVBTChannelScanParameters` class represents the parameters needed to perform a channel scan on a DVB-T or DVB-T2 network. This class implements the interface defined by `ChannelScanParameters`, with the following additions.

7.13.17.1 Properties

`Integer` **`startFrequency`**
The start frequency of the scan, in kHz.

`Integer` **endFrequency**

The end frequency of the scan, in kHz.

`String` **ofdm**

The Orthogonal Frequency Division Multiplexing (OFDM) for the indicating frequency. Valid values are:

Value	Description
MODE_1K	OFDM mode 1K
MODE_2K	OFDM mode 2K
MODE_4K	OFDM mode 4K
MODE_8K	OFDM mode 8K
MODE_16K	OFDM mode 16K
MODE_32K	OFDM mode 32K

`Integer` **modulationModes**

The modulation modes to be scanned. Valid values are:

Value	Description
1	QPSK modulation
4	QAM16 modulation
8	QAM32 modulation
16	QAM64 modulation
32	QAM128 modulation
64	QAM256 modulation

More than one of these values may be ORed together in order to indicate that more than one modulation mode should be scanned.

`String` **bandwidth**

The expected bandwidth. Valid values are:

Value	Description

BAND_1.7MHZ	1.7 MHz bandwidth
BAND_5MHz	5 MHz bandwidth
BAND_6MHz	6 MHz bandwidth
BAND_7MHz	7 MHz bandwidth
BAND_8MHz	8 MHz bandwidth
BAND_10MHz	10 MHz bandwidth

7.13.18 The DVBSChannelScanParameters class

The DVBSChannelScanParameters class represents the parameters needed to perform a channel scan on a DVB-S or DVB-S2 network. This class implements the interface defined by ChannelScanParameters, with the following additions.

7.13.18.1 Properties

Integer **startFrequency**

The start frequency of the scan, in kHz.

Integer **endFrequency**

The end frequency of the scan, in kHz.

Integer **modulationModes**

The modulation modes to be scanned. Valid values are:

Value	Description
1	QPSK modulation
2	8PSK modulation
4	QAM16 modulation

More than one of these values may be ORed together in order to indicate that more than one modulation mode should be scanned.

String **symbolRate**

A comma-separated list of the symbol rates to be scanned, in symbols/sec.

Integer **polarisation**

The polarisation to be scanned. Valid values are:

Value	Description
1	Horizontal polarisation
2	Vertical polarisation

More than one of these values may be ORed together in order to indicate that more than one polarisation should be scanned.

`String` **`codeRate`**

The code rate, e.g. “3/4” or “5/6”.

`Integer` **`networkId`**

The network ID of the network to be scanned, or `undefined` if all networks should be scanned.

7.13.19 The DVBCChannelScanParameters class

The `DVBCChannelScanParameters` class represents the parameters needed to perform a channel scan on a DVB-C or DVB-C2 network. This class implements the interface defined by `ChannelScanParameters`, with the following additions.

7.13.19.1 Properties

`Integer` **`startFrequency`**

The start frequency of the scan, in kHz.

`Integer` **`endFrequency`**

The end frequency of the scan, in kHz.

`Integer` **`modulationModes`**

The modulation modes to be scanned. Valid values are:

Value	Description
4	QAM16 modulation
8	QAM32 modulation
16	QAM64 modulation
32	QAM128 modulation

64	QAM256 modulation
128	QAM1024 modulation
256	QAM4096 modulation

More than one of these values may be ORed together in order to indicate that more than one modulation mode should be scanned.

String **symbolRate**
A comma-separated list of the symbol rates to be scanned, in symbols/sec.

Integer **networkId**
The network ID of the network to be scanned, or undefined if all networks should be scanned.

7.14 Media playback APIs

This section specifies several extensions to the audio object and the video object defined in Section 5.7.1 of [CEA-2014-A]. It also contains a subsection (i.e. Section 7.14.12) that defines the audio playback from memory API.

7.14.1 The CEA 2014 A/V Control embedded object

An OITF SHALL support a CEA 2014 A/V Control object as defined in Section 5.7.1 of [CEA-2014-A] for all mandatory media formats as defined in Section 10.1 of [OIPF_MEDIA2].

7.14.1.1 State diagram for A/V control objects

The following state transition diagram SHOULD be used for an A/V control object:

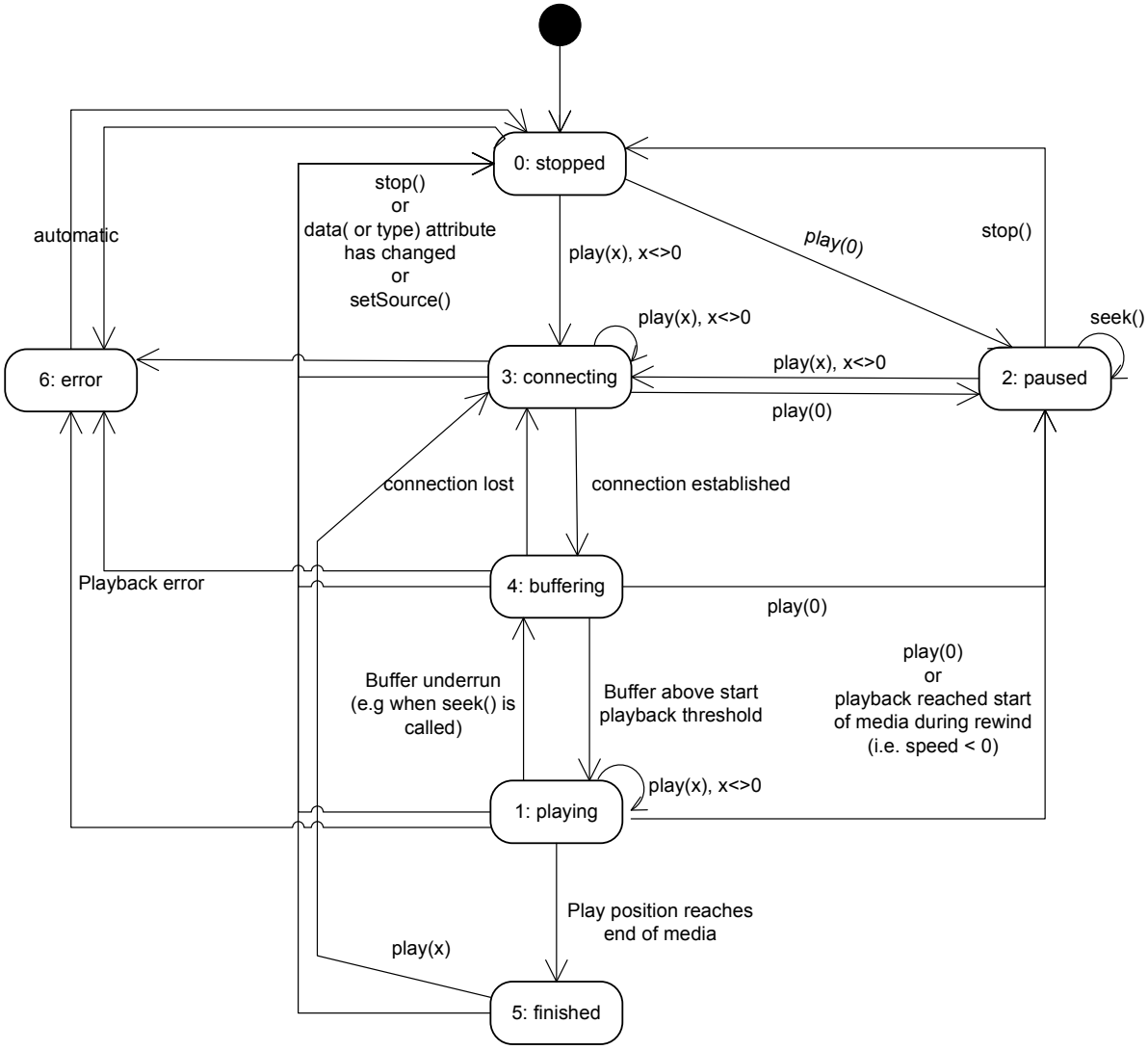


Figure 17: State diagram for embedded A/V Control objects

The following clarifications apply:

1) A detailed description for all the states in this state diagram is given in Annex B.5.

2) Scarce resources for playback using the A/V Control object, such as the MPEG decoder, are claimed during state 3 (‘`connecting`’), state 4 (‘`buffering`’) or during state transitions from state 3 (‘`connecting`’) to state 4 (‘`buffering`’), from state 4 (‘`buffering`’) to state 1 (‘`playing`’) or from state 0 (‘`stopped`’) or from state 3 (‘`connecting`’) to state 2 (‘`paused`’). If at any point in time during playback the scarce resources are not available anymore, due to a resource conflict, then the play state of the A/V Control object SHALL be set to 6 (‘`error`’) with a detailed error code of 3 (‘`insufficient resources`’). Scarce resources for playback using the A/V Control object SHALL be released when state 6 (‘`error`’) or 0 (‘`stopped`’) are reached. In addition, if the A/V Control object gets destroyed, e.g. because another URL is loaded into the containing window, scarce resources claimed for playback using the A/V Control object SHALL be released, except in cases described for the optional ‘`persist`’ property of A/V Control objects.

3) When the ‘`data`’ attribute and/or the ‘`type`’ attribute of the `HTMLObjectElement` representing the A/V Control object is given a different value, the object SHALL go to state 0 (‘`stopped`’).

4) For playback of DRM protected content, the rights for playback are retrieved during state 3 (‘`connecting`’).

5) If the play position reaches the end of the available content the A/V Control object SHALL be set to state 5 (‘finished’) in addition to generating a playback speed change of zero.

   If there is an attempt to play() with a speed greater than zero and there is no content available then the request fails.

6) If the play position reaches the beginning of the available content the A/V Control object SHALL be set to state 2 (‘paused’) in addition to generating a playback speed change of zero.

   If there is an attempt to play() with a speed less than zero and there is no content available then the request fails.

7) If seek() is performed beyond the available content the request is rejected and the current playout is maintained.

8) The visibility of an A/V Control object SHALL NOT affect its state or its use of scarce resources. An A/V Control object which is hidden which is hidden using one of the following techniques:

   - the CSS visibility or opacity properties
   - using the CSS display:none rule
   - removed from the document’s DOM
   - obscured by other elements
   - positioned off the visible area of the screen

   SHALL still be decoding video if it is in the playing state and any audio associated with the currently playing media will still be audible. State transitions caused by calls to methods on the A/V Control object, or due to permanent or transient errors, will occur as shown above regardless of the visibility of the object. Section 4.4.4 describes the effect on scarce resources when an A/V Control object is removed from the DOM tree.

9) When an A/V Control object is destroyed (e.g. by the A/V Control object being garbage collected, or because of a page transition within the application), presentation of streamed audio or video shall be terminated.

7.14.1.2 Using an A/V control object to play streaming content

If an A/V control object is used to play streamed content using either RTSP or HTTP the OITF then the following holds:

- If play(0) is called in state 0 (‘stopped’), the A/V Control object SHALL automatically go to play state 2 ('paused'). The necessary resources are secured and no external signalling is performed.
- If play(0) is called in the connecting or buffering state, the A/V Control object SHALL automatically go to play state 2 ('paused')

7.14.1.3 Using an A/V control object to play downloaded content

If an A/V control object is used to play content that has been downloaded and stored on the OITF on the OITF (by using method setSource() as defined in Section 7.14.8) then the following holds:

1) if the download was triggered using registerDownloadURL or the download was triggered using a Content Access Download Descriptor with <TransferType> value “playable_download” as defined in Annex E.1, then:

   a. if the play() method is called before sufficient data has been download to initiate playback, then the play state of the A/V Control object SHALL be set to 6 (‘error’) with a detailed error code of 5 (“content not available”).

2) if the downloaded content was triggered using a Content Access Download Descriptor with <TransferType> value “full_download” as defined in Annex E.1, then:

   a. if the play() method is called whilst the content is still downloading and has not yet successfully completed, then the play state of the A/V Control object SHALL be set to 6 (‘error’) with a detailed error code of 5 (“content not available”).

7.14.1.4 Using an A/V control object to play recorded content

If an A/V control object is used to play content that has been recorded or is being recorded on the OITF (by using method setSource() as defined in Section 7.14.8) then the following holds:

- if the play() method is called before sufficient data has been recorded to initiate playback, then the play state of the A/V Control object SHALL be set to 6 ('error') with a detailed error code of 5 ("content not available").

7.14.2 Extensions to A/V Control object for playback through Content-Access Streaming Descriptor

As specified in Section 4.7.1, an OITF SHALL support setting up the A/V stream using the information provided by a valid Content Access Streaming Descriptor referred to by the 'data' attribute. To this end, the OITF SHALL fetch the Content Access Streaming Descriptor from the URL provided by the data attribute, after which the descriptor SHALL be interpreted, resulting in an appropriate <ContentURL> to be selected (e.g. based on which DRM system the OITF supports). The OITF SHALL then initiate a streaming CoD session to the selected <ContentURL>, after which playback can be started when the play() method is invoked.

The OITF SHALL pass included DRM-information of the selected content and DRM system ID as part of the <DRMControlInformation> elements of a Content Access Streaming Descriptor to the DRM agent, if it supports a DRM agent with a matching DRMSystemID as per Section 9.3.10.

If the Content Access Streaming Descriptor is not valid according to the XML Schema and semantics as defined in Annex E.2, the A/V control object SHALL go to playState 6 (i.e. error), with error value 4 as defined in Annex B.

For more information about setting up the A/V stream based on a Content Access Streaming descriptor, see Section 4.7.1, Section 8 and Annex D.

7.14.3 Extensions to A/V Control object for media queuing

The following additional method SHALL be supported on the audio object and video object defined in Section 5.7.1 of [CEA-2014-A].

boolean **queue**(String uri)	
Description	Queue the media referred to by uri for playback after the current media item has finished playing. If a media item is already queued, uri will not be queued for playback and this method will return false. If the item is queued successfully, this method returns true. If no media is currently playing, the queued item will be played immediately. If uri is null, any currently queued item will be removed from the queue and this method will return true. If an A/V Control object is an audio object as defined by Section 5.7.1.b.1 of [CEA-2014-A] then queued media items shall only contain audio. If an A/V Control object is a video object as defined by Section 5.7.1.b.2 of [CEA-2014-A] then queued media items shall always contain video and may also contain audio and other media components. Applications SHOULD ensure the value of uri refers to a media format appropriate to the instance of the A/V Control object. When the current media item has finished playing, the A/V Control object shall transition to the finished state, update the value of the data property with the URL of the queued media item and automatically start playback of the queued media item. The A/V Control object MAY transition to the connecting or buffering states (and generate the necessary PlayStateChange events) before entering the playing state when the queued media item is being presented. Implementations may pre-buffer data from the queued URL before the current media item has finished playing in order to reduce the delay between items. If the queued media item can be played without transitioning to the connecting or buffering states, then the A/V Control object SHALL generate a PlayStateChanged

	event to the playing state to indicate that the queued media item has started playing. If playback of the current media item is stopped using the `stop()` method, or if the `data` property is modified, the queued media item SHALL NOT be played and the queued media item shall be discarded as if no item was queued. Play speed is not affected by transitioning between the current and queued media item. To avoid race conditions when queuing multiple items for playback, applications should wait for the currently queued item to begin playback before queuing subsequent items, e.g. by queuing the subsequent item when the A/V Control object transitions to the `connecting`, `buffering` or `playing` state for the currently queued item.	
Argument	*url*	The media item to be queued, or `null` to remove the currently-queued item.

7.14.4 Extensions to A/V Control object for trickmodes

7.14.4.1 Properties

The following additional properties SHALL be supported on the audio object and video object defined in Section 5.7.1 of [CEA-2014-A].

`function` **`onPlaySpeedChanged`**`( Number speed )`

The function that is called when the playback speed of the media changes.

The specified function is called with one argument, `speed`, which is defined as follows:

- `Number speed` – the playback speed of the media at the time the event was dispatched.

The behaviour of the A/V Control object when the end of media (or the end of the currently-available media) is reached is defined in Section 7.14.1.

`function` **`onPlayPositionChanged`**`( Integer position )`

The function that is called when change occurs in the play position of the media due to the use of trick play functions.

The specified function is called with one argument, `position`, which is defined as follows:

- `position` – the playback position of the media at the time the event was dispatched, measured in milliseconds since the beginning of the referenced media as denoted by the server.

The behaviour of the A/V Control object when the end of media (or the end of the currently-available media) is reached is defined in section 7.14.1.

`readonly Number` **`playSpeeds`**`[]`

Returns an ordered list of playback speeds, expressed as values relative to the normal playback speed (1.0), at which the currently specified A/V content can be played (either through an CEA-2014 audio or

video object), or undefined if the supported playback speeds are not (yet) known.

function **onplaySpeedsArrayChanged**()
The function that is called when the playSpeeds array values have changed. An application that makes use of the playSpeeds array needs to read the values of the playSpeeds property again.

readonly String **oitfSourceIPAddress**
The OITF source IP address for RTSP or HTTP signalling, as well as, the address where the RTSP stream is expected to arrive. The information shall be available in “buffering”, “paused” or “playing” states.

readonly String **oitfSourcePortAddress**
The OITF Port Address where the RTSP stream is expected to arrive. The information shall be available in “buffering”, “paused” or “playing” states.

Boolean **oitfNoRTSPSessionControl**
When the oitfNoRTSPSessionControl is set to true then the OITF SHALL NOT signal the RTSP messages DESCRIBE, SETUP or TEARDOWN.

String **oitfRTSPSessionId**
The sessionId to be used by the A/V Control Object when signalling RTSP. This property is only applicable when property oitfNoRTSPSessionControl is set to true.

7.14.4.2 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onPlaySpeedChanged	PlaySpeedChanged	Bubbles: No Cancelable: No Context Info: speed
onPlayPositionChanged	PlayPositionChanged	Bubbles: No Cancelable: No Context Info: position

onPlaySpeedsArrayChanged	PlaySpeedsArrayChanged	Bubbles: No Cancelable: No Context Info: none

Note: the DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the A/V Control object itself. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.14.5 Extensions to A/V Control object for playback of selected components

To support the selection of specific A/V components for playback (e.g. a specific subtitle language, audio language, or camera angle), the classes defined in Sections 7.16.5.2 – 7.16.5.5 SHALL be supported and the constants, properties and methods defined in Section 7.16.5.1 SHALL be supported on the A/V Control object.

7.14.6 Extensions to A/V Control object for parental rating errors

For parental rating errors during playback of A/V content through the CEA-2014 A/V Control object (as defined in Section 5.7.1 of [CEA-2014-A]) an OITF SHALL support the following intrinsic event properties and corresponding DOM 2 events for the CEA-2014 A/V Control object

```
function onParentalRatingChange( String contentID, ParentalRating rating, String
DRMSystemID, Boolean blocked )
```

The function that is called whenever the parental rating of the content being played inside the A/V Control object changes.

These events may occur at the start of a new content item, or during playback of a content item (e.g. during playback of linear TV content).

The specified function is called with four arguments `contentID`, `rating`, `DRMSystemID`, and `blocked` which are defined as follows:

- `String contentID` – the content ID to which the parental rating change applies. If the event is generated by the DRM system, it SHALL be the unique identifier for that content in the context of the DRM system (i.e. in the case of Marlin BB it is the Marlin contentID). Otherwise, it MAY be `null` or `undefined`.
- `ParentalRating rating` – the parental rating value of the currently playing content. The ParentalRating object is defined in Section 7.9.
- `String DRMSystemID` –the DRM System ID of the DRM system that generated the event as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. The value SHALL be `null` if the parental control is not enforced by a particular DRM system.
- `Boolean blocked` – flag indicating whether consumption of the content is blocked by the parental control system as a result of the new parental rating value.

```
function onParentalRatingError( String contentID, ParentalRating rating, String
```

DRMSystemID)
The function that is called when a parental rating error occurs during playback of A/V content inside the A/V Control object, and is triggered whenever a parental rating value is discovered for a parental rating system that is not supported by the OITF. The specified function is called with three arguments `contentID`, `rating`, and `DRMSystemID` which are defined as follows: • `String contentID` – the content ID to which the parental rating change applies. If the event is generated by the DRM system, it SHALL be the unique identifier for that content in the context of the DRM system (i.e. in the case of Marlin BB it is the Marlin contentID). Otherwise, it MAY be `null` or `undefined`. • `ParentalRating rating` – the parental rating value of the currently playing content. The ParentalRating object is defined in Section 7.9. • `String DRMSystemID` – optional argument that specifies the DRM System ID of the DRM system that generated the event as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. The value SHALL be `null` if the parental control is not enforced by a particular DRM system.

7.14.6.1 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onParentalRatingChange`	`ParentalRatingChange`	Bubbles: No Cancelable: No Context Info: `contentID`, `rating`, `DRMSystemID` and `blocked`
`onParentalRatingError`	`ParentalRatingError`	Bubbles: No Cancelable: No Context Info: `contentID, rating`, and `DRMSystemID`.

Note: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. The applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the CEA-2014 A/V embedded object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.14.7 Extensions to A/V Control object for DRM rights errors

This section SHALL apply to OITF and/or server devices which have indicated support for DRM protection by providing one or more `<drm>` elements as specified in Section 9.3.10:

For notifying Javascript about DRM licensing errors during playback of DRM protected A/V content through the CEA-2014 A/V Control object (as defined by as defined in Section 5.7.1 of CEA-2014-A) an OITF SHALL support the following intrinsic event property and corresponding DOM 2 event, for the CEA-2014 A/V Control object.

`function onDRMRightsError( Integer errorState, String contentID, String DRMSystemID, String rightsIssuerURL )`
The function that is called: • Whenever a rights error occurs for the A/V content (no license, license invalid), which has led to blocking consumption of the content. • Whenever a rights change occurs for the A/V content (license valid), which leads to unblocking the consumption of the content. This may occur during playback, recording or timeshifting of DRM protected AV content.The specified function is called with four arguments `errorState`, `contentID`, `DRMSystemID` and `rightsIssuerURL` which are defined as follows: • `Integer errorState` – error code detailing the type of error: 0: no license, consumption of the content is blocked. 1: invalid license, consumption of the content is blocked. 2: valid license, consumption of the content is unblocked. • `String contentID` – the unique identifier of the protected content in the scope of the DRM system that raises the error (i.e. in the case of Marlin BB it is the Marlin contentID, in the case of CSPG-CI+ and CSPG-DTCP this field is empty). • `String DRMSystemID` – DRMSystemID as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is "`urn:dvb:casystemid:19188`". • `String rightsIssuerURL` – optional element indicating the value of the rightsIssuerURL that can be used to non-silently obtain the rights for the content item currently being played for which this DRM error is generated, in cases whereby the `rightsIssuerURL` is known. Cases whereby the `rightsIssuerURL` is known include cases whereby the `rightsIssuerURL` has been extracted from the MPEG2_TS of the protected content, retrieved from the SD&S discovery record or from the associated BCG metadata. The corresponding `rightsIssuerURL` fields are defined in Section 4.1.3.4 of [OIPF_CSP2] and in section 3.3.2 of [OIPF_META2] respectively. If different URLs are retrieved from the stream and the metadata, then the conflict resolution is implementation-dependent.

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onDRMRightsError`	`DRMRightsError`	▪ Bubbles: No ▪ Cancelable: No ▪ Context Info: `errorState, contentID,`

		DRMSystemID, rightsIssuerURL

Note: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a `DRMRightsError` event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the CEA-2014 A/V Control object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.14.8 Extensions to A/V Control object for playing media objects

To support integration between sections 7.12, 7.4.6 and 7.4 of this specification and the A/V Control object defined in [CEA-2014-A], OITFs SHOULD add the method defined below on the A/V Control object if any of the APIs defined in those sections are supported.

`Boolean` **`setSource`**`( String id )`	
Description	Change the content item to be played by the A/V control object to the content item represented by `id`. Valid ids include: • Download identifiers (i.e. corresponding to property `Download.id`) • Recording identifiers (i.e. corresponding to property `Recording.id`) • CODAsset identifiers (i.e. corresponding to property `CODAsset.uid`) Support for each of these identifier types depends on the support for the individual sections in which they are defined. Depending on the type of content for `id`, the following semantics apply: If `id` is a download identifier, the OITF SHALL change the content item to be played to the downloaded item, or item being downloaded, for which the `Download.id` property (as defined in Section 7.4.4.2) corresponds to the given download identifier. The `type` attribute of the A/V control object SHOULD change to the MIME type of the content item represented by the download identifier, or the MIME type of the content item corresponding to the first content item listed in the Content Access Download Descriptor in case the download identifier represents a download of a Content Access Download Descriptor that contains multiple <ContentItem> elements. The `data` attribute SHALL change to the same value as the download identifier. Section 7.14.2 defines more details about playback of downloaded content, and how it relates to the states of the A/V control object. If `id` is a recording identifier, the OITF SHALL change the content item to be played to the recorded item, or item being recorded, for which the `Recording.id` property (as defined in Section 7.10.5.1) corresponds to the given recording identifier. The `type` attribute of the A/V control object SHOULD change to the MIME type of the format in which the content was recorded. The `data` attribute SHALL change to the same value as the recording identifier. If `id` is a COD asset identifier, the OITF SHALL change the content item to be played to the `CODAsset`, for which the `CODAsset.uid` property (as defined in Section 7.5.5.1) corresponds to the given COD asset identifier. The `type` attribute of the A/V control object SHOULD change to the MIME type of the COD Asset. The `data` attribute SHALL change to the same value as to COD asset identifier. If the content item represented by `id` can be accepted by the A/V control object for

	playback, the method returns `true`. The method returns `false` if the item cannot be accepted for playback.	
Arguments	*id*	The ID of the item to be played.

7.14.9 Extensions to A/V Control object for UI feedback of buffering A/V content

The following additional properties and methods SHALL be supported on audio and video objects as defined in Section 5.7.1 of [CEA-2014-A].

7.14.9.1 Properties

`function` **`onReadyToPlay`**`()`
The function that gets called when enough (as determined by the OITF) of the media after the current play position has been buffered to start/continue playback. The specified function shall be called with no arguments. This event SHALL be generated whenever there is a state transition between state 4 (”`buffering`”) and state 1 (”`playing`”). The event SHALL also be generated at the moment that enough data has been buffered to start playback, whilst in state 2 (”`paused`”).

`Boolean` **`readyToPlay`**
Property that can be used to inspect whether or not enough (as determined by the OITF) of the media after the current play position has been buffered to start playback. Returns `true` if enough data has been buffered. Returns `false` if not enough data has been buffered.

7.14.9.2 Methods

`Integer` **`getAvailablePlayTime`**`( Boolean fromPlayPosition )`		
Description	Returns how much content is available for playback. If argument `fromPlayPosition` has value `true`, this method returns an estimate of how much data in milliseconds is available in the buffer for play back after the current play position. If argument `fromPlayPosition` has value `false`, this method returns an estimate of the total buffer length in milliseconds (i.e. this includes all data available in the buffer before and after the current play position).	
Arguments	*fromPlayPosition*	Indicates whether the available play time should be calculated from the current play position onwards, or from the start of the buffer.

Boolean **setBufferingStrategy**(String name)	
Description	Request to change the buffering strategy. Valid values for argument name include: "sustained_playback": this is the default strategy, whereby the incoming video stream should be rendered with as little hiccups or lost frames as possible. This means that the buffering threshold for triggering an onReadyToPlay event is chosen to be sufficiently large to deal with variations in network throughput. "low_latency": this is a strategy whereby the incoming video stream should be rendered with an as low as possible latency between receiving the content and the actual playback of the content. This means that buffering threshold for triggering an onReadyToPlay event needs to be made sufficiently small in order to playback the content as soon as possible after it has been received. The default strategy if the method is not called is "sustained_playback". This method can be called during any play state, including play state 1 ('playing'). This method returns true if the buffering strategy has been successfully changed to the preferred buffering strategy. The method returns false if the buffering strategy has not been successfully changed. If the OITF does not distinguish between the two modes, the method returns false.

7.14.9.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onReadyToPlay	ReadyToPlay	Bubbles: No Cancelable: No Context Info: None

Note: these DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the addEventListener() method on the CEA-2014 A/V Control object. The third parameter of addEventListener, i.e. "useCapture", will be ignored.

7.14.10 Extensions to A/V Control object for volume control

7.14.10.1 Methods

The following additional method SHALL be supported on the audio object and video object defined in Section 5.7.1 of [CEA-2014-A].

Integer **getVolume**()	
Description	Returns the actual volume level set; for systems that do not support individual volume control of players, this method will have no effect and will always return 100.

7.14.11 DOM 2 events for A/V Control object

To make the A/V Control object as defined in CEA-2014-A in line with the other scripting objects in section 7 of this specification, for the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onfocus	focus (as specified in Section 1.6.5 of [DOM 2 Events])	Bubbles: No Cancelable: No Context Info: None
onblur	blur (as specified in Section 1.6.5 of [DOM 2 Events])	Bubbles: No Cancelable: No Context Info: None
onPlayStateChange	PlayStateChange	Bubbles: No Cancelable: No Context Info: None
onFullScreenChange	FullScreenChange	Bubbles: No Cancelable: No Context Info: None

Note: these DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving these events during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the addEventListener() method on the CEA-2014 A/V Control object. The third parameter of addEventListener, i.e. "useCapture", will be ignored.

7.14.12 Playback of memory audio

This section describes specific usage of A/V media object corresponding to memory audio

7.14.12.1 Usage of CE-HTML tags

An <object> element which corresponds to a memory audio SHALL apply following restrictions:

1) The type attribute SHALL be included to define the MIME type that matches the memory audio referred to by the value of the data attribute. The MIME types for the memory audio SHALL adhere to Section 8.2.1 of [OIPF_MEDIA2].

2) The file extensions that SHALL be used for the memory audio are:

- "`.aac`" for **HE-AAC** memory audio.
- "`.wav`" for **WAVE** memory audio.

Only in the case of HE-AAC memory audio, an <object> element MAY contain a `<param>` element to set the `loop` parameter. This parameter indicates the number of times the HE-AAC memory audio will play when `play()` is called. The value SHALL be positive integers or the case sensitive string "`infinite`", which will play the memory audio continuously until `stop()` is called or the data attribute is set to `null`. The default value of this parameter is "1". To give DAE a hint to pre-fetch memory audio from the server when the CE-HTML document is loaded, a `<link>` element MAY be used whereby:

7) The `rel` attribute SHALL be set to a value "`prefetch`" and the "`href`" attribute SHALL be set to the URL of the memory audio which is expected to be pre-fetched. The OITF MAY pre-fetch the audio file referred to by the `href` attribute, but is not required to do so.

7.14.12.2 Usage of the DOM interface

The <object> element as defined in Section 7.14.12.1 of this document SHALL be made accessible through the Javascript A/V Control object specified in [CEA-2014-A], in the following manner:

1) The following attributes SHALL be supported: `data`, `playState`, `error` and `onPlayStateChange`, as defined in Req. 5.7.1.f of [CEA-2014-A].

Following methods SHALL be supported: `play()` and `stop()`, as defined in Req. 5.7.1.f of [CEA-2014-A]. The <param> element as defined in Section 7.14.12.1 of this document SHALL be made accessible through the `HTMLParamElement`.

7.14.12.3 DAE requirements

If the `data` attribute of the `<object>` element for memory audio is set to a valid value and `type` attribute of the `<object>` element indicates the format being HE-AAC, DAE SHALL play the memory audio, as specified below

If the `data` attribute of the <object> element for memory audio is set to a valid value and `type` attribute of the <object> element indicates the format being WAVE, DAE MAY play the memory audio, as specified in bullets 1) and 2) below.

1) When the `play()` method is called by script, it SHALL start the playback of the memory audio designated by the `data` attribute. If the audio file cannot be loaded because of insufficient memory in the OITF, calls to `play()` SHALL cause the A/V Control object to move to state 6 (the error state) with the `error` property set to 3 ("insufficient resources").

2) When the `stop()` method is called or the `data` attribute is set to `null` by a script, OITF SHALL stop the playback of the memory audio which had previously played.

If the `rel` attribute of the `<link>` element is set to a value "`prefetch`", the OITF MAY pre-fetch the memory audio referred by the `href` attribute of the `<link>` element, when the HTML document is loaded to the OITF.

An HE-AAC memory audio need not to be played simultaneously with other HE-AAC memory audio or streamed A/V contents defined in Section 5.7.1 of [CEA-2014-A].

7.14.12.4 Example usage (Informative)

The following HTML document shows an example of a script to start the playback of memory audio:

```
<head>
 :
<script type="text/javascript">
     function startBGM() {
         document.getElementById("aid1").play(1);
     }
     :
</script>
</head>
<body>
<object type="audio/mp4" id="aid1" data="http://www.avsource.com/audio/bgm.aac">
<param name="loop" value="infinite"/>
</object>
 :
<div id="btn1" onclick=" startBGM()"><img src="start1.gif" /></div>
```

```
  :
</body>
```

The following HTML document shows an example of a script to stop the playback of memory audio:

```
<head>
  :
<script type="text/javascript">
     function stopBGM() {
         document.getElementById("aid1").stop();
     }
  :
</script>
</head>
<body>
<object type="audio/mp4" id="aid1" data="http://www.avsource.com/audio/bgm.aac">
<param name="loop" value="infinite"/>
</object>
  :
<div id="btn2" onclick=" stopBGM()"><img src="stop1.gif" /></div>
  :
</body>
```

7.15 Miscellaneous APIs

7.15.1 The application/oipfMDTF embedded object

If an OITF has indicated support for the multicast delivery terminating function (MDTF) (i.e., `<mdtf>true</mdtf>`) as defined in Section 9.3.15 in its capability description, the OITF SHALL support MDTF through the use of the following non-visual object:

```
<object type="application/oipfMDTF"/>
```

The MDTF API provides the necessary javascript methods to indicate to the MDTF what FLUTE multicast channel it should join, and what tags it should listen for on those channels.

7.15.1.1 Properties

`function` **`onFLUTEListenerResult`**`( String multicastAddress, Integer resultMsg )`

This function is called with return result from the methods `addFLUTEListener` and `removeFLUTEListener`.

The specified script function is called with 2 arguments – `multicastAddress` and `resultMsg`.

- `String multicastAddress` – The multicast address associated with the callback.
- `Integer resultMsg` – result message. Valid values include:

Result message	Description	Semantics
0	Successful	The action performed by the underlying functionality was successful.
1	Unknown error	The action performed by the underlying functionality failed because an unspecified error occurred.
2	Invalid multicast address	The multicast address is not valid, e.g. bad syntax or out of range.
3	Multicast address	The multicast address does not exist in the listener table.

	does not exist	
4	No resources	There was not enough resources in the OITF to join the multicast address (only valid for `addFLUTEListener`).

7.15.1.2 Methods

`void` **`addFLUTEListener`**`( String multicastAddress )`		
Description	This method adds a FLUTE channel listener in the OITF. The result from this method is sent to the callback method `onFLUTEListenerResult`.	
Arguments	*multicastAddress*	The multicast address that the OITF should join in order to listen.

`void` **`addFLUTEListenerTags`**`( String multicastAddress, String tags, String downloadCallback)`		
Description	This method adds tags that the FLUTE listener should listen for. The result from this method is sent to the callback method `onFLUTEListenerResult`.	
Arguments	*multicastAddress*	The multicast address that the OITF should join in order to listen.
	tags	A comma separated list of tags that the OITF should listen for on the FLUTE channel.
	downloadCallback	Optional. This callback function is called when an object has been downloaded. The arguments to this function are the Content Location URI of the downloaded object and the Content-Type.

`StringCollection` **`getFLUTEListeners`**`()`	
Description	Returns a collection of multicast addresses for the FLUTE channels that the OITF listens to.

`String` **`getTags`**`( String multicastAddress )`	
Description	Returns a comma-separated list of the tags associated with a particular multicast address.

`void` **`removeFLUTEListener`**`( String multicastAddress )`	
Description	Removes the associated listener. The result from this method is sent to the callback method `onFLUTEListenerResult`.

Arguments	*multicastAddress*	The multicast address that the OITF should leave.

7.15.1.3 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onFLUTEListenerResult	FLUTEListenerResult	Bubbles: No Cancelable: No Context Info: `multicastAddress`, `resultMsg`

NOTE: the above DOM 2 event is directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving a `ReceiveRemoteMessage` event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the application/oipfRemoteControlFunction object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

7.15.2 The application/oipfStatusView embedded object

7.15.2.1 Overview of download status

The following embedded objects allow a visualization of the native download manager to be included as part of the UI coming from a (third party) server, without the need for any security model, and without compromising security and privacy.

An OITF SHALL support the `application/oipfStatusView` embedded object. This is a visual object that can be included in a HTML document, and is subject to the following CSS-properties: `width`, `height`, `position`, `float`, `top`, `left`, `right`, `bottom`, `vertical-align`, `padding`, and `padding-*` properties, `margin`, and `margin-*` properties, `border`, and `border-*` properties, `visibility`, and `display`. This embedded object SHALL provide an overall consistent graphical view of the status of the current downloads, the content that has been downloaded, and/or the content that has been recorded, as denoted by the states:

- "`list_of_recent_downloads`": shows the progress of the most recently started downloads, with the amount of items shown as specified by `<param>` element with the name "`nritems`".
- "`list_of_downloaded_content`": shows the list of items that have been successfully downloaded, with the amount of items shown as specified by `<param>` element with the name "`nritems`".

The object SHALL support a `<param>` element with the name "`state`", which indicates the state that SHALL be visualized inside the object. An OITF that has indicated support for downloading content in its capability description (i.e. `<download>true</download>`) SHALL at least support the monitor states "`list_of_recent_downloads`" and "`list_of_downloaded_content`". An OITF MAY support the visualization of additional states. An OITF SHALL silently ignore a request to visualize a state that it does not support; if this results in no state information being visualized at all (because the each `<param>` element with name state referred to a non-supported state), the `application/oipfStatusView` object SHALL NOT be visualized and the object will have CSS `width` and `height` values of 0.

The object SHALL support a <param> element with the name "nritems", which indicates the number of items that should be shown for the given state.

The object SHALL also support the inclusion of style hints through <param> elements. At least the "background-color" and "font-size" style hints SHALL be supported using the syntax defined by CSS 2.1. An OITF MAY support additional style hints in addition to "background-color" and "font-size". Additional style hints SHALL also follow the CSS 2.1 syntax. An OITF SHALL silently ignore any style hints that it does not support.

Next to these parameters, the object SHALL support methods getMinimumItemWidth() and getMinimumItemHeight() as defined in Section 7.15.2.1.1.

Example usage:

```
<object id="d1" type="application/oipfStatusView" width="200" height="100">
    <param name="state" value="list_of_recent_downloads"/>
    <param name="nritems" value="2"/>
    <param name="background-color" value="black"/>
    <param name="font-size" value="16px"/>
</object>
```

NOTE: this object is intended to allow services to link in to the privileged functionality of accessing privacy sensitive download information, without the need for certificates and privileged access requests. In certain managed network deployments this may not be sufficient. The application/oipfDownloadManager API described in Section 7.4.3 provides more extensive APIs which provide Javascript control for a service platform provider over such highly privileged functionality.

7.15.2.1.1 Methods

Integer **getMinimumItemWidth**(String state)		
Description	Returns the minimum width needed for rendering the name, status and other data of the downloaded items for the given state (e.g. "list_of_recent_downloads").	
Arguments	*state*	The state for which the visualization is requested. This is one of the strings that are defined for <param> element with the name "state" (e.g. "list_of_recent_downloads").

Integer **getMinimumItemHeight**(String state)		
Description	Returns the minimum height needed for rendering the name, status and other data of the downloaded items for the given state (e.g. "list_of_recent_downloads").	
Arguments	*state*	The state for which the visualization is requested. This is one of the strings that are defined for <param> element with the name "state" (e.g. "list_of_recent_downloads").

7.15.2.2 Overview of recordings

An OITF that has indicated support for control of its recording functionality by a server (i.e., <record>true</record>) SHALL support the application/oipfStatusView embedded object defined in Section 7.15.2.1, for which it SHALL at least support the following additional monitor state:

- "list_of_recorded_content": shows the list of items that have been recorded or that are currently being recorded, with the amount of items shown as specified by <param> element with the name "nritems".

NOTE: this object is intended to allow services to link in to highly privileged functionality, without the need for certificates and privileged access requests. In certain managed network deployments this may not be sufficient.

Therefore, section 7.10.4 defines more extensive APIs which provide Javascript control for a service platform provider over such highly privileged functionality.

7.15.3 The application/oipfCapabilities embedded object

The OITF SHALL support following non-visual embedded object with the mime type `application/oipfCapabilities`.

7.15.3.1 Properties

`readonly Document` **`xmlCapabilities`**
Returns the OITF's capability description as an XML Document object using the syntax as defined in Annex F without using any namespace definitions.

`readonly Number` **`extraSDVideoDecodes`**
This property holds the number of possible additional decodes for SD video. Depending on the current usage of system resources this value may vary. The value of this property is likely to change if an HD video is started. Adding an A/V Control object or video/broadcast object may still fail, even if extraSDVideoDecodes is larger than 0. For A/V Control objects, in case of failure the play state for the A/V Control object shall be set to 6 (error) with a detailed error code of 3 ('insufficient resources'). For video/broadcast objects, in case of failure the play state of the A/V Control object shall be set to 0 (unrealized) with a detailed error code of 11 ('insufficient resources').

`readonly Number` **`extraHDVideoDecodes`**
This property holds the number of possible additional decodes for HD video. Depending on the current usage of system resources this value may vary. The value of this property is likely to change if an SD video is started. Adding an A/V Control object or video/broadcast object may still fail, even if extraHDVideoDecodes is larger than 0. For A/V Control objects, in case of failure the play state for the A/V Control object shall be set to 6 (error) with a detailed error code of 3 ('insufficient resources'). For video/broadcast objects, in case of failure the play state of the A/V Control object shall be set to 0 (unrealized) with a detailed error code of 11 ('insufficient resources').

7.15.3.2 Methods

`Boolean` **`hasCapability`**`( String profileName )`		
Description	Check if the OITF supports the passed capability. Returns `true` if the OITF supports the passed capability, `false` otherwise.	
Arguments	*profileName*	An OIPF base UI profile string or a UI Profile name fragment string as defined in Section 9.2.

		Examples of valid values are “OITF_HD_UIPROF” or “+PVR”.

7.15.4 The Navigator class

The Navigator object represents the identity of the OITF. This is intended to be equivalent to the Navigator interface as defined in section 6.8 of [HTML5].

7.15.4.1 Properties

readonly String **appName**
Returns the name of the browser. If supported, this corresponds to the <appName> element in the user-agent header as defined in Section 8.1.1. Otherwise, it SHALL be the empty string.

readonly String **appVersion**
Returns the version of the browser. If supported, this corresponds to the <appVersion> element in the user-agent header as defined in Section 8.1.1. Otherwise, it SHALL be the empty string.

7.15.5 Debug print API

The following method is available on the global (Window) object.

void **debug**(DOMString arg)		
Description	Let the application developer print debug information on the debug output (for example, a console, a serial link or a file). The means to access this debug output is outside the scope of this specification and implementation-dependent. A line feed character SHALL NOT be inserted automatically at the end of the string by the implementation. Example: `debug("[APP] value = " + value + "\n");`	
Arguments	*arg*	String to print on the debug output.

7.16 Shared Utility classes and features

7.16.1 The StringCollection class

```
typedef Collection<String> StringCollection
```

The StringCollection class represents a collection of String objects. See annex K for the definition of the collection template.

7.16.2 The Programme class

The `Programme` class represents an entry in a programme schedule.

Note: as described in the `record( Programme programme )` method of the `application/oipfRecordingScheduler` object, only the `programmeID` property of the programme object is used to determine the programme or series that will be recorded. The other properties are solely used for annotation of the (scheduled) recording with programme metadata. The use of these metadata properties is optional. If such programme metadata is provided, it is retained in the `ScheduledRecording` object that is returned if the recording of the programme was scheduled successfully.

7.16.2.1 Constants

The following constants are defined as properties on the `Programme` class.

Name	Value	Use
`ID_TVA_CRID`	0	Used in the `programmeIDType` property to indicate that the programme is identified by its TV-Anytime CRID (Content Reference Identifier).
`ID_DVB_EVENT`	1	Used in the `programmeIDType` property to indicate that the programme is identified by a DVB URL referencing a DVB-SI event as enabled by section 4.1.3 of [OIPF_META2]. OPTIONAL.

7.16.2.2 Properties

`String` **`name`**
The short name of the programme, e.g. 'Star Trek: DS9'.

`String` **`longName`**
The long name of the programme, e.g. 'Star Trek: Deep Space Nine'. If the long name is not available, this property will be `undefined`.

`String` **`description`**
The description of the programme, e.g. an episode synopsis. If no description is available, this property will be `undefined`.

`String` **`longDescription`**
The long description of the programme. If no description is available, this property will be `undefined`.

`Integer` **`startTime`**
The start time of the programme, measured in seconds since midnight (GMT) on 1/1/1970.

Integer **duration**
The duration of the programme (in seconds).

String **channelID**
The identifier of the channel from which the broadcasted content is to be recorded. Specifies either a ccid or ipBroadcastID (as defined by the Channel object in Section 7.13.12)

Integer **episode**
The episode number for the programme if it is part of a series. This property is undefined when the programme is not part of a series or the information is not available.

Integer **totalEpisodes**
If the programme is part of a series, the total number of episodes in the series. This property is undefined when the programme is not part of a series or the information is not available.

String **programmeID**
The unique identifier of the programme or series, e.g., a TV-Anytime CRID (Content Reference Identifier).

Integer **programmeIDType**
The type of identification used to reference the programme, as indicated by one of the ID_* constants defined above.

readonly ParentalRatingCollection **parentalRating**
A collection of parental rating values for the programme for zero or more parental rating schemes supported by the OITF. For instances of the Programme class created by the createProgramme() method defined in section 7.10.1.1, the initial value of this property (upon creation of the Programme object) is an instance of the ParentalRatingCollection object (as defined in Section 7.9.5) with length 0. Parental rating values can be added to this empty readonly parental rating collection by using the addParentalRating() method of the ParentalRatingCollection object. The ParentalRatingCollection is defined in Section 7.9.5. The related ParentalRating and ParentalRatingScheme objects are defined in Section 7.9.4 and 7.9.2 respectively. For instances of the Programme class returned through the metadata APIs defined in section 7.12 or through the programmes property of the video/broadcast object defined in section 7.13.3, the initial value of this property SHALL include the parental rating value(s) carried in the metadata or DVB-SI entry describing the programme, if this information is included. Note that if the service provider specifies a certain parental rating (e.g. PG-13) through this property and the actual parental rating extracted from the stream says that the content is rated PG-16, then the

conflict resolution is implementation dependent.

7.16.2.3 Metadata extensions to Programme

The OITF SHALL extend the `Programme` class defined in section 7.16.2 with the properties and methods described below.

This subsection SHALL apply for OITFs that have indicated `<clientMetadata>` with value "`true`" and a `type` attribute with values "`bcg`", "`eit-pf`" or "`dvb-si`" as defined in Section 9.3.7 in their capability profile.

7.16.2.3.1 Properties

`readonly Channel` **`channel`**

Reference to the broadcast channel where the programme is available.

The value of this field is derived from the `serviceIDref` attribute of the `Schedule` element that refers to this programme.

`readonly Boolean` **`blocked`**

Flag indicating whether the programme is blocked due to parental control settings or conditional access restrictions.

The `blocked` and `locked` properties work together to provide a tri-state flag describing the status of a programme. This can best be described by the following table:

Description	blocked	locked
No parental control applies.	`false`	`false`
Item is above the parental rating threshold (or manually blocked); no PIN has been entered to view it and so the item cannot currently be viewed.	`true`	`true`
Item is above the parental rating threshold (or manually blocked); the PIN has been entered and so the item can be viewed.	`true`	`false`

`readonly Integer` **`showType`**

Flag indicating the type of show (live, first run, rerun, etc,).

The value of this property is determined by the child elements of the programme's `BroadcastEvent` or `ScheduleEvent` element from the Program Location Table. Values are determined as follows:

Value	Description
1	The programme is live; indicated by the presence of a `Live` element with a value attribute set to true.
2	The programme is a first-run show; indicated by the presence of a `FirstShowing` element with a value attribute set to true.

3	The programme is a rerun; indicated by the presence of a `Repeat` element with a value attribute set to true.

If none of the above conditions are met, the default value of this field SHALL be 2.

`readonly Boolean` **`subtitles`**

Flag indicating whether subtitles or closed-caption information is available.

This flag SHALL be `true` if one or more BCG `CaptionLanguage` elements are present in this programme's description, `false` otherwise.

`readonly Boolean` **`isHD`**

Flag indicating whether the programme has high-definition video.

This flag SHALL be `true` if a `VerticalSize` element is present in the programme's description and has a value greater than 576, `false` otherwise.

`readonly Integer` **`audioType`**

Bitfield indicating the type of audio that is available for the programme.

The value of this field is determined by the `NumOfChannels` elements in a programme's A/V attributes. Values are determined as follows:

Value	Description
1	A mono audio stream is available (at least one `AvAttributes.AudioAttributes` element is present which has a child `NumOfChannels` element whose value is 1).
2	A stereo audio stream is available (at least one `AvAttributes.AudioAttributes` element is present which has a child `NumOfChannels` element whose value is 2).
4	A multi-channel audio stream is available (at least one `AvAttributes.AudioAttributes` element is present which has a child `NumOfChannels` element whose value is greater than 2).

For programmes with multiple audio streams, these values may be ORed together.

`readonly Boolean` **`isMultilingual`**

Flag indicating whether more than one audio language is available for the programme.

This flag SHALL be `true` if more than one BCG `Language` element is present in the programme's description, `false` otherwise.

`readonly StringCollection` **`genre`**

A collection of genres that describe this programme. The value of this field is the concatenation of the values of any `Name` elements that are children of `Genre` elements in the programme's description.

`readonly Boolean` **`hasRecording`**
Flag indicating whether the `Programme` has a recording associated with it (either scheduled, in progress, or completed).

`readonly StringCollection` **`audioLanguages`**
Supported audio languages, indicated by iso639 language codes.

`readonly StringCollection` **`subtitleLanguages`**
Supported subtitle languages, indicated by iso639 language codes.

`readonly Boolean` **`locked`**
Flag indicating whether the current state of the parental control system prevents the programme from being viewed (e.g. a correct parental control PIN has not been entered to allow the programme to be viewed).

7.16.2.3.2 Methods

`String` **`getField`**`( String fieldId )`		
Description	Get the value of the field referred to by `fieldId` that is contained in the metadata for this programme. If the field does not exist, this method SHALL return `undefined`.	
Arguments	*fieldId*	The name of the field whose value SHALL be retrieved.

7.16.2.4 DVB-SI extensions to Programme

The following method SHALL be added to the `Programme` object, if the OITF has indicated support for accessing DVB-SI information, by giving the value "`true`" to element `<clientMetadata>` and the value "`dvb-si`" or "`eit-pf`" to the `type` attribute of that element as defined in Section 9.3.7 in their capability profile.

`StringCollection` **`getSIDescriptors`**`( Integer descriptorTag, Integer descriptorTagExtension )`	
Description	Get the contents of the descriptor specified by `descriptorTag` from the DVB SI EIT programme's descriptor loop. If more than one descriptor with the specified tag is available for the given programme, the contents of all matching descriptors SHALL be

	returned in the order the descriptors are found in the stream. The descriptor content bytes SHALL be encoded in a string whose characters shall be restricted to the ISO Latin-1 character set. Each character in the string represents a byte of a DVB-SI descriptor, such that a byte at position "i" in the descriptor is equal the Latin-1 character code of the character at position "i" in the string. Described in the syntax of ECMAScript: let desc[] be the byte array of a descriptor, in which desc[0] is the descriptor_tag, then, the returned string (retval in the example below) is its equivalent string, if : `desc.length==retval.length` and `for each integer i : 0<=i<desc.length` holds `desc[i] == retval.charCodeAt(i)`. If the descriptor specified by `descriptorTag` and (optionally) `descriptorTagExtension` does not exist, or if the metadata for this programme was retrieved from a source other than DVB-SI, this method SHALL return `null`. If metadata for this programme has not yet been retrieved, this method SHALL return `undefined`. If the OITF supports the `application/oipfSearchManager` object as defined in Section 7.12.1, the OITF SHALL notify applications of the availability of additional metadata via `MetadataSearchEvents` targeted at the `application/oipfSearchManager` object used to retrieve the programme metadata.	
Arguments	*descriptorTag*	The descriptor tag as specified by [EN 300 468].
	descriptorTagExtension	An optional argument giving the descriptor tag extension as specified by [EN 300 468].

7.16.2.5 Recording extensions to Programme

The OITF SHALL support the following extensions to the `Programme` class.

Clients supporting the recording management APIs defined in this section SHALL indicate this by adding the attribute `manageRecordings` to the <`recording`> element with a value unequal to ”`none`” in the client capability description as defined in section 9.3.3.

The functionality as described in this section is subject to the security model of Section 10.

`readonly ScheduledRecording` **`scheduledRecording`**
If available, this property represents the scheduled recording associated with this programme. Has value undefined if this programme has no scheduled recording associated with it.

`readonly RecordingCollection` **`recordings`**
The list of in-progress or completed recordings associated with this programme, sorted by start time in increasing order.

7.16.3 The ProgrammeCollection class

```
typedef Collection<Programme> ProgrammeCollection
```

The `ProgrammeCollection` class represents a collection of `Programme` objects. See annex K for the definition of the collection template.

7.16.4 The DiscInfo class

The `DiscInfo` class provides details of the storage usage and capacity in the OITF.

7.16.4.1 Properties

`readonly Integer` **free**
The space (in megabytes) available on the storage device for recordings.

`readonly Integer` **total**
The total capacity (in megabytes) of the storage device. Depending upon the system, free MAY be less than total even with no recordings as some of the disc space MAY be used for management purposes.

`readonly Integer` **reserved**
The space (in megabytes) reserved for scheduled or ongoing recordings and downloads.

7.16.5 Extensions for playback of selected media components

This section defines APIs for the selection of specific A/V components for playback.

NOTE: The term component may correspond to MPEG_2 components, but is not restricted to that.

7.16.5.1 Media playback extensions

7.16.5.1.1 Constants

The following constants are defined as properties on any objects implementing this section:

Name	Value	Use
COMPONENT_TYPE_VIDEO	0	Represents a video component. This constant is used for all video components regardless of encoding.
COMPONENT_TYPE_AUDIO	1	Represents an audio component. This constant is used for all audio components regardless of encoding.
COMPONENT_TYPE_SUBTITLE	2	Represents a subtitle component. This constant is used for all subtitle components regardless of subtitle format. NOTE: A

Name	Value	Use
		subtitle component may also be related to closed captioning as part of a video stream.

7.16.5.1.2 Properties

`function` **`onSelectedComponentChanged`**`( Integer componentType )`
This function is called when there is a change in the set of components being presented. This may occur if one of the currently selected components is no longer available and an alternative is chosen based on user preferences, or when presentation has changed due to a different component or set of components being selected. OITFs MAY optimise event dispatch by dispatching a single event in response to several calls to `selectComponent()` or `unselectComponent()` made in rapid succession. The specified function is called with one argument: • `Integer componentType` - The type of component whose presentation has changed, as represented by one of the constant values listed in section 0. If more than one component type has changed, this argument will take the value `undefined`.

7.16.5.1.3 Methods

`AVComponentCollection` **`getComponents`**`( Integer componentType )`		
Description	Returns a collection of `AVComponent` values representing the components of the specified type in the current stream. If componentType is set to null or undefined then all the currently active components are returned. One or more of the components returned MAY be passed back to one of the other methods unchanged (e.g. `selectComponent()`). If property `preferredAudioLanguage` in the `Configuration` object (refer to section 7.3.2.1) is set then a component is by default selected and is considered as an active component. If property `preferredSubtitleLanguage` in the `Configuration` object (refer to section 7.3.2.1) is set and property `subtitleEnabled` in `AVOutput` class (refer to section 7.3.5.1) is enabled then a component is by default selected and is considered as an active component.	
Argument	*componentType*	The type of component to be returned , as represented by one of the constant values listed in section 0.

`AVComponentCollection` **`getCurrentActiveComponents`**`( Integer componentType )`	
Description	Returns a collection of `AVComponent` values representing the currently active components of the specified type that are being rendered. One or more of the components returned MAY be passed back to one of the other

	methods unchanged (e.g. `selectComponent()`).	
Argument	*componentType*	The type of currently active component to be returned. represented by one of the constant values listed in section 0.

`void` **`selectComponent`**`( AVComponent component )`		
Description	Select the component that will be subsequently rendered when A/V playback starts or select the component for rendering if A/V playback has already started. If playback has started, this SHALL replace any other components of the same type that are currently playing. If property `preferredAudioLanguage` in the `Configuration` object (refer to section 7.3.2.1) is set then a component is by default selected and it is not necessary to perform `selectComponent()`. If property `preferredSubtitleLanguage` in the `Configuration` object (refer to section 7.3.2.1) is set and property `subtitleEnabled` in `AVOutput` class (refer to section 7.3.5.1) is enabled then a component is by default selected and it is not necessary to perform `selectComponent()`.	
Argument	*component*	A component object available in the stream currently being played.

`void` **`unselectComponent`**`( AVComponent component )`		
Description	Stop rendering of the specified component of the stream. If property `preferredAudioLanguage` in the `Configuration` object (see section 7.3.2.1) is set then unselecting a specific component returns to the default preferred audio language. If property `preferredSubtitleLanguage` in the `Configuration` object (see section 7.3.2.1) is set and property `subtitleEnabled` in `AVOutput` class (see section 7.3.5.1) is enabled then unselecting a specific component returns to the default preferred subtitle language. In order to stop rendering subtitles completely it is necessary to disable subtitles with property `subtitleEnabled` in `AVOutput` class.	
Argument	*component*	The component to be stopped.

`void` **`selectComponent`**`( Integer componentType )`		
Description	If A/V playback has already started, start rendering the default component of the specified type in the current stream. This SHALL replace any other components of the same type that are currently playing. If A/V playback has not started, the default component of the specified type will be subsequently rendered after calling the `setChannel` method on the `video/broadcast` object.	
Argument	*componentType*	The type of component for which the default component should be rendered.

void **unselectComponent**(Integer componentType)		
Description	If A/V playback has already started, stop rendering of the specified type of component. If A/V playback has not started, no components of the specified type will be subsequently rendered after calling the `setChannel` method on the `video/broadcast` object.	
Argument	*component*	The type of component to be stopped.

7.16.5.1.4 Events

For the intrinsic event "`onSelectedComponentChange`", corresponding DOM level 2 events SHALL be generated, in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
`onSelectedComponentChange`	`SelectedComponentChange`	Bubbles: No Cancelable: No Context Info: `componentType`

7.16.5.2 The AVComponent class

`AVComponent` represents a component within a complete media stream - a single stream of video, audio or data that can be played or manipulated. This is not necessary for basic playback, record or EPG services. However, it provides a mechanism to get at extended streams for enhanced services.

For forward compatibility the DAE application SHALL check the value of the `type` property to ensure that it is accessing an `AVComponent` object of the correct type.

7.16.5.2.1 Properties

readonly Integer **componentTag**
The component tag identifies a component. The component tag identifier corresponds to the component_tag in the component descriptor in the ES loop of the stream in the PMT [EN 300 468], or `undefined` if the component is not carried in an MPEG-2 TS .

readonly Integer **pid**
The MPEG Program ID (PID) of the component in the MPEG2-TS in which it is carried, or `undefined` if the component is not carried in an MPEG-2 TS.

readonly Integer **type**
Type of the component stream. Valid values for this field are given by the constants listed in section 7.16.5.1.1.

readonly String **encoding**
The encoding of the stream. The value of video format or audio format defined in section 3 of [OIPF_MEDIA2] SHALL be used.

readonly Boolean **encrypted**
Flag indicating whether the component is encrypted or not.

7.16.5.3 The AVVideoComponent class

The AVVideoComponent class implements the AVComponent interface.

7.16.5.3.1 Properties

readonly Number **aspectRatio**
Indicates the aspect ratio of the video or undefined if the aspect ratio is not known. Values SHALL be equal to width divided by height, rounded to a float value with two decimals, e.g. 1.78 to indicate 16:9 and 1.33 to indicate 4:3.

7.16.5.4 The AVAudioComponent class

The AVAudioComponent class implements the AVComponent interface.

7.16.5.4.1 Properties

readonly String **language**
An ISO 639 language code representing the language of the stream.

readonly Boolean **audioDescription**
Has value true if the stream contains an audio description intended for people with a visual impairment, false otherwise.

readonly Integer **audioChannels**
Indicates the number of channels present in this stream (e.g. 2 for stereo, 5 for 5.1, 7 for 7.1).

7.16.5.5 The AVSubtitleComponent class

The AVSubtitleComponent class implements the AVComponent interface

7.16.5.5.1 Properties

readonly String **language**
An ISO 639 language code representing the language of the stream.

readonly Boolean **hearingImpaired**
Has value true if the stream is intended for the hearing-impaired (e.g. contains a written description of the sound effects), false otherwise.

7.16.5.6 The AVComponentCollection class

```
typedef Collection<AVComponent> AVComponentCollection
```

An `AVComponentCollection` represents a collection of `AVComponentCollection` objects. See annex K for the definition of the collection template.

7.17 DLNA RUI Remote Control Function APIs

This section defines the APIs related to the DLNA RUI RCF.

The DLNA RUI RCF APIs provide the necessary javascript properties and methods for a DAE application to communicate with Remote Control Devices and provide a Control UI (i.e. one or more CE-HTML documents that enable the DAE application to be controlled from the Remote Control Device) on such devices. Using these APIs, Remote Control Devices can:

- obtain a Control UI from the OITF or the IPTV Applications server via the OITF,
- send information such as control messages to the OITF and
- receive information from the OITF.

This section SHALL apply for OITFs that have indicated `<remoteControlFunction>` with value "`true`" as defined in Section 9.3.17 in its capability description.

7.17.1 The application/oipfRemoteControlFunction embedded object

OITFs that have indicated <remoteControlFunction> with value "`true`" SHALL support the DLNA RUI RCF APIs through the use of the following non-visual embedded object:

```
<object type="application/oipfRemoteControlFunction"/>
```

7.17.1.1 Constants

The following constants are defined as properties of the `application/oipfRemoteControlFunction` embedded object:

Constant name	Numeric Value	Use
`REQUEST_CUI`	0	A Remote Control Device (a Control UI or an XML UI Listing) requests a control UI by using the pre-defined URI "`/rcf/request_cui`".
`REQUEST_MSG`	1	A Control UI in the Remote Control Device sends a message by using the pre-defined URI "`/rcf/request_msg`".

CREATE_APP	2	A Control UI in the Remote Control Device sends a message by using a URI defined by an OITF. This message has triggered the application receiving this event to be launched by the OITF.

7.17.1.2 Properties

`readonly Integer` **currentRemoteDeviceHandle**

The handle of the Remote Control Device which is associated with the DAE application in the mapping information table (see Section 8.4.6) and is waiting for the response from the DAE application. The value of this handle is assigned by the OITF, and is unique within the OITF for the duration of a session (the duration of the connection between the OITF and that Remote Control Device). Applications SHALL NOT rely on the value of this handle being preserved across sessions.

This property is retrieved from the mapping information table (see Section 8.4.6) in the OITF which contains the pairing information between the Remote Control Device and the DAE application. Only one Remote Control Device is allowed to connect to a given DAE application at a time.

If there is no mapping information between a Remote Control Device and the DAE application, this property returns undefined.

`readonly String` **currentRemoteDeviceUA**

The Remote Control Device User-Agent string that has been provided in the Remote Control Device's HTTP request.

The `application/oipfRemoteControlFunction` object stores the value of the User-Agent header included in the most recent HTTP request of the Remote Control Device currently being connected to this DAE application.

Note: The User-Agent string of the Remote Control Device is expected to conform to the format of the User-Agent string defined in [Req. 5.3.a] of [CEA-2014-A].

If there is no mapping information between a Remote Control Device and the DAE application, this property returns undefined.

```
function onReceiveRemoteMessage ( Integer type, Integer remoteDeviceHandle,
Integer reqHandle, String requestLine, String headers, String body )
```

The function that is called when the Remote Control Device sends an HTTP request with one of the pre-defined URIs ("`/rcf/request_cui`" or "`/rcf/request_msg`"), or sends an HTTP request to the OITF to launch a DAE application. The DAE application can distinguish between these two cases by the type parameter as follows:

- When the Remote Control Device requests a control UI by using the pre-defined URI "`/rcf/request_cui`", the function is called with the type parameter `REQUEST_CUI`.
- When the Remote Control Device sends a message by using the pre-defined URI "`/rcf/request_msg`", the function is called with the type parameter `REQUEST_MSG`.

When the DAE application is launched by the OITF in response to a request from the control UI in the Remote Control Device, the function is called with the type parameter `CREATE_APP`. The function will be called after the DAE application has loaded (i.e. after the onLoad event has been dispatched to the

DAE application). The DAE application being launched is expected to contain an instance of the application/oipfRemoteControlFunction object.The OITF SHALL dispatch the event to the application/oipfRemoteControlFunction object in the DAE application matched with the Remote Control Device handle which are paired in the mapping information table (see Section 8.4.6). When the ReceiveRemoteMessage event is dispatched to the target application, the DAE application receives the Remote Control Device's User-Agent header value containing the Remote Control Device's capability (in the headers parameter) which the OITF was given with the HTTP request from the Remote Control Device. The DAE application SHALL include the User-Agent value from the Remote Control Device in the XMLHTTPRequest object it uses to retrieve the appropriate Control UI from the IPTV Applications server (see Section 8.1.2).

When this event is invoked, the DAE application SHALL respond by calling the sendRemoteMessage() method. This method need not be called from the event handling function, and may be called after a request to the IPTV Applications Server for an appropriate Control UI has completed.

Only one Remote Control Device is allowed to connect to a DAE application (see Section 8.4.6) at any time. If an HTTP request from another Remote Control Device directed at the DAE application is received by the OITF while a Remote Control Device is connected, the OITF SHALL NOT make and dispatch ReceiveRemoteMessage events to the target DAE application but SHALL send an HTTP response (HTTP 500 - Internal Server Error) to the Remote Control Device.

Every HTTP request from a Remote Control Device to the DAE application with which it is paired SHALL generate an onReceiveRemoteMessage event, even if there are previous HTTP requests which the DAE application has not yet responded to. Each HTTP request SHALL be given a unique reqHandle by the OITF to allow the DAE application to distinguish between outstanding requests.

The specified function is called with six arguments: type, remoteDeviceHandle, reqHandle, requestLine, headers and body which are defined as follows:

- Integer type – the type of the HTTP request from the Remote Control Device. This SHALL take one of the following values:
  - REQUEST_CUI
  - REQUEST_MSG
  - CREATE_APP
- Integer remoteDeviceHandle – the handle of the Remote Control Device which is sending the HTTP request to the DAE application. This handle has a unique value which is assigned by the OITF.
- Integer reqHandle – the handle of the request from the Remote Control Device. The value of this handle is assigned by the OITF, and is unique within the OITF for the duration of a session (the duration of the connection between the OITF and that Remote Control Device). Applications SHALL NOT rely on the value of this handle being preserved across sessions.
- String requestLine – the HTTP requestLine string that comes from the Remote Control Device.
- String headers – the HTTP request header string that comes from the Remote Control Device.
- String body – the HTTP request body that comes from the Remote Control Device.

The values of the requestLine, headers and body parameters are derived from the received HTTP request as follows:

Where: HTTP Request = Request-Line CRLF Header-Lines CRLF Message

Header-Lines = *((general-header | request-header | entity-header) CRLF) Message = [message-body] Then: requestLine = “Request-Line”. headers = ”Header-Lines”. body = “Message “.

`function` **`onResultMuticastNotif`** `( Integer remoteDeviceHandle, Integer reqHandle, Boolean dynamic )`
The function that is called when the Remote Control Device sends an HTTP request with an URL which is a value of a `<ruiEventURL>` element in the Multicast Notification Message. When this event is invoked with true value in the `dynamic` parameter, the DAE application SHALL respond by calling the `sendRemoteMessage` method. This method need not be called from the event handling function, and may be called after a request to the IPTV Applications Server for an appropriate notification CE-HTML document has completed.

7.17.1.3 Methods

`Boolean` **`useServerSideXMLUIListing`**`( String xmlUIListing, Boolean advertiseImmediately )`		
Description	Generate an XML UI Listing by merging the XML UI Listing currently being exposed by the DLNA RUIS in the OITF with the XML UI Listing provided by the `xmlUIListing` parameter of this method. If the OITF successfully generates the new XML UI Listing, this method SHALL return `true`. Otherwise, it SHALL return `false`.	
Arguments	*xmlUIListing*	The Server Side XML UI Listing.
	advertiseImmediately	After generating the new XML UI Listing, if this parameter is true, the DLNA RUIS in the OITF SHALL send a UPnP Discovery (SSDP:byebye) message followed by a UPnP Discovery (SSDP:alive) message. This notifies the DLNA RUIC in any Remote Control Device that it should retrieve the new XML UI Listing.

`Boolean` **`sendRemoteMessage`**`( Integer remoteDeviceHandle, Integer reqHandle, String headers, String message )`	
Description	Send the HTTP response with the headers and the message to the Remote Control Device related to `remoteDeviceHandle`. This method is called by a DAE application in response to a HTTP request from the Remote Control Device. This method can be called at any time for any pending HTTP request (i.e. a request with handle `reqHandle` from the Remote Control Device with handle `remoteDeviceHandle` that has not had a response from the OITF via a `sendRemoteMessage()` or `sendInternalServerError()` call).

	This method SHALL return `true` if the operation succeeded, or `false` if failed. If there is no HTTP connection, it also returns `false`.	
Arguments	*remoteDeviceHandle*	The handle of the Remote Control Device.
	reqHandle	The handle of the request as provided by `onReceiveRemoteMessage`.
	headers	The HTTP response header string. This string is added to the default HTTP header string generated by the OITF to form the HTTP header string used for the HTTP response. Any parameters that are specified in both strings SHALL be set to the value in the headers argument. If the headers supplied by the application do not include a `Content-Type` header, the OITF SHALL use the default content type of `application/ce-html+xml`.
	message	The HTTP response body string whose type is text (e.g. XML, JSON, CE-HTML or Plain Text).

`Boolean` **`sendMulticastNotif`**`(Integer remoteDeviceHandle, Integer eventLevel, String notifCEHTML, String friendlyName, String profilelist )`		
Description	Send the 3rd party multicast notification to any Remote Control Devices (as defined in Section 5.6.1 of [CEA-2014-A]) based on target Remote Device information. The OITF SHALL store the text (essentially a CE-HTML document) provided in the `notifCEHTML` parameter inside the DLNA RUIS and SHALL create a URL to it which can be used by Remote Control Devices to retrieve the original text. This URL SHALL be inserted in the <ruiEventURL> element in the Multicast Notification Message. If the `notifCEHTML` parameter is set to `null`, the HTTP request from the Remote Device to retrieve the text SHALL be being pended and dispatch the `onResultMuticastNotif` event to the DAE application which will retrieve a CE-HTML document dynamically. The DAE application SHALL use the sendRemoteMessage method with a CE-HTML document related parameters to send the text (notification message). If the `remoteDeviceHandle` parameter in this method has a value other than -1, the notification CE-HTML document will be retrieved by the only Remote Device matched with the `remoteDeviceHandle` parameter, whereas if the parameter has -1, all of the Remote Devices could retrieve the notification CE-HTML document from the OITF (see 8.4.5). This method SHALL return true if the operation succeeded, or false if it failed.	
Arguments	*remoteDeviceHandle*	The handle of the Remote Device.
	eventLevel	The value of the HTTP LVL. This allows the Remote Control Devices to filter the multicast notification messages. The following are the defined event levels and the expected meaning of those values (see Section 5.6.1 of [CEA-2014-A] for more information):

Status	Semantics
0	The “upnp:/emergency” is included in the

			LVL header of the multicast notification. The event carries critical information that the Remote Control Device should act upon immediately.
		1	The “upnp:/fault” is included in the LVL header of the multicast notification. The event carries information related to an error case.
		2	The “upnp:/warning” is included in the LVL header of the multicast notification. The event carries information that is a non-critical condition that the Remote Control Device may want to process or pass to the user.
		3	The “upnp:/info” is included in the LVL header of the multicast notification. The event caries informational contents that is not part of the main service interaction but may be useful to some Remote Control Devices in some circumstances, such as debugging information or other data.
		4	The “upnp:/general” is included in the LVL header of the multicast notification. For events that fit into no other defined category.
	notifCEHTML	The text that makes up the notification CE-HTML document, the link to which is sent to the Remote Control Device.	
	profileList	All the profiles that the Remote UI Server in the OITF requires the Remote UI Client in the Remote Control Device to support to properly render the notification CE-HTML document. The value of the `<profilelist>` element SHALL conform to the definition of the `<profilelist>` element in the XML schema in Annex B of [CEA-2014-A].	

`Boolean` **`sendInternalServerError`**`( Integer remoteDeviceHandle, Integer reqHandle )`		
Description	Send the HTTP status code (500: Internal Server Error) in response to a pending HTTP request from the Remote Control Device. This method SHALL return `true` if the operation succeeded, or `false` if it failed.	
Arguments	*remoteDeviceHandle*	The handle of the Remote Control Device.

	reqHandle	The handle of the request as provided by onReceiveRemoteMessage.

`Boolean` **`dropConnection`**`( Integer remoteDeviceHandle )`		
Description	Remove the mapping information in the table between the DAE application and the Remote Control Device currently bound to the DAE application. This method SHALL return `true` if the operation succeeded, or `false` if it failed.	
Arguments	*remoteDeviceHandle*	The handle of the Remote Control Device.

7.17.1.4 Events

For the intrinsic events listed in the table below, a corresponding DOM level 2 event SHALL be generated in the following manner:

Intrinsic event	Corresponding DOM 2 event	DOM 2 Event properties
onReceiveRemoteMessage	ReceiveRemoteMessage	Bubbles: No Cancelable: No Context Info: `type`, `remoteDeviceHandle`, reqHandle, `requestLine`, `headers`, `body`
`onResultMuticastNotif`	`ResultMuticastNotif`	Bubbles: No Cancelable: No Context Info: `remoteDeviceHandle`, `reqHandle`, `dynamic`

NOTE: the above DOM 2 events are directly dispatched to the event target, and will not bubble nor capture. Applications SHOULD NOT rely on receiving `ReceiveRemoteMessage` or a `ResultMuticastNotif` event during the bubbling or the capturing phase. Applications that use DOM 2 event handlers SHALL call the `addEventListener()` method on the application/oipfRemoteControlFunction object. The third parameter of `addEventListener`, i.e. "`useCapture`", will be ignored.

8 System integration aspects

8.1 HTTP Protocol

In addition to what is required by Section 5.3 of [CEA 2014A] an OITF SHALL apply the following requirements.

8.1.1 HTTP User-Agent header

All DAE application HTTP requests SHALL include a `User-Agent` header using the syntax described in this section. Embedded objects HTTP requests MAY include a `User-Agent` header using this syntax.

The `User-Agent` header SHALL include:

```
OIPF-<oipfProfile>/<releaseVersion>.<majorVersion>.<minorVersion> (<capabilities>;
[<vendorName>]; [<modelName>]; [<softwareVersion>];  [<hardwareVersion>]; <reserved>)
[<appName>[/<appVersion>]]
```

Where:

- the `<oipfProfile>` field identifies the profile implemented by the OITF as defined in the specification of the oipfProfile property of the LocalSystem class (see section 7.3.3).
- the `<releaseVersion>`, `<majorVersion>` and `<minorVersion>` fields identify the version of the specification implemented by the OITF as defined in section 7.3.3 with properties of the same name.
- the `<capabilities>` field consists of a description of the OITFs capabilities. Valid values include a base profile string concatenated with one or more optional Profile name fragment strings, such as the base UI profile strings and UI profile name fragment strings as defined in Section 9.2.
- the `<vendorName>`, `<modelName>`, `<softwareVersion>` and `<hardwareVersion>` fields are the same as the one defined in Section 7.11.1 and are optional.
- the `<reserved>` field is reserved for future extensions
- the `<appName>` and `<appVersion>` fields are defined in the `window.navigator` object and are optional.

This `User-Agent` header MAY be extended with other implementation-specific information.

Valid examples of such syntax are:

```
User-Agent: OIPF-OIP/2.2.0 (OITF_HD_UIPROF+PVR+DL; Sonic; TV44; 1.32.455; 2.002;)
Bee/3.5
User-Agent: OIPF-BMP/2.2.0 (OITF_HD_UIPROF+PVR+DL;;;;;)
```

8.1.2 HTTP X-OITF-RCF-User-Agent header

When the DAE application or embedded object ("`application/oipfRemoteControlFunction`") makes a HTTP request for the Control UI to the IPTV Applications server, the value of the `X-OITF-RCF-User-Agent` header SHALL be filled with the value of the `User-Agent` header provided by the DAE application (and which came from the DLNA RUIC on the Remote Control Device).

8.2 Mapping from APIs to Protocols

This section describes mapping of DAE APIs to the specific protocol entities as defined [OIPF_PROT2].

Section 8.2.1 describes mappings on the UNI that apply to both the managed and unmanaged cases.

Section 8.2.2 describes mappings on the HNI-IGI interface, and only apply in the managed case.

Section 8.2.3 describes mappings on the UNI that only apply to the unmanaged case.

8.2.1 Network (Common to Managed and Unmanaged Services)

This section provides details of mapping of the DAE APIs to the descriptions provided in [OIPF_PROT2] for APIs between the OITF and the Network over reference points UNIT-17.

8.2.1.1 Download CoD

Methods	Procedures
registerDownload(String contentAccessDownloadDescriptor, Date downloadStart)	API described in Section 7.4.1.1 to download content described in the contentAccessDownloadDescriptor. Data structure of the contentAccessDownloadDescriptor as described in Annex E.1. If the OITF includes the Content Download functional entity ,the information in the contentAccessDescriptor is passed to the Content Download functional entity to download content over UNIT-17 using HTTP as described in Section 5.2.3.1 of [OIPF_PROT2].

8.2.1.2 Media Queuing

Methods	Procedures
queue(String url)	API described in section 7.14.3 to queue an additional media item for playback when the current media item finishes playback. Queued media items available via HTTP or stored on the terminal MAY be pre-buffered by the OITF in order to reduce transition delays. When pre-buffering media items, the specified buffering policy SHALL NOT be affected. For queued media items available via RTSP, session setup MAY be carried out prior to the end of the currently playing media item.

play(Number speed)	API described in Section 5.7.1 of [CEA-2014-A] to play a media item.
	
	When the start of a media item is reached due to a negative play speed, the playback SHOULD resume at normal play speed without playing any previous media items.
	
	When the end of a media item is reached, playback of any queued media items SHALL be initiated automatically at the specified play speed. The OITF SHALL map this on to the underlying protocol (HTTP or RTSP) as the following sequence of DAE method calls:
	`data = <URI of the queued media item>;`
	`play(<current play speed>);`

seek(Integer pos)	API described in Section 5.7.1 of [CEA-2014-A] to seek to the specified position in a media item. If the value of pos is outside the current media item, the play position SHALL NOT be changed.
next()	Not Supported.
previous()	Not Supported.
read/write String **data**	API described in Section 5.7.1 of [CEA-2014-A] to play a media item. Modification of this property SHALL cause any queued media items to be discarded.

8.2.2 OITF-IG Interface (Managed Services Only)

This section provides details of mapping of the DAE APIs to the descriptions provided in [OIPF_PROT2] for APIs between the OITF and the Network over reference points HNI-IGI. Some methods and properties are closely associated to HNI-IGI and are included in this section. These are the RTSP control, reference point UNIS-11, and IGMP control, reference point UNIS-13,

8.2.2.1 Streaming CoD

The following tables describe the mapping of several methods of the CEA-2014 AV embedded object to the HNI-IGI protocol interfaces defined in [OIPF_PROT2]

Method	Procedures
play(Number speed)	Selection of a content item results in session initiation and access to content stream. Parameters needed to build the offer SDP may be pre defined locally in the OITF or the OITF SHALL request the IG to retrieve missing SDP parameters as described in [OIPF_PROT2] Sec 5.2.2.1 'Protocol over HNI-IGI'. If the OITF does not have all transport parameters (RTP or UDP transport for MPEG2TS encapsulation or direct RTP, FEC layers addresses and ports), code information or bandwidth information to populate the SDP the OITF SHALL prompt the IG to send OPTIONS request in order to retrieve the missing parameters The OITF SHALL provide the following information for the OPTIONS request. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list. X-OITF-Request-Line: Identify the HNI-IGI method with the content identifier as described by the **data** property. e.g. `OPTION sip:PSI-Twister@IPTV_Service_Control.orange.com SIP/2.0`

Method	Procedures
	<table><tr><td>X-OITF-From</td><td>Local defined OITF **CurrentUser** property. e.g. <sip:family@ims.live.ericsson.com>;tag=1211455936632545012</td></tr><tr><td>X-OITF-To</td><td>Copied from the **data** property. e.g. sip: PSI-Twister@IPTV_Service_Control.orange.com</td></tr></table> The response to the OPTIONS message request contains the information to populate the SDP offer. The OITF prepares an SDP offer and requests the IG to initiate a session, in addition to the SDP the following parameters are forwarded from the OITF to the IG. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list. <table><tr><td>X-OITF-Request-Line</td><td>Identify the HNI-IGI method with the content identifier as described by the **data** property. e.g. INVITE sip:PSI-Twister@IPTV_Service_Control.orange.com SIP/2.0</td></tr><tr><td>X-OITF-From</td><td>Local defined OITF **CurrentUser** property. e.g. <sip:family@ims.live.ericsson.com>;tag=1211455936632545012</td></tr><tr><td>X-OITF-To</td><td>Copied from the **data** property. e.g. sip: PSI-Twister@IPTV_Service_Control.orange.com</td></tr></table> After a successful session setup the OITF SHALL use the media player to access the RTSP URI with the session ID negotiated and received as part of the SDP offer, described in [OIPF_PROT2] sec 7.1.1.2 ‘RTSP for managed model UNIS-11 and NPI 10’. The OITF SHALL send an RTSP PLAY over UNIS-11 using attribute values received in the SDP from the session initiation procedure. The RTSP PLAY is as described in the [OIPF_PROT2] Sec 7.1.1.2 ‘RTSP for managed model UNIS-11 and NPI 10’. The RTSP fields in the RTSP PLAY message SHALL be filled as follows: • The RTSP URL SHALL be set from the SDP h-uri attribute in the case of an absolute URI. The “data” property SHALL be updated with the SDP h-uri attribute. If the value of h-uri is a relative URI that is in the form of a media path, then the RTSP absolute URL is constructed by the OITF using the SDP IPAddress (from c-line) and port (from m-line) as the base followed by h-uri value for the media path. (e.g. rtsp://10.5.1.72:22554/TV3/823527) • The RTSP Scale header SHALL be set to the value specified in argument speed in method play. The argument SHOULD

<table>
<tr><th>Method</th><th>Procedures</th></tr>
<tr><td></td><td>equal one of the values in the playsSpeeds property. The Scale values [RTSP sec 12.34] are as follows:

▪ 1 indicates normal play.

▪ If not 1, the value corresponds to the rate with respect to normal viewing rate.

▪ A negative value indicates reverse direction.

If the speed argument of method play does not equal a supported play speed indicated by the playSpeeds property, the player SHALL play the content at the closest available playback speed. The <code>play()</code> method SHOULD only return false if the best effort to play back the file at any speed has failed.

The actual playback speed SHALL be available through the “speed” property of the A/V Control object.

After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a <code>PlaySpeedChanged</code> event indicating the actual playback speed.</td></tr>
<tr><td>stop()</td><td>The method enables the OITF to terminate and ongoing CoD session. The OITF SHALL request the IG to terminate the session as described in [OIPF_PROT2] Sec 5.2.2.1 ’Protocol over HNI-IGI’.

The OITF SHALL include the following information from the request. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.

<table><tr><td><code>X-OITF-Request-Line</code></td><td>Identify the HNI-IGI method with the content identifier as described by the data property. e.g.

<code>BYE sip:PSI-Twister@IPTV_Service_Control.orange.com SIP/2.0</code></td></tr><tr><td><code>X-OITF-From</code></td><td>Local defined OITF CurrentUser property. eg.

<code><sip:family@ims.live.ericsson.com>;tag=1211455936632545012</code></td></tr><tr><td><code>X-OITF-To</code></td><td>Copied from the data property. eg.

<code>sip: PSI-Twister@IPTV_Service_Control.orange.com</code></td></tr></table>
The OITF SHALL remove all context information relevant to the terminated COD session upon a successful response from the IG.</td></tr>
<tr><td>seek(Integer pos)</td><td>If the <code>seek()</code> method is called while the player is in the “playing” state, it sets current play position to “<code>pos</code>”, by using the “Range” parameter in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.2 ‘RTSP for managed model UNIS-11 and NPI 10’.

After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a <code>PlayPositionChanged</code></td></tr>
</table>

Method	Procedures
	event indicating a new playback position of "pos". If the seek() method is called while the player is in the "paused" state, the value of playPosition is changed to reflect the new play position. This is the new play position that SHALL be used for the "Range" parameter of the RTSP PLAY message when playback is resumed.
play(0)	This method causes the OITF to send an RTSP PAUSE message (refer to [OIPF_PROT2] sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI-10'). The RTSP PAUSE message SHALL include: • The RTSP URL SHALL be set to the value retrieved from the fmtp:iptv_rtsp h-uri attribute of the SDP answer. • Session header SHALL be set as specified in the SDP answer fmtp:iptv_rtsp h-session attribute After a successful response to the RTSP PAUSE message has been received, the OITF SHALL generate a PlaySpeedChanged event indicating a playback speed of 0.
next()	Not Supported. Note: Track information is not supported in [OIPF_PROT2] and is therefore out of scope.
previous()	Not Supported. Note: Track information is not supported in [OIPF_PROT2] and is therefore out of scope.

Property	Procedures
read/write String **data**	This property holds the URL that identifies the content, as defined in [OIPF_PROT2] Sec 6.2.2.1.1 'Protocol over UNIS-8' for details on CoD URI. It is used by the OITF compose the following headers for requests towards the IG X-OITF-Request-Line X-OITF-To If the "data" property of the A/V Control object refers to a Content-Access Streaming Descriptor (i.e. the object has type "application/vnd.oipf.ContentAccessStreaming+xml" as defined in Section 7.14.2), the OITF must perform the following steps prior to performing the procedures defined in [OIPF_PROT2] as described for method play(): • An HTTP GET request SHALL be made with the Request-URI set to the URL of the Content-Access Descriptor as denoted by the "data" property of the A/V Control object. • After the server has returned a Content Access Streaming Descriptor (i.e. a document with type "application/vnd.oipf.ContentAccessStreaming+xml"),

Property	Procedures
	the OITF SHALL interpret the contents of the Content-Access Descriptor and choose a URL defined by one of the <ContentURL> elements. The criteria for choosing a URL can be the DRM system supported by the OITF. The URL SHALL then be used for setting up a Streaming CoD session, after which playback can be started (when the `play()` method is invoked). The "`data`" property of the A/V Control object SHALL be changed to represent the chosen URL. • Based on the information retrieved from the Content-Access Streaming Descriptor, the OITF SHALL passing the <DRMControlInformation> to the appropriate DRM agent, and SHOULD initialize the AV playback, i.e. by loading the correct codecs as identified by the Content-access Streaming Descriptor.
`readonly Number` **`playPosition`**	The property holds the current play position in milliseconds of the media referenced by the data property. The property value SHALL be based on the value retrieved using the RTSP GET_PARAMETER method and parameter "`position`" (refer to Section 7.1.1.2 of [OIPF_PROT2]) adjusted for played duration and used scale. If information is not available the value SHALL be `undefined`. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.
`readonly Number` **`playSpeeds`**`[ ]`	The property holds the available speeds (known in RTSP as scales) at which the media can be played back. The property value SHALL be based on the value retrieved using RTSP GET_PARAMETERS method and parameter "`scales`" (refer to Section 7.1.1.2 of [OIPF_PROT2]). If information is not available the value SHALL be `undefined`. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.
`readonly Number` **`playTime`**	The property holds the total duration in milliseconds of the media referenced by the data property. The property value SHALL be based on the value retrieved using RTSP GET_PARAMETER method and parameter "duration" (refer to Section 7.1.1.2 of [OIPF_PROT2]). If information is not available the value SHALL be `undefined`. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.
`readonly Number` **`playState`**	No procedures defined since it is not related to protocol specification.
`readonly Number` **`error`**	No procedures defined since it is not related to protocol specification.
`readonly Number` **`speed`**	Float value indicating the actual playback speed for the content referenced by the data property. The normal default playback speed is represented by value 1.

Intrinsic event	Procedure
onPlaySpeedChanged	When RTSP ANNOUNCE with either beginning-of-stream or end-of-stream codes arrives the OITF SHALL generate onPlaySpeedChanged event with a speed value of 0.
onPlayPositionChanged	When the response to the RTSP PLAY with Range header request (Range is included when performing seek() with a position) the OITF SHALL generate onPlayPositionChanged event with the accepted position.

8.2.2.2 Scheduled Content

8.2.2.2.1 Conveyance of channel list

Service discovery description procedure as described in [OIPF_PROT2] sec 6.3.1.1 'Service Provider discovery' and [OIPF_PROT2] Annex B 2.3 'IPTV Service discovery description' enables the OITF to obtain the URL to access the broadcast channel information. The OITF SHALL utilise UNIS-7 using this URL to obtain the Broadcast Discovery Record.

8.2.2.2.2 Switching channels

Methods	Procedures
setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)	The setChannel() method of the <video/broadcast> object SHALL be used to initiate a broadcast session or switch channels. The procedures that are performed over the HNI-IGI reference point depend on the current state of broadcast session, either it is active or not. Note that an inactive broadcast session means no service is being viewed. If the channel has an idType of ID_IPTV_URI, the OITF SHALL send and IGMP Leave and an IGMP Join request on the UNIS-13 as described in [OIPF_PROT2] Sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13'. If the channel has an idType of ID_IPTV_SDS, the following steps are taken: **Session Initiation** The OITF SHALL generate a session initiation request over the HNI-IGI including and SDP offer as described in [OIPF_PROT2] sec 5.2.1 'Scheduled Content'. The bandwidth is set according to the explanation under heading "Selection of Bandwidth" further down. If a "contentAccessDescriptorURL" has been specified for the setChannel() method, the OITF must perform the following steps prior to performing the procedures defined in [OIPF_PROT2] for performing setChannel() as described below: • An HTTP GET request SHALL be made with the

Request-URI set to the URL of the Content-Access Descriptor as denoted by the “contentAccessDescriptor” attribute.

- Based on the information retrieved from the Content-Access Descriptor, the OITF SHALL passing the <DRMControlInformation> to the appropriate DRM agent.

The OITF SHALL provide the following information as part of the scheduled session initiation request as described in [OIPF_PROT2] Sec 6.2.1 ‘Scheduled Content’. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.

X-OITF-Request-Line	Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content. eg. `INVITE sip:IPTV_SC_Service@iptv.ericsson.com SIP/2.0`
X-OITF-From	Local defined OITF **CurrentUser** property. eg. `<sip:family@ims.live.ericsson.com>; tag=1211455936632545012`
X-OITF-To	PSI of the scheduled content. eg. `sip:IPTV_SC_Service@iptv.ericsson.com`

The Offer SDP included in the OITF be SHALL have attributes as described in [OIPF_PROT2] Annex E.2 ‘Service Package SDP attributes.

On positive response to the INVITE request the OITF SHALL send an IGMP Join request on the UNIS-13 as described in [OIPF_PROT2] Sec 8.1.1.1 ‘Procedure for Scheduled Content on UNIS-13’.

Session Modification

If the bandwidth conditions change as described under heading “Selection of Bandwidth” further down then the OITF SHALL generates a session modification request over the HNI-IGI including the new SDP offer.

The OITF SHALL provide the following information as part of the scheduled session modification request as described in [OIPF_PROT2] Sec 6.2.1 ‘Scheduled Content’. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.

X-OITF-Request-Line	Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content. eg. `INVITE sip:IptvBroadcast@iptv.ericsson.com SIP/2.0`
X-OITF-From	Local defined OITF **CurrentUser** property. eg.

	<sip:family@ims.live.ericsson.com>; tag=1211455936632545012
X-OITF-To	PSI of the scheduled content. eg. sip:IptvBroadcast@iptv.ericsson.com

The Offer SDP included by the OITF SHALL have attributes as relevant to the new channel as described in [OIPF_PROT2] Annex E.2 'Service Package SDP attributes'.

On receiving a successful response to the INVITE request the OITF SHALL send and IGMP Leave and IGMP Join request on the UNIS-13 as described in [OIPF_PROT2] Sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13'.

No Session Modification

If the bandwidth conditions as described under heading "Selection of Bandwidth" further down have not changed then the OITF SHALL send a membership report to leave the previously viewed channel, if applicable, and with the same membership report join to the multicast group associated with the selected channel. The multicast group information is retrieved from the Broadcast Discovery Record.

Selection of Bandwidth

The bandwidth to be used for the broadcast session depends on the information provided in the Broadcast Discovery Record (refer to [OIPF_META2]). The Broadcast Discovery Record uses the term "service" to indicate a channel.

If the TimeToRenegotiate (TTR) element is not provided within the IPService of the Broadcast Discovery Record then the bandwidth SHALL be based on the maximum bandwidth for all the services in the Broadcast Discovery Record. In this case only one session initiation is performed at initial activation of broadcast service, and no session modification is required.

If the TTR element is provided then the MaxBitRate from the new service and current service are compared. If broadcast service is not active and there is no active current service, session initiation is performed with the new service MaxBitRate. For already active broadcast service there are three conditions.

- If the MaxBitrate of the new service is greater than that of the current service and the reserved bandwidth is exceeded, network bandwidth reservation using the MaxBitrate of the new service SHALL occur immediately with session modification to ensure sufficient bandwidth is made available for the new service.

- If the MaxBitrate of the new service is equal to that of the current service, network bandwidth reservation procedures SHALL NOT be performed as sufficient bandwidth is already available for the new service.

- If the MaxBitrate of the new service is less than that of

	the current service and there is no pending TTR timer, a timer using the TTR element of the new service is started which will renegotiate the bandwidth with session modification. Note that at every channel change if there is a pending timeout for session modification due to a previous service change then the timer is restarted. When the timer expires the bandwidth for the currently viewed service is used in a session modification. The session initiation, session modification and no session modification are further described above.

8.2.2.2.3 End broadcast service

Methods	Procedures
`release`()	The release method of the video/broadcast object causes the OITF to perform an IGMP Leave on the active broadcast session as described in [OIPF_PROT2] sec 8.1.1.1 “Procedure for leaving a Scheduled Content service”. If the channel has an `idType` of `ID_IPTV_SDS`, the OITF SHALL then execute a session termination procedure by sending a BYE request over the HNI-IGI interface as described in section [OIPF_PROT2] Sec 5.2.1.1 ‘Protocol over HNI-IGI’. The request SHALL include the following information. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.

`X-OITF-Request-Line`	Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content. E.g. `INVITE sip:IPTV_SC_Service@iptv.ericsson.com SIP/2.0`
`X-OITF-From`	Local defined OITF **CurrentUser** property. eg. `<sip:family@ims.live.ericsson.com>; tag=1211455936632545012`
`X-OITF-To`	PSI of the scheduled content. e.g.: `sip:IPTV_SC_Service@iptv.ericsson.com`

8.2.2.2.4 Network timeshift of broadcast service

Methods	Procedures
`pause`()	The method has different behaviour if the `pause()` method has previously been invoked. While the first `pause()` request sets up the session over HNI-IGI the subsequent `pause()` requests simply issue an RTSP PAUSE request. **First `pause()` request**

	The OITF SHALL generates a session modification request over the HNI-IGI including the modified SDP offer. The SDP offer included by the OITF SHALL have attributes as relevant to the unicast stream to be setup.
	
	The OITF SHALL provide the following information as part of the scheduled session modification request as described in [OIPF_PROT2] sec 6.2.1 'Scheduled Content'. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.
	
	`X-OITF-Request-Line`: Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content, e.g.
	`INVITE sip:IptvBroadcast@iptv.ericsson.com SIP/2.0`
	
	`X-OITF-From`: Local defined OITF CurrentUser property, e.g.
	`<sip:family@ims.live.ericsson.com>; tag=1211455936632545012`
	
	`X-OITF-To`: PSI of the scheduled content, e.g. `sip:IptvBroadcast@iptv.ericsson.com`
	
	On receiving a successful response to the INVITE request and if the channel has an `idType` of `ID_IPTV_URI`, the OITF SHALL send and IGMP Leave and request on the UNIS-13 as described in [OIPF_PROT2] sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13'.
	
	Subsequent `pause()` requests
	
	This request causes the OITF to send an RTSP PAUSE message (refer to [OIPF_PROT2] sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI-10'). The RTSP PAUSE message SHALL include:
	
	• The RTSP URL SHALL be set to the value retrieved from the fmtp:iptv_rtsp h-uri attribute of the SDP answer.
	
	• Session header SHALL be set as specified in the SDP answer fmtp:iptv_rtsp h-session attribute
	
	After a successful response to the RTSP PAUSE message has been received, the OITF SHALL generate a `PlaySpeedChanged` event indicating a playback speed of 0.

`resume()`	The OITF SHALL send an RTSP PLAY over UNIS-11 using attribute values received in the SDP from the session modification procedure. The RTSP PLAY is as described in [OIPF_PROT2] sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI 10'. The RTSP fields in the RTSP PLAY message SHALL be filled as follows: • The RTSP URL SHALL be set from the SDP h-uri attribute in the case of an absolute URI. The "data" property SHALL be

<table>
<tr><td></td><td>updated with the SDP h-uri attribute. If the value of h-uri is a relative URI that is in the form of a media path, then the RTSP absolute URL is constructed by the OITF using the SDP IPAddress (from c-line) and port (from m-line) as the base followed by h-uri value for the media path. (e.g. rtsp://10.5.1.72:22554/TV3/823527)

• The RTSP URL SHALL be set from the SDP h-uri attribute in the case of an absolute URI. The “data” property SHALL be updated with the SDP h-uri attribute. If the value of h-uri is a relative URI that is in the form of a media path, then the RTSP absolute URL is constructed by the OITF using the SDP IPAddress (from c-line) and port (from m-line) as the base followed by h-uri value for the media path.(e.g. rtsp://10.5.1.72:22554/TV3/823527)

After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a PlaySpeedChanged event indicating the actual playback speed.</td></tr>
<tr><td>setSpeed(Number speed)</td><td>Sets current speed by using the “Scale” header in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.1 ‘RTSP for managed model UNIS-11 and NPI 10’.

After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a PlaySpeedChanged event indicating a new playback speed.</td></tr>
<tr><td>seek(Integer offset, Integer reference)</td><td>Sets current play position to “pos”, by using the “Range” parameter in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.2 ‘RTSP for managed model UNIS-11 and NPI 10’.

After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a PlayPositionChanged event indicating a new playback position of “pos”.</td></tr>
<tr><td>stopTimeShift()</td><td>The OITF SHALL generates a session modification request over the HNI-IGI including the modified SDP offer. The SDP offer included by the OITF SHALL have attributes as relevant to the channel as described in [OIPF_PROT2] Annex E.2 ‘Service Package SDP attributes’.

The OITF SHALL provide the following information as part of the scheduled session modification request as described in [OIPF_PROT2] sec 6.2.1 ‘Scheduled Content’. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list.

<table>
<tr><td>X-OITF-Request-Line</td><td>Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content, e.g.

INVITE sip:IptvBroadcast@iptv.ericsson.com SIP/2.0</td></tr>
<tr><td>X-OITF-From</td><td>Local defined OITF CurrentUser property, e.g. <sip:family@ims.live.ericsson.com>; tag=1211455936632545012</td></tr>
</table>
</td></tr>
</table>

	<table><tr><td>X-OITF-To</td><td>PSI of the scheduled content, e.g. sip:IptvBroadcast@iptv.ericsson.com</td></tr></table> On receiving a successful response to the INVITE request and if the channel has an idType of ID_IPTV_URI, the OITF SHALL send and IGMP Join and request on the UNIS-13 as described in [OIPF_PROT2] sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13'.
setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)	The following procedure is only applicable if Network Timeshift of broadcast service is in progress. The OITF SHALL generates a session modification request over the HNI-IGI including the modified SDP offer. The SDP offer included by the OITF SHALL have attributes as relevant to the new channel as described in [OIPF_PROT2] Annex E.2 'Service Package SDP attributes'. The OITF SHALL provide the following information as part of the scheduled session modification request as described in [OIPF_PROT2] sec 6.2.1 'Scheduled Content'. Not all required headers are listed. Refer to [OIPF_PROT2] for a complete list. <table><tr><td>X-OITF-Request-Line</td><td>Identify the HNI-IGI method with the well known PSI (Public Service Identifier) of the scheduled content, e.g. INVITE sip:IptvBroadcast@iptv.ericsson.com SIP/2.0</td></tr><tr><td>X-OITF-From</td><td>Local defined OITF CurrentUser property, e.g. <sip:family@ims.live.ericsson.com>; tag=1211455936632545012</td></tr><tr><td>X-OITF-To</td><td>PSI of the scheduled content, e.g. sip:IptvBroadcast@iptv.ericsson.com</td></tr></table> On receiving a successful response to the INVITE request and if the channel has an idType of ID_IPTV_URI, the OITF SHALL send and IGMP Join and request on the UNIS-13 as described in [OIPF_PROT2] sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13'.

Property	Procedures
read/write String **data**	This property holds the RTSP URI from the SDP h-uri attribute. Prior to a successful SIP INVITE the value is undefined.

Note that all the remaining properties listed under section 8.2.2.1, Streaming CoD, SHALL be supported as described.

8.2.2.3 IMS APIs

Methods	Procedures
registerUser(String userId, String pin)	Performs IMS registration with the specified user ID as described in [OIPF_PROT2] sec 5.3.6.1 'Procedure for User Registration and Authentication in Managed Model on HNI-IG Interface'.
deRegisterUser(String userId)	Performs IMS de-registration with the specified user ID as described in [OIPF_PROT2] sec 5.3.6.1 'Procedure for User Registration and Authentication in Managed Model on HNI-IG Interface'.
subscribeIMSNotification (FeatureTagCollection featureTagCollection, Boolean performuserregistration)	OITF maintains applications that have subscribed to notifications. If applicable it will send a re-registration to the IG. When new messages arrive at the IG it shall notify the OITF. (as defined in [OIPF_PROT2] sec 5.5.1.2).
unsubscribeIMSNotification()	This is a local call within OITF to notify that the DAE application SHALL NOT receive unsolicited notification. The OITF shall use native code to handle new dialogues. Any feature tag values that were added by the DAE application are removed for the indicated userId since no native code is setup to process the new dialogues for the feature tag values.

8.2.3 Network (Unmanaged Services only)

This section provides details of mapping of the DAE APIs to the descriptions provided in [OIPF_PROT2] for APIs between the OITF and the Network. These are the RTSP control, reference point UNIS-11, reference point UNIS-13.

8.2.3.1 Streaming CoD

Method	Procedures
play(Number speed)	The "speed" parameter is a floating point value indicating the requested playback speed. A value of 1 represents normal playback speed, and other values are relative to this. A "speed" value of zero SHALL NOT initiate any procedures. **RTSP** The RTSP URL signalled by the "data" attribute SHALL be used to initiate the process defined in [OIPF_PROT2] Sec 7.1.1.1.1. The "data" attribute SHALL furthermore be updated with the new URI after redirection requests (moved). The RTSP PLAY request SHALL include a "Scale" header set to the value of the "speed" parameter passed to the API. The server will play the stream at the specified speed, if supported. If property oitfNoRTSPSessionControl is set to true then the RTSP messages DESCRIBE and SETUP are not used. If the play() method is called with a non-zero speed the property oipfRTSPSessionId is copied to the RTSP SessionId header for the RTSP PLAY request. If the oipfRTSPSessionId is undefined the play() method SHALL fail. After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a PlaySpeedChanged event indicating the actual playback speed. **HTTP** The HTTP URL signalling by the "data" attribute SHALL be used to initiate the process defined in [OIPF_PROT2] Sec 5.2.2.2. The "data" attribute SHALL furthermore be updated with the new URI after redirection requests (moved). The "speed" parameter SHALL be passed to the OITF media player, which SHOULD attempt to play back the content at the requested speed. If the media player successfully begins to play back the content, the OITF SHALL generate a PlaySpeedChanged event indicating the actual playback speed.
stop()	**RTSP** The OITF SHALL initiate the process defined in [OIPF_PROT2] Sec 7.1.1.1.2 except if the property oitfNoRTSPSessionControl is set to true. **HTTP** The OITF SHALL stop playback. The OITF MAY close the connection to the server and MAY clear any buffered content.

Method	Procedures
seek(Integer pos)	**RTSP** If the seek() method is called while the player is in the "playing state", it sets current play position to "pos", by using the "Range" parameter in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.1 'RTSP for managed model UNIS-11 and NPI 10'. After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a PlayPositionChanged event indicating a new playback position of "pos". If the seek() method is called while the player is in the "paused" state, the value of playPosition is changed to reflect the new play position. This is the new play position that SHALL be used for the "Range" parameter of the RTSP PLAY message when playback is resumed. **HTTP** If the seek() method is called while the player is in the "playing state", the OITF SHALL attempt to playback from the specified position "pos". It MAY use the RANGE header as described in [OIPF_PROT2] Sec 5.2.2.2 as necessary. If the media player successfully begins to play back the content from the specified position, the OITF SHALL generate a PlayPositionChanged event indicating a new playback position of "pos". If the seek() method is called while the player is in the "paused" state, the value of playPosition is changed to reflect the new play position. This is the new play position from which playback SHALL be resumed.
play(0)	**RTSP** This method causes the OITF to send an RTSP PAUSE message (refer to [OIPF_PROT2] sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI-10'). The RTSP PAUSE message SHALL include: After a successful response to the RTSP PAUSE message has been received, the OITF SHALL generate a PlaySpeedChanged event indicating a play speed of 0. **HTTP** The OITF SHALL pause playback. If the media player successfully pauses playback, the OITF SHALL generate a play speed event indicating a PlaySpeedChanged of 0.
next()	Not Supported. Note: Track information is not supported in [OIPF_PROT2] and is therefore out of scope.
previous()	Not Supported. Note: Track information is not supported in [OIPF_PROT2] and is therefore out of scope.

Property	Procedures
read/write String **data**	**RTSP** This property holds the RTSP URI for the content item. **HTTP** The property holds the HTTP URI for the content item. If the "data" property of the A/V Control object refers to a Content-Access Streaming Descriptor (i.e. the object has type "application/vnd.oipf.ContentAccessStreaming+xml" as defined in Section 7.14.2), the OITF must perform the following steps prior to performing the procedures defined in [OIPF_PROT2] as described for method play(): • An HTTP GET request SHALL be made with the Request-URI set to the URL of the Content-Access Streaming Descriptor as denoted by the "data" property of the A/V Control object. • After the server has returned a Content Access Streaming Descriptor (i.e. a document with type "application/vnd.oipf.ContentAccessStreaming+xml"), the OITF SHALL interpret the contents of the Content-Access Streaming Descriptor and choose a URL defined by one of the <ContentURL> elements. The criteria for choosing a URL can be the DRM system supported by the OITF. The URL SHALL then be used for setting up a Streaming CoD session, after which playback can be started (when the play() method is invoked). The "data" property of the A/V Control object SHALL be changed to represent the chosen URL. • Based on the information retrieved from the Content-Access Streaming Descriptor, the OITF SHALL passing the <DRMControlInformation> to the appropriate DRM agent, and SHOULD initialize the AV playback, i.e. by loading the correct codecs as identified by the Content-access Streaming Descriptor.
readonly Number **playPosition**	The property holds the current play position in milliseconds of the media referenced by the data property. For RTP, The property value SHALL be based on the value retrieved using the RTSP GET PARAMETERS method and parameter "position" (refer to [OIPF_PROT2] Sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI-10') adjusted for played duration and used scale. If information is not available the value SHALL be undefined. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.

readonly Number **playSpeeds**[ ]	For RTSP, the property holds the available speeds, or referred in RTSP as Scale, to be used to change the playback speed. The property value SHALL be based on the value retrieved using RTSP GET PARAMETERS method and parameter "scales" (refer to [OIPF_PROT2] Sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI-10'). For HTTP, the possible playback speeds are determined by the OITF internal capabilities and buffering model, and the speed at which content is delivered. The OITF MAY make this information available via this property. If information is not available the value SHALL be undefined. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.
readonly Number **playTime**	The property holds the total duration in milliseconds of the media referenced by the data property. For RTSP, the property value SHALL be based on the value retrieved using RTSP GET_PARAMETER method and parameter "duration" (refer to [OIPF_PROT2] Sec 7.1.1.2 'RTSP for managed model UNIS-11 and NPI10'). For HTTP, the property value MAY be determined using the "Content-Length" HTTP header, although it is noted that this method does not work for variable bit rate content. If information is not available the value SHALL be undefined. Note this may happen at the beginning of playing a video and GET_PARAMETER has not returned a value.
readonly Number **playState**	No procedures defined since it is not related to protocol specification.
readonly Number **error**	No procedures defined since it is not related to protocol specification.
readonly Number **speed**	Float value indicating the actual playback speed of the player for the content referenced by the data property. The normal default playback speed is represented by value 1.

Intrinsic event	Procedure
onPlaySpeedChanged	When RTSP ANNOUNCE with either beginning-of-stream or end-of-stream codes arrives the OITF SHALL generate onPlaySpeedChanged event with a speed value of 0.
onPlayPositionChanged	When the response to the RTSP PLAY with Range header request (Range is included when performing seek() with a position) the OITF SHALL generate onPlayPositionChanged event with the accepted position.

8.2.3.2 Scheduled content

8.2.3.2.1 Switching channels

Methods	Procedures
setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)	The setChannel method of the <video/broadcast> object SHALL be used to initiate a broadcast session or switch channels. If the channel has an idType of ID_IPTV_URI, the OITF SHALL send and IGMP Leave and an IGMP Join request on the UNIS-13 as described in [OIPF_PROT2] Sec 8.1.1.1 'Procedure for Scheduled Content on UNIS-13 with Session Initiation'.

8.2.3.2.2 End broadcast service

Methods	Procedures
release()	The release method of the video/broadcast object causes the OITF to perform an IGMP Leave on the active broadcast session as described in [OIPF_PROT2] Sec. 8.1.1.1 'Procedure for Scheduled Content on UNIS-13 with Session Initiation'.

8.2.3.2.3 Network timeshift of broadcast services

Methods	Procedures
pause()	The pause method of the video/broadcast object causes the OITF to perform an IGMP Leave on the active broadcast session as described in [OIPF_PROT2] sec. 8.1.1.1 "Procedure for leaving a Scheduled Content service".
resume()	The RTSP URL signalled by the "data" attribute SHALL be used to initiate the process defined in [OIPF_PROT2] sec 7.1.1.1.1. The "data" attribute SHALL furthermore be updated with the new URI after redirection requests (moved). The value of the "scale" header in the RTSP PLAY message SHALL be the value set by the most recent call to setSpeed(), or 1.0 if the most recent call to setSpeed() set the playback speed to 0 or setSpeed() has not been called. If property oitfNoRTSPSessionControl is set to true then the RTSP messages DESCRIBE and SETUP are not used. If the play() method is called with a non-zero speed the property oipfRTSPSessionId is copied to the RTSP SessionId header for the RTSP PLAY request. If the oipfRTSPSessionId is undefined the play() method SHALL fail.

	After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a `PlaySpeedChanged` event indicating the actual playback speed.
setSpeed(Number speed)	Sets current speed by using the “Scale” header in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.1 ‘RTSP for managed model UNIS-11 and NPI 10’. After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a `PlaySpeedChanged` event indicating a new playback speed. If playback is previously paused (either by a call to `pause()` or by setting the playback speed to 0) then the new speed SHALL NOT be applied until the `resume()` method is called, as described above.
seek(Integer offset, Integer reference)	Sets current play position to “pos”, by using the “Range” parameter in the RTSP PLAY as described in [OIPF_PROT2] sec 7.1.1.1 ‘RTSP for managed model UNIS-11 and NPI 10’. After a successful response to the RTSP PLAY message has been received, the OITF SHALL generate a `PlayPositionChanged` event indicating a new playback position of “pos”.
stopTimeShift()	The `setChannel()` method of the video/broadcast object SHALL be used to initiate a broadcast session. If the channel has an `idType` of `ID_IPTV_URI`, the OITF SHALL send and IGMP Join request on the UNIS-13 as described in [OIPF_PROT2] sec 8.1.1.1 ‘Procedure for Scheduled Content on UNIS-13’.

Property	Procedures
read/write String **data**	This property holds the RTSP URI for the content item.

Note that all the remaining properties listed under section 8.2.3.1, Streaming CoD, SHALL be supported as described.

8.3 URI Schemes and their usage

The following table lists possible URL schemas and their usages within DAE documents (XHTML, ECMAScript, images, and references to A/V content). If a certain URL scheme is supported, the corresponding protocols to an URL scheme SHALL be supported as defined by the reference(s)

Table 13: URI schemes and usages

URI scheme	Usage	Reference	Comments
dvb-mcast	Scheduled content delivery	DVB-MCAST URI scheme as defined by Annex A1 of [TS 102 539]	A URL to refer to a scheduled content channel supported by the OITF and delivered via multicast.
dvb	Application launching	Locator for applications signalled in SD&S as defined by Section 6.3.3 of [TS 102 851]	The orgid and appid encoded in the DVB URI are compared with the applications signalled in SD&S to identify one with the same orgid and appid.
igmp	Scheduled content	Annex F of [OIPF_PROT2]	The transport IP Multicast Address to access the service as defined in [DVB-IPTV].
http and https	Transport of DAE documents	Section 5.3.3.1 of [OIPF_PROT2] Section 5.3 of [CEA-2014-A] Section 5 of [OIPF_CSP2]	A URL to refer documents supported by DAE.
	COD streaming	Annex F of [OIPF_PROT2]	A Content URL specified in the data attribute of A/V Control object as defined in the section 5.7.1 of [CEA-2014-A]. A Content URL specified in a Content Access Descriptor described in Annex E.
crid	COD streaming	Section 4.2.3 of [OIPF_META2]	
sip	Programme identification via BCG	Annex F of [OIPF_PROT2]	
	COD streaming		
rtsp	COD streaming		

8.4 DLNA RUI Remote Control Function implementation

This section aims to give guidelines to the DAE application developer suggesting how the DAE application should be implemented to use a DLNA Remote UI Function, covering the following areas:

- Relationship between DAE application and control UI
- XML UI Listing Provisioning
- Retrieving the Control UI
- Receiving a message (control command) from the Remote Control Device and Responding to a message from the Remote Control Device
- Notification to the Remote Control Device
- Multiple application handling

The sections below provide more details including example code in each case.

8.4.1 Relationship between DAE application and control UI

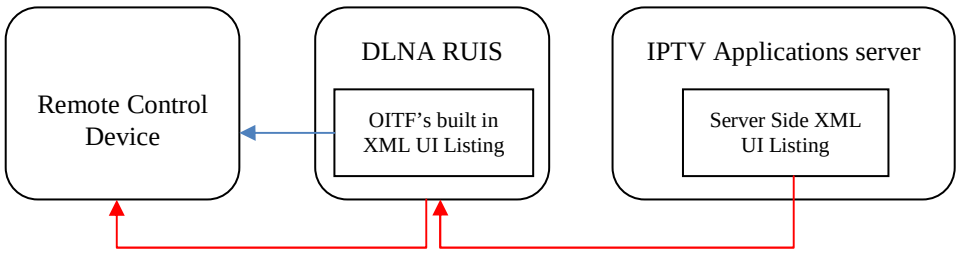
It is assumed that the service provider authors both the DAE application and the control UI to run on the Remote Control Device that communicates with the DAE application. It means that the DAE application and the control UI are managed by one service provider, and the DAE application could handle the HTTP request message which comes from the control UI currently being rendered in the DLNA RUIC.

8.4.2 XML UI Listing Provisioning

There are two kinds of XML UI Listing (details are described in Section 5.1.1.5 of [CEA-2014-A]):

- The OITF's built in XML UI Listing, that originates from the OITF (DLNA RUIS) and which is usually pre-defined by the device vendor,
- The Server Side XML UI Listing, that is provided by the DAE application and which is defined by the service provider.

Below is a description of where each type of XML UI Listing come from.



- OITF's built in XML UI Listing (blue arrow in above diagram):
  - This XML UI Listing contains a set of URI pre-defined by the OITF corresponding to a number of Control UIs that are available in the OITF device itself.
  - The OITF SHALL use this XML UI Listing until a DAE application calls the `useServerSideXMLUIListing()` method.
- Server Side XML UI Listing (red arrows in above diagram):
  - This XML UI Listing contains both the URIs which identify the control UIs located on the appropriate IPTV Applications server through the pre-defined URI "`/rcf/request_cui`".
    - Examples: `/rcf/request_cui?url=www.cui-server.com/avcontrol.html&param1=value1…`

    The XML UI Listing is retrieved (or created dynamically) by a DAE application, which then merges the new XML UI Listing with a current XML UI Listing in the DLNA RUIS using the `useServerSideXMLUIListing()` method. The merged XML UI Listing will be located in the DLNA RUIS.

The OITF SHALL associate all entries in the XML UI Listing added by a DAE application with that application, such that any HTTP requests from a Remote Control Device for the control UI specified by the XML UI Listing entry SHALL be passed to the corresponding application.

All URIs provided in the XML UI Listing SHALL start with the pre-defined URI "/rcf/request_cui", which can then be followed by some application-specific parameters. These parameters can be used by the DAE application to identify the Control UI being requested by the Remote Control Device.

The format of the parameters in the URI is out of scope of the DAE specification.

- When the DAE application is terminated, the OITF SHALL remove any XML UI Listings previously added by the application.

The following example shows the format of the Server Side XML UI Listing. The <uri> element in the Server Side XML UI Listing SHALL start with the value "/rcf/request_cui".

```
<?xml version="1.0" encoding="UTF-8"?>
<uilist xmlns="urn:schemas-upnp-org:remoteui:uilist-1-0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:schemas-
upnp-org:remoteui:uilist-1-0 CompatibleUIs.xsd">
  <ui>
    <uiID>4560-9876-1265-8758</uiID>
    <name>CoD Control UI Type 1</name>
    <description>Controlling the CoD contents</description>
    <protocol shortName="CE-HTML-1.0">
       <uri>/rcf/request_cui?url=http://21.31.24.55:5910/codcui1</uri>
       <protocolInfo>
         <relatedData xmlns="urn:schemas-ce-org:ce-html-server-caps-1-0"
                      xsi:schemaLocation="urn:schemas-ce-org:ce-html-server-caps-1-0
ServerProfiles.xsd">
            <profilelist>
              <ui_profile name="MD_UIPROF"/>
            </profilelist>
         </relatedData>
       </protocolInfo>
     </protocol>
  </ui>
  <ui>
     <uiID>2123-3679-3568-2121</uiID>
     <name>CoD Control UI Type 2</name>
     <protocol shortName="CE-HTML-1.0">
        <uri>/rcf/request_cui?url=http://21.31.24.55:5910/codcui2</uri>
        <protocolInfo>
           <relatedData xmlns="urn:schemas-ce-org:ce-html-server-caps-1-0">
              <profilelist>
                 <ui_profile name="MD_UIPROF"/>
              </profilelist>
           </relatedData>
        </protocolInfo>
      </protocol>
  </ui>
</uilist>
```

Below is example source code showing how an application can merge a Server Side XML UI Listing that it has retrieved with the OITF's built-in XML UI Listing.

```
var rcMgr;
var xmlhttp;

function init() {
    ...
    rcMgr = document.getElementById("rcfmanager");
    retrieveXMLUIListingFromServer("/iptv_app/xml_location/request_xml?xml=31",
mergeXMLUIListing);
    ...
}

function retrieveXMLUIListingFromServer(url, callbackFunc) {
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            if(xmlhttp.status == 200){
                callbackFunc(xmlhttp.responseText);
            }
        }
    }
    xmlhttp.open("GET", url, true);
```

```
    xmlhttp.send(null);
}

function mergeXMLUIListing(xmluilisting) {
    rcMgr.useServerSideXMLUIListing(xmluilisting, false);
}
<body onload="init();">
<object id="rcfmanager" type="application/oipfRemoteControlFunction"/>
...
```

8.4.3 Retrieving the Control UI

The process of retrieving a Control UI based on an OITF's built in XML UI Listing is described below:

1) The Remote Control Device sends the request to the DLNA RUIS for the XML UI Listing.

2) The Remote Control Device presents a UI based on the information in the XML UI Listing. The user selects an entry from the list.

3) The Remote Control Device sends the HTTP request containing the URI (which has been specified by the OITF itself) to the DLNA RUIS. The OITF returns the Control UI (from its internal memory).

4) The Remote Control Device presents the Control UI. This Control UI may be an application itself or may be a list of other available applications. In the latter case, the user selects a link from the Control UI.

5) The Remote Control Device sends the HTTP request containing the URI from the selected link to the DLNA RUIS. The OITF retrieves the DAE application from the IPTV Applications server and executes it.

6) The DAE application recognises that it needs to get the control UI.

7) The DAE application retrieves the Control UI from the IPTV Applications server.

8) The DAE application passes the Control UI received from the IPTV Applications server to the Remote Control Device.

The process of retrieving a Control UI based on a Server Side XML UI Listing is as below:

1) The Remote Control Device sends the request to the DLNA RUIS for the XML UI Listing.

2) The Remote Control Device presents a UI based on the information in the XML UI Listing. The user selects an entry from the list.

3) The Remote Control Device sends the HTTP request containing the URI (which must start with "`/rcf/request_cui`") to the OITF DLNA RUIS. The OITF matches the URI with the correct DAE application and passes the request to that DAE application as a `ReceiveRemoteMessage` event.

4) The DAE application translates the request which came from the Remote Control Device into a URI.

5) The DAE application retrieves the Control UI from the IPTV Applications server using this URI.

6) The DAE application passes the Control UI received from the IPTV Applications server to the Remote Control Device using `sendRemoteMessage()`.

More details can be found in Annex J.

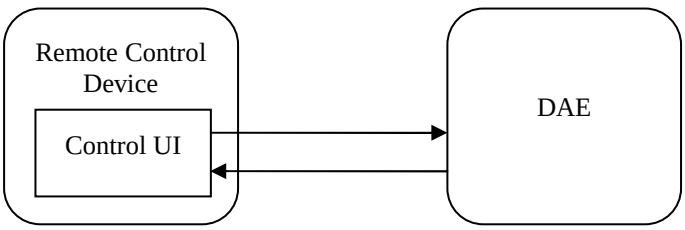
When the control UI (CE-HTML document) is being rendered in the Remote Control Device, it can retrieve resources (For example, image, css or javascript files) directly from the IPTV Applications server over a secure connection. For deployments where the IPTV Applications server is outside the consumer network, the consumer network's WAN gateway SHALL allow the DLNA RUIC to access the IPTV Applications server to retrieve resources directly. The Remote Control Device that connects to the IPTV Applications server SHALL implement the Secure Sockets Layer

(SSL) Protocol, the Transport Layer Security (TLS) and the “https:” URI scheme, in order to support secure Internet transactions (as defined in [Req. 5.1.2.b] of [CEA-2014-A]).

Below is example source code to show sending the control UI to the Remote Control Device.

```
var rcMgr;
var xmlhttp;
var deviceHandle;
var reqHandles = new Array();

function init() {
    ...
    rcMgr = document.getElementById("rcfmanager");
    rcMgr.addEventListener("ReceiveRemoteMessage", receiveRemoteMessageFromRD, false);

    // check whether the DAE app is launched by the Remote Control Device or not
    if (rcMgr.currentRemoteDeviceHandle != undefined) {
         deviceHandle = rcMgr.currentRemoteDeviceHandle;
         retrieveCUIFromServer("/iptv_applications/cui_location/request_cui?cui=123",
sendCUIToRemoteDevice);
    }
    ...
}

function retrieveCUIFromServer(url, callbackFunc){
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            if(xmlhttp.status == 200){
                callbackFunc(xmlhttp.responseText);
            }
        }
    }
    xmlhttp.open("GET", url, true);
    xmlhttp.setRequestHeader("X-OITF-RCF-User-Agent",
rcMgr.getRemoteDeviceUserAgent(deviceHandle));
    xmlhttp.send(null);
}

function sendCUIToRemoteDevice(cuiCEHTML) {
    rcMgr.sendRemoteMessage(remoteDeviceHandle, reqHandles.shift(), cuiCEHTML);
}

function receiveRemoteMessageFromRD(type, remoteDeviceHandle, reqHandle, requestLine,
headers, body) {
    if (type == 0) {
         deviceHandle = remoteDeviceHandle;
         reqHandles.push(reqHandle);

        // retrieve the CUI CE-HTML document from the IPTV Applications server
        retrieveCUIFromServer("/iptv_applications/cui_location/request_cui?cui=123",
sendCUIToRemoteDevice);
    }
}
<body onload="init();">
<object id="rcfmanager" type="application/oipfRemoteControlFunction"/>
...
```

8.4.4 Receiving and responding a message between the control UI in the Remote Control Device and OITF



This example shows the usage of receiving and responding to a message between the control UI presented on the Remote Control Device and the OITF. When the control UI sends a message to the DAE application via an HTTP request, the DAE application receives the message via a `ReceiveRemoteMessage` event. The DAE application SHALL return the

response to the control UI in the Remote Control Device by using the `sendRemoteMessage()` or `sendInternalServerError()` methods.

The OITF is not able to notify the Remote Control Device whether the DAE application has been terminated or deactivated, or whether the `application/oipfRemoteControlFunction` object has been removed from the application's DOM tree. For this reason, the Remote Control Device may be presenting an outdated copy of the control UI and could send a request from this outdated control UI. In this case, the OITF SHALL return a 500 response error code to the Remote Control Device.

The OITF SHALL limit the number of HTTP requests (from the control UI in the Remote Control Device) which have not been responded to by the DAE application. If there are any requests over this limit, the OITF SHALL automatically reject them and send an HTTP response (HTTP 500 - Internal Server Error) to the Remote Control Device. The OITF SHALL buffer at least 10 outstanding HTTP requests.

Note: Annex J.3 provides a procedure related to this example.

Below is example source code showing the handling of messages between the DAE application and the control UI that controls the DAE application.

DAE application

```
var rcMgr;
var reqHandles = new Array();

function init() {
    ...
    rcMgr = document.getElementById("rcfmanager");
    rcMgr.addEventListener("ReceiveRemoteMessage", getMessageFromRD, false);
    ...
}

function getMessageFromRD(type, remoteDeviceHandle, reqHandle, requestLine, headers,
body) {
    if (type == 1) {
        // Handling the received message with parameters (requestLine, headers, body)
        parseAndExecute(body);

        // Sending the proper return message to the Remote Control Device
        Var contentType = "Content-Type: text/plain\n"
        rcMgr.sendRemoteMessage(remoteDeviceHandle, reqHandle, contentType, "ok");
    }
}

function parseAndExecute(body) {
    //For example, the request from the RD contains the message related to "play of
audio" with JSON form (Ex: {'command':415})
    var retVal = eval("("+body+")");
    if (retVal.command == VK_PLAY) {
        document.getElementById("aid1").play(1);
    }
}

<body onload="init();">
<object id="rcfmanager" type="application/oipfRemoteControlFunction"/>
<object type="audio/mp4" id="aid1" data="http://www.avsource.com/audio/bgm.aac">
<param name="loop" value="infinite"/>
</object>
...
```

Control UI

```
var xmlhttp;

function sendPlay() {
    var msg = {'command':415};
    sendMessage("/rcf/request_msg", msg, receiveMsg);
}

function sendMessage(url, msg, callbackFunc){
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
```

```
            if(xmlhttp.status == 200){
                callbackFunc(xmlhttp.responseText);
            }
        }
    }
    xmlhttp.open("POST", url, true);
    request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send(msg);
}

function receiveMsg(msg) {
    alert("Received message from the DAE application: " + msg);
}
<body>
...
<input type="button" value="Play" onclick="javascript:sendPlay();">
...
```

8.4.5 Notification to the Remote Control Device

The `application/oipfRemoteControlFunction` object supports generating 3rd party multicast notifications and dispatching them to Remote Control Devices. The DAE application can make and send a notification to the Remote Control Devices by using the `sendMulticastNotif()` method.

If the DAE application wants to send a notification CE-HTML document to all of the Remote Devices, the DAE application SHALL set the `remoteDeviceHandle` parameter in the `sendMulticastNotif` method to -1.

Otherwise, if the DAE application wants to allow only targeted Remote Device (currently being connected to the DAE application) to retrieve the notification CE-HTML document, the DAE application set the proper `remoteDeviceHandle` parameter in the `sendMulticastNotif` method when it calls. Then, the OITF SHALL generate the notification URI with `devicehandle` and `daeid` parameters.

If the DAE application wants to send a notification CE-HTML document without storing it in the OITF, the DAE application executes the `sendMulticastNotif` method with null value in the `notifCEHTML` parameter. The OITF SHALL make the notification URI which contains a `dynamic` parameter with true value, otherwise false is set in the dynamic parameter.

Below is a generated notification URI based on parameter information in the `sendMulticastNotif` method.

- ?SendToTargetedRD&devicehandle=<target device handle value>&daeid=<DAE App ID>&dynamic=<true or false>

This URL is sent to the Remote Devices through the <`ruiEventURL`> element of the multicast notification event and the Remote Devices send requests to the OITF with this URL upon receiving it. When the OITF receives the requests from the Remote Devices, it SHALL return the notification CE-HTML document in case the handle of the Remote Device which sends the request is the same with the parameter value "<target device handle value>" in the HTTP request URL, otherwise the OITF SHALL return the HTTP 403 response.

Below is example source code to show that the only targeted Remote Device retrieves the notification CE-HTML document.

```
var rcMgr;
var xmlhttp;
var deviceHandle;
var reqHandles = new Array();

function init() {
    ...
    rcMgr = document.getElementById("rcfmanager");
    rcMgr.addEventListener("ResultMuticastNotif", resultMuticastNotifFromRD, false);
    ...
}

function sendTargetedNotif() {
    // A remoteDeviceHandle SHALL be set to -1 if the OITF wants to send the
notification CE-HTML UI to all of the Remote Devices
    // A remoteDeviceHandle SHALL be set to a specific value of the device handle if the
OITF wants to send the notification CE-HTML UI to the targeted Remote Control Devices
```

```
    var remoteDeviceHandle = rcMgr.currentRemoteDeviceHandle;
    var eventLevel = 0;
    var notifCEHTML = "<html>…</html>";
    var friendlyName = "Important notification";
    var profilelist = "<ui_profile name='MD_UIPROF'/>";

    rcMgr.sendMulticastNotif (remoteDeviceHandle , eventLevel, notifCEHTML, friendlyName,
profilelist);
}

function resultMuticastNotifFromRD(remoteDeviceHandle, reqHandle, dynamic) {
    if (dynamic != true) {
        alert("Notification is sent to the Remote Control Device well");
    } else {
        //Retrieve a notifcation CE-HTML UI from server
        ...
    }
}

<body onload="init();">
<object id="rcfmanager" type="application/oipfRemoteControlFunction"/>
...
```

8.4.6 Handling Multiple DAE applications and Multiple Remote Control Devices

The OITF SHALL dispatch requests from a Remote Control Device to the DAE application that it is currently controlling. Only one Remote Control Device SHALL communicate with a DAE application at any time although this could change over time as described below.

- Multiple Remote Devices SHALL not be mapped to a same DAE application at the same time. If a second Remote Control Device attempts to send an HTTP request to a DAE application which is already mapped to a different Remote Control Device, this request SHALL fail (the OITF sends an HTTP 500 response to the Remote Control Device).

- One Remote Device SHALL not be mapped to multiple DAE applications at the same time. If a Remote Device is currently connected to a DAE application and then attempts to make a request to another DAE application, this request SHALL fail (the OITF sends an HTTP 500 response to the Remote Device).

The OITF SHALL support three mechanisms to drop the connection between a Remote Control Device and a DAE application as follows:

- The Remote Control Device currently bound to the DAE application sends a pre-defined URL "`/rcf/drop_connection`".

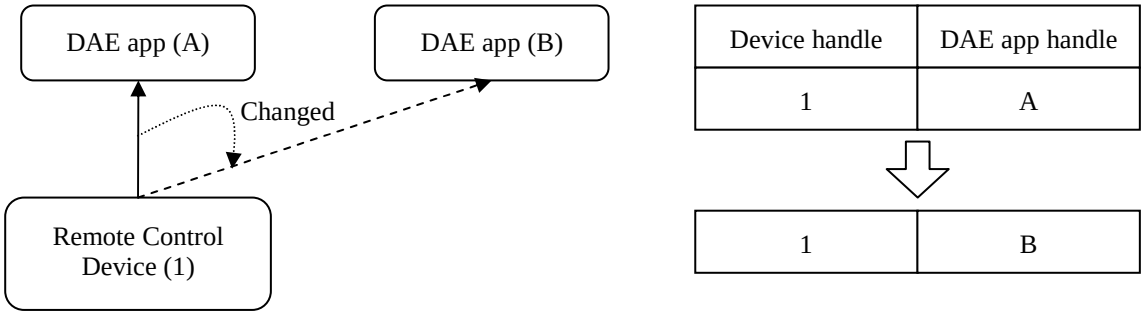
- The DAE application drops the connection with the Remote Control Device by using the `dropConnection()` method.

- The OITF provide a timer mechanism to drop the connection with the Remote Control Device after a period of inactivity (i.e. no HTTP requests received and no HTTP responses sent). The value of the inactivity timer expiry is terminal specific. One timer will be assigned per Remote Control Device.

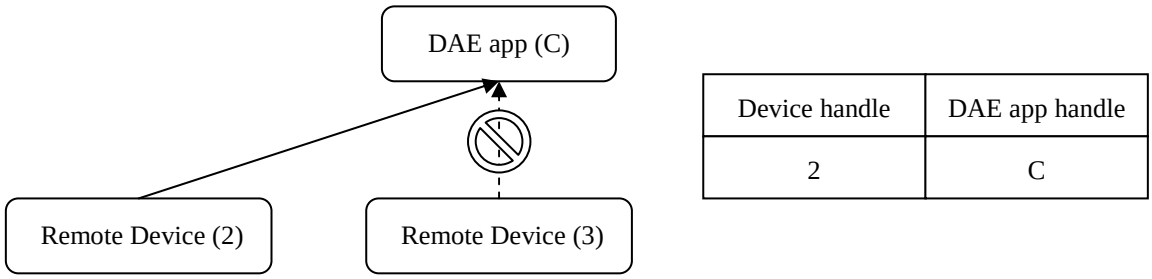
If the OITF is unable to dispatch requests to that application (e.g. because the application has terminated or because the `application/oipfRemoteControlFunction` object has been destroyed), the request SHALL fail (the OITF sends an HTTP 500 response to the Remote Control Device). If the OITF is notified that the Remote Control Device is no longer connected to the network, then the OITF SHALL allow other Remote Control Devices to connect to the application and assume control.

Below is example showing a mapping relationship between Remote Control Devices and DAE applications.

- Remote Control Device 1 is mapped to DAE application A. The Remote Control Device sends a request to drop the connection with A, using the pre-defined URL "`/rcf/drop_connection`" and then makes a request to DAE application B. DAE application B responds to the Remote Control Device. The OITF updates its internal state to show that Remote Control Device 1 is now mapped to DAE application B.



- Remote Control Device 2 is mapped to DAE application C. A second Remote Control Device 3 then makes a request to DAE application C. The OITF sends an HTTP 500 response to the Remote Control Device 3.



9 Capabilities

9.1 Minimum DAE capability requirements

This section defines minimum capabilities which OITF implementations are required to provide to the Declarative Application Environment and the applications running in that environment.

The following section defines minimum capabilities which SHALL apply to all OITFs.

OITFs MAY support multiple simultaneous applications loaded and running in the browser.

When the CEA-2014 notification framework (see section 5.3.1) is supported, OITFs SHALL support at least 2 DAE applications being visible at one time, one application showing a notification in the notification window (as defined in Section 5.6.3 of CEA-2014-A) and one in the main browser area. OITFs MAY support more than one DAE application being visible at one time in the main browser area. On OITFs where only one DAE application is visible at one time in the main browser area, it is OITF implementation specific how the visible application is changed.

OITFs with an HD output SHALL support 1280x720 graphics on that output when HD video is being decoded or when no video is being decoded. OITFs MAY support 1920x1080 graphics.

The present document does not define any requirements concerning support for SD graphics.

OITFs SHALL support unrestricted scaling of IP delivered video.

The present document does not define any requirements for scaling of video not delivered via IP, e.g. in hybrid OITFs.

The present document does not define requirements for supporting decoder format conversion.

The present document does not define requirements for pixel depth in the graphics system except that OITFs SHALL support at least one bit of per-pixel alpha.

The present document does not require the capability to mix audio from memory and audio from a currently decoded stream.

OITFs SHALL support decoding one stream containing video and audio. They MAY support decoding more than one stream.

OITFs SHALL support the “Tiresias Screenfont” font or equivalent with the “Generic Application Western European Character Set” as defined in Annex C of [TS 102 809]. They MAY support other fonts in addition.

OITFs SHALL provide some means for text input. The present document does not specify any particular solution.

The present document recommends support for pointer based input. Please note that Annex B contains some requirements regarding pointer based input.

In their SSL/TLS implementation, OITFs SHALL support

a) key lengths of up to 2048 bits for the asymmetric encryption part

b) for the symmetric part, at least 128-bit for AES and at least 168-bit for 3DES

c) for verifying server certificates, at least these root certificates:

1. Thawte Personal Basic CA
2. Thawte Personal Freemail CA
3. Thawte Personal Premium CA
4. Thawte Premium Server CA
5. Thawte Server CA
6. Thawte Timestamping CA

7. VeriSign, Inc. Class 1-3 Public Primary Certification Authority G1
8. VeriSign, Inc. Class 1-4 Public Primary Certification Authority G2
9. VeriSign, Inc. Class 1-4 Public Primary Certification Authority G3
10. RSA Security 2048 v3
11. RSA Security 1024 v3
12. Equifax Secure CA
13. Entrust.net CA
14. Entrust.net CA 2048
15. Entrust.net Client CA
16. GTE CyberTrust Global Root
17. Microsoft Root Authority

The present document does not define requirements for minimum memory sizes for DAE applications or OITF behaviour when available memory is low. This specification is deliberately silent about the conditions under which the `LowMemory` event defined in section 7.2.1.4 is generated.

OITFs SHALL provide support for cookies as defined in [RFC2109].

In addition to what is specified in [RFC2109], the following limits are defined:

- OITFs SHALL support at least 100 cookies with a maximum of 20 per domain and a maximum size for any individual cookie of 4096 bytes (as measured by the sum of the lengths of the cookie's name, value, and attributes).
- If the cookie is bigger than 4096 bytes it SHALL be discarded, not truncated.
- OITFs SHALL support a maximum total size for the “Set-Cookie” header of 5120 bytes. If the header is bigger than 5120 bytes, it SHALL be discarded, not truncated.

The present document does not require control of audio volume to be exposed to the DAE.

The OITF SHALL include a mechanism for the end user to generate the following key events:

- `VK_0` – `VK_9`
- `VK_UP`, `VK_DOWN`, `VK_LEFT`, `VK_RIGHT`, `VK_ENTER` ,`VK_BACK`
- `VK_RED`, `VK_GREEN`, `VK_YELLOW`, `VK_BLUE`

Because physical color keys may not always be available on remote controls, DAE applications which use the colour keys SHOULD make the same feature, function or link accessible through a button in their user interface which can be navigated to by `up`, `down`, `left` and `right` and selected with `enter` / `OK` and SHOULD make their intended usage known through the `Keyset` object as defined in Section 7.2.5

If the OITF includes a mechanism to generate the following key events then they SHALL be available to DAE applications and SHALL be indicated as part of the capability mechanism defined in section 9 of this specification.

- `VK_PLAY`, `VK_PAUSE`, `VK_STOP`, `VK_NEXT`, `VK_PREV`
- `VK_PLAY_PAUSE`
- `VK_FAST_FWD`
- `VK_REWIND`

Note: Some remote controls have separate “play” and “pause” keys; others have a single “play/pause” toggle key. For that reason, in general, it is recommended that applications are written to handle both the VK_PLAY/VK_PAUSE key codes and the VK_PLAY_PAUSE key code.

The OITF MAY include mechanisms to generate the following key events and if it does, making them available to DAE applications is OPTIONAL.

- VK_HOME
- VK_MENU
- VK_GUIDE
- VK_TELETEXT
- VK_SUBTITLES
- VK_CHANNEL_UP
- VK_CHANNEL_DOWN
- VK_VOLUME_UP
- VK_VOLUME_DOWN
- VK_MUTE

Where OITFs make other remote control key events available to DAE applications, this SHALL be done as specified by the capability mechanism defined in section 9 of this specification. Whenever applicable, this SHOULD be done using the complementary UI profiles defined in the next paragraph.

9.2 Default UI profiles

The OITF SHALL support at least one of the UI-related base profiles defined in Table 14.

Table 14: Base UI Profile Names

Base UI Profile Name	Default values
"OITF_SDEU_UIPROF"	<width>720</width> <height>576</height> <colors>high</colors> <hscroll>false</hscroll> <vscroll>true</vscroll> Tiresias with support for the Unicode character range “Basic Euro Latin Character set” as defined in Annex C of [TS 102 809]. <key>VK_BACK</key> <navigationkeys>true</navigationkeys> <numerickeys>true</numerickeys> <pointer>false</pointer>

	<security protocolNames="ssl tls">true</security> <overlay>per-pixel</overlay><!-- whereby at least one level of partial transparency between graphics and video must be supported as per the minimum requirements of Section 9.1 --> <overlaylocal>per-pixel</overlaylocal><!-- whereby at least one level of partial transparency between graphics and video must be supported as per the minimum requirements of Section 9.1 --> <overlaylocaltuner>per-pixel</overlaylocaltuner><!-- whereby at least one level of partial transparency between graphics and video must be supported as per the minimum requirements of Section 9.1 --> <overlayIPbroadcast>per-pixel</overlayIPBroadcast><!-- whereby at least one level of partial transparency between graphics and video must be supported as per the minimum requirements of Section 9.1 --> <notificationscripts>false</notificationscripts> <save-restore>false</save-restore>
"OITF_SD60_UIPROF"	Same as OITF_SDEU_UIPROF, with the following modifications: <width>720</width> <height>480</height>
"OITF_SDUS_UIPROF"	Same as OITF_SDEU_UIPROF, with the following modifications: <width>640</width> <height>480</height>
"OITF_HD_UIPROF"	Same as OITF_SDEU_UIPROF, with the following modifications: <width>1280</width> <height>720</height> <colors>high</colors> Tiresias Screenfont with support for the Unicode character range “Generic Application Western European Character Set” as defined in Annex C of [TS 102 809].
"OITF_FULL_HD_UIPROF"	Same as OITF_HD_UIPROF, with the following modifications: <width>1920</width> <height>1080</height>

In order to capture the heterogeneity of the features supported by OITF devices, this specification also defines a set of complementary UI Profile name fragments, each constituting a particular logical subset of capabilities, for which a OITF can indicate support by appending the UI Profile name fragment to the name of the supported base UI profile as defined in Table 14. Both the OITF and server SHALL support the concatenation of a series of UI profile name fragments in any order.

Table 15: Complementary UI Profile Name Fragments

UI Profile Name Fragment	Default values
"+TRICKMODE"	<key>VK_PLAY</key><key>VK_PAUSE</key> and/or <key>VK_PLAY_PAUSE</key> (*) <key>VK_STOP</key> <key>VK_REWIND</key> <key>VK_FAST_FWD</key> (*) The +TRICKMODE profile fragment identifier does not distinguish between remote controls having separate “play” and “pause” keys; and remote controls having a single “play/pause” toggle key. For that reason, in general, it is recommended that applications are written to handle both the VK_PLAY/VK_PAUSE key codes and the VK_PLAY_PAUSE key code
"+AVCAD"	<video_profile type="application/vnd.oipf.ContentAccessStreaming+xml"/>
"+DL"	<download protocolNames="http">true</download>
"+IPTV_SDS"	<video_broadcast type=”ID_IPTV_SDS” scaling=”arbitrary”>true</video_broadcast>
"+IPTV_URI"	<video_broadcast type=”ID_IPTV_URI” scaling=”arbitrary”>true</video_broadcast>
"+ANA"	<video_broadcast type=”ID_ANALOG” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_C"	<video_broadcast type=”ID_DVB_C ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_T"	<video_broadcast type=”ID_DVB_T ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_S"	<video_broadcast type=”ID_DVB_S ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_C2"	<video_broadcast type=”ID_DVB_C2 ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_T2"	<video_broadcast type=”ID_DVB_T2 ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+DVB_S2"	<video_broadcast type=”ID_DVB_S2 ID_DVB_SI_DIRECT” scaling=”quarterscreen”>true</video_broadcast>
"+ISDB_C"	<video_broadcast type=”ID_ISDB_C” scaling=”quarterscreen”>true</video_broadcast>
"+ISDB_T"	<video_broadcast type=”ID_ISDB_T” scaling=”quarterscreen”>true</video_broadcast>

"+ISDB_S"	<video_broadcast type=”ID_ISDB_S” scaling=”quarterscreen”>true</video_broadcast>
"+META_BCG”	<clientMetadata type=”bcg”>true</clientMetadata >
"+META_EIT”	<clientMetadata type=”eit-pf”>true</clientMetadata >
“+META_SI”	<clientMetadata type=”dvb-si”>true</clientMetadata >
"+ITV_KEYS"	<key>VK_HOME</key> <key>VK_MENU</key> <key>VK_CANCEL</key> <key>VK_SUBTITLES</key> <colorkeys>true</colorkeys>
"+CONTROLLED"	<key>VK_CHANNEL_UP</key> <key>VK_CHANNEL_DOWN</key> <key>VK_VOLUME_UP</key> <key>VK_VOLUME_DOWN</key> <key>VK_MUTE</key> <configurationChanges>true</configurationChanges> <extendedAVControl>true</extendedAVControl> When relevant (i.e. when coupled with +DL, resp +PVR): <download manageDownloads="sameDomain">true</download> <recording manageRecordings="sameDomain">true</ recording > <remote_diagnostics>true</remote_diagnostics>
”+PVR”	<key>VK_RECORD</key> <recording>true</recording>
"+DRM"	<drm DRMSystemID="urn:dvb:casystemid:19188">TS_BBTS TTS_BBTS MP4_PDCF</drm>
“+IMS”	<ims>true</ims>
“+SVG”	<mime-extensions>image/svg+xml</mime-extensions>
“+POINTER”	<pointer>true</pointer>
“+POLLNOTIF”	<pollingNotifications>true</pollingNotifications>
“+WIDGETS”	<widgets>true</widgets>

“+HTML5_MEDIA”	<html5_media>true</html5_media>
“+RCF”	<remoteControlFunction>true</ remoteControlFunction> (*) If an OITF supports the DLNA RUI RCF as defined in Section 7.17, the 3rd party multicast notification mechanism as defined in Section 5.6.1 of [CEA-2014-A] SHALL be supported for the OITF to send the 3rd party multicast notification to the DLNA RUICs.

Whenever an OITF supports an extension to the capabilities that can be defined using a combination of a base UI Profiles and a (number of) UI Profile fragment(s), it SHALL advertise this extension using the mechanism as defined in Section 8.1.

9.3 CEA-2014 capability negotiation and extensions

This section contains extensions and modifications to the CEA-2014 [CEA-2014-A] capability negotiation mechanism. The XML format that is used to describe the capabilities forms the basis for the profile definitions and profile fragments as defined in Section 9.2, and is also the format that is used by the “`xmlCapabilities`” property of the `application/oipfCapabilities` object.

The schema with the extensions and modifications to the capability description as defined in this section can be found in Annex F. The schema in Annex F SHALL be used instead of the existing capability description schema as defined in Annex C of CEA-2014 [CEA-2014-A].

The conveyance of the OITF capability description through the User-Agent header is described in Section 8.1.

Examples of valid OITF capability profiles are (using the full XML syntax as defined in Annex F):

A pure HD-capable IPTV OITF, which supports live DVB-IP TV via SD&S, streamed mpeg at SD and HD formats, the MPAA parental rating scheme, trickplay, and access to an embedded BCG metadata client:

```
<profilelist>
    <ui_profile
      name="OITF_HD_UIPROF+IPTV_SDS+AVCAD+META_BCG+TRICKMODE+ITV_KEYS+CONTROLLED+DRM">
        <ext>
            <parentalcontrol schemes="urn:mpeg:mpeg7:cs:MPAAParentalRatingCS:2001">
               true
            </parentalcontrol>
        </ext>
    </ui_profile>
    <video_profile name="TS_AVC_SD_25_HEAAC"
                   type="video/mpeg"
                   transport="http-get rtsp-rtp-udp"
                   DRMSystemID="urn:dvb:casystemid:19188"/>

    <video_profile name="TS_AVC_HD_25_HEAAC"
                   type="video/mpeg"
                   transport="http-get rtsp-rtp-udp"
                   DRMSystemID="urn:dvb:casystemid:19188"/>
</profilelist>
```

A hybrid HD-capable box, supporting live DVB broadcasts over satellite, PVR functionality, and (Marlin-protected and unprotected) VoD in progressive download:

```
<profilelist>
    <ui_profile
      name="OITF_HD_UIPROF+AVCAD+TRICKMODE+ITV_KEYS+CONTROLLED+DRM+DVB_S+META_SI+PVR">
    </ui_profile>
    <video_profile name="TS_AVC_SD_25_HEAAC"
                   type="video/mpeg"
                   transport="http-get rtsp-rtp-udp"
                   DRMSystemID="urn:dvb:casystemid:19188"/>

    <video_profile name="TS_AVC_HD_25_HEAAC"
                   type="video/mpeg"
                   transport="http-get rtsp-rtp-udp"
                   DRMSystemID="urn:dvb:casystemid:19188"/>
```

```
</profilelist>
```

A hybrid device providing access to its ATSC terrestrial tuner (supporting two different parental rating schemes), DVB-IPTV 'tuner', and PVR functionality to DAE applications, but not exposing 'trickmode' or 'controlled' key events to DAE applications running in the browser:

```
<profilelist>
    <ui_profile name="OITF_HD_UIPROF+PVR+IPTV_SDS">
        <ext>
            <video_broadcast type="ID_ATSC_T" scaling="arbitrary">true</video_broadcast>
            <parentalcontrol schemes="urn:mpeg:mpeg7:cs:MPAAParentalRatingCS:2001
                                      urn:mpeg:mpeg7:cs:MPAAParentalRatingTVCS:2001">
                true
           </parentalcontrol>
        </ext>
    </ui_profile>
</profilelist>
```

9.3.1 Tuner/broadcast capability indication

If an OITF supports control over its local tuner functionality by a server, an OITF SHALL indicate this through the base profile and UI profile name fragment strings as defined in section 9.2 "Default UI Profiles" and the schema defined in Annex F. To this end the following new elements SHALL be supported for a capability description or capability profile (see Annex F for more information):

<video_broadcast> - indicates whether or not the OITF supports the video/broadcast object to enable control of its local tuner functionality by a server (i.e. retrieving the tuner's channel line up, switching channels of the tuner, and rendering the output of the broadcasted content inside the browser). The <video_broadcast> element has the following attributes:

- Attribute *type* specifies the type(s) of tuner(s) for which the OITF allows tuner control, by using a space-separated list of idType values as specified in Section 7.13.12.1 for the `Channel` object (i.e. "ID_ANALOG", "ID_DVB_C", etc.).

- Attribute *transport* specifies a space-separated list of supported (transport) protocols in case of IP Broadcasts (i.e. if the type attribute contains one of the ID_IPTV_* idType values as specified in Section 7.13.12.1). This is done by using one or more of the (transport) protocol names as defined in Annex F of the [Protocols specification].

- Attribute *scaling* specifies the method of video scaling the OITF supports for the tuner output (i.e. "arbitrary", "quartersize", "0.33x0.33" or "none"), with default value "arbitrary" if omitted.

- Attribute *minSize* specifies the minimal size, as a percentage of the full extent of the OITF's display, to which the OITF supports scaling of video content received over the (logical or physical) tuner if attribute *scaling* has value "arbitrary". The value "0" for the *minSize* attribute indicates support for arbitrary and unrestricted scaling of the video. The value of the attribute *minSize* SHALL be silently ignored if the value of the attribute *scaling* is not "arbitrary".

- Attribute *nrstreams* provides an indication of the number of video streams that can be rendered simultaneously by the indicated tuner functionality (typically limited by the number of tuners supported by the device), with a default value of "1" if omitted.

- Attribute *postList* specifies, if included in the client's capability description, whether or not the OITF supports the HTTP POST method defined in Section 4.8.1.2. If included in the server's capability description, *postList* specifies whether or not the server supports using the channel list information sent through the HTTP POST method to exercise tuner control. If an OITF does not post the channel list information, a server SHALL, irrespective of the value it specified for the *postList* attribute in its server capability description, rely on the `getChannelConfig` method defined in Section 7.13.1.3 to access the channel list information.

- Attribute *localTimeshift* indicates whether or not the OITF supports timeshift of scheduled content using local storage.

- Attribute *networkTimeshift* indicates whether or not the OITF supports network timeshift of scheduled content. Different from PVR or local timeshift capability in that no local resources are required to support network timeshift

The <video_broadcast> element is defined using the following XML Schema fragment. Multiple <video_broadcast> elements may be specified to distinguish between tuners with different behaviour or capabilities, for example with respect to scaling:

```
<xs:element name="video_broadcast" type="videoBroadcastType" minOccurs="0"
maxOccurs="unbounded"/>
<xs:complexType name="videoBroadcastType">
    <xs:attribute name="type" type="xs:string" use="required"/>
    <xs:attribute name="transport" type="xs:string"/>
    <xs:attribute name="nrstreams" type="xs:unsignedInt" default="1"/>
    <xs:attribute name="scaling" type="scalingType" default="arbitrary"/>
    <xs:attribute name="minSize" type="xs:unsignedInt" default="0"/>
    <xs:attribute name="postList" type="xs:boolean" default="false"/>
    <xs:attribute name="networkTimeshift" type="xs:boolean" default="false"/>
    <xs:attribute name="localTimeshift" type="xs:boolean" default="false"/>
</xs:complexType>
```

<overlaylocaltuner> - indicates whether or not the OITF supports overlays for video broadcasts received through the local tuner, i.e. allows XHTML content to be rendered on top of video content broadcasted over local tuner. If included, the value of this element SHALL be: (none|on-off|global|per-pixel), whereby the same requirements as defined for element <overlay> in [Req. 5.2.1.a] of CEA-2014-A SHALL apply.

NOTE: As defined by [Req. 5.2.1.e] of CEA-2014-A also a server MAY use these elements in the server capability description, if a server requires control of the tuner functionality of an OITF for the correct rendering of its service.

9.3.2 Broadcast content over IP capability indication

If an OITF supports functionality for rendering the output of the broadcasted content received over IP inside the browser and optionally providing an IPTV related channel line-up and favourite list to the server, an OITF SHALL indicate this through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the schema defined in Annex F. This SHALL be done using the same <video_broadcast> element as defined in Section 9.3.1, whereby the type attribute contains one of the ID_IPTV_* idType values as specified in Section 7.13.12.1:

- **<video_broadcast>** - indicates whether or not the OITF supports the video/broadcast object to enable control rendering the output of the broadcasted content received over IP inside the browser and optionally providing an IPTV related channel line-up and favourite list to the server.

To indicate support for overlays over IP broadcasts the following element SHALL be used (see Annex F for more information):

- **<overlayIPbroadcast>** - indicates whether or not the OITF supports overlays for IP video broadcasts, i.e. allows XHTML content to be rendered on top of video content broadcasted over IP. If included, the value of this element SHALL be: (none|on-off|global|per-pixel), whereby the same requirements as defined for element <overlay> in [Req. 5.2.1.a] of CEA-2014-A SHALL apply.

9.3.3 PVR capability indication

Support for the control of recording functionality that is available to the OITF by a server SHALL be indicated through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <recording> element defined in Annex F. This specification defines the following element that can be added to a capability description:

<recording> - indicates whether or not the OITF supports control of its local recording (i.e. PVR) functionality by a server. If included, the value of this element SHALL be (true|false). The boolean attribute *ipBroadcast* specifies whether or not the OITF also supports recording of A/V content broadcasted over IP, and the Boolean attribute *postList* specifies whether or not the OITF supports the HTTP POST method defined in Section 4.8.2, respectively whether or not the server uses the posted channel list information, if conveyed by the OITF, to control the recording functionality available to the OITF. If an OITF does not post the channel list information, a server SHALL, irrespective of the value it specified for the *postList* attribute, rely on the `getChannelConfig()` method defined in Section 7.10.1.1 to access the channel

list information. The Boolean attribute *manageRecordings* specifies whether or not the OITF supports managing recordings through the ECMAScript APIs defined in section 7.10.4.

The <recording> element is defined using the following XML Schema fragment (see Annex F for more information):

```
<xs:element name="recording" type="pvrType"/>
  <xs:complexType name="pvrType">
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:attribute name="ipBroadcast" type="xs:boolean" default="false"/>
        <xs:attribute name="manageRecordings" type="xs:string" default="none"/>
        <xs:attribute name="postList" type="xs:boolean" default="false"/>
      </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

If the `manageRecordings` attribute is present, this attribute SHALL take one of the following values:

- **"none"**: indicates that the client does not support managing recordings.
- **"initiator"**: indicates that recordings initiated by the current application may be managed.
- **"samedomain"**: indicates that recordings initiated by applications from the same fully-qualified domain may be managed.
- **"all"**: indicates that recordings initiated both by the current application and other applications may be managed.

If not present, a value of "**none**" SHALL be assumed.

9.3.4 Download CoD capability indication

If a client supports downloading content to a client (with or without DRM protection), the client SHALL indicate this through the base profile and UI profile name fragment strings as defined in section 9.2 "Default UI Profiles" and the schema defined in Annex F. The <download> element SHALL adhere to the definition of bullet o) of [Req. 5.2.1.a] of CEA-2014-A.

A client MAY include an informative list of MIME types it supports for playback after download through the <mime-extensions> element. Note that since content download may be separated from content playback, a server SHOULD NOT rely on this information to be present.

If a client supports managing downloads through the ECMAScript content download API specified in Section 7.4.3 then the client SHALL indicate this using the attribute manageDownloads. This attribute has the following definition (see Annex F for more information):

```
<xs:attribute name="manageDownloads" type="xs:string" default="none"/>
```

If present, this attribute SHALL take one of the following values:

- **"none"**: indicates that the client does not support managing downloads**.**
- **"initiator"**: indicates that downloads initiated by the current application may be managed.
- **"samedomain"**: indicates that downloads initiated by applications from the same fully-qualified domain may be managed.
- **"all"**: indicates that downloads initiated both by the current application and other applications may be managed.

If not present, a value of "none" SHALL be assumed.

Example:

```
<download protocolNames="http ftp" manageDownloads="all" > true </download>
```

9.3.5 Parental ratings

If an OITF supports a parental control system, the OITF SHALL indicate this by using the value “true” for element <parentalcontrol> in the OITF capability profile/description, and define a space separated list of names of parental rating schemes using the “schemes” attribute.

The schema of the <parentalcontrol> element is defined as follows (see Annex F for more information):

```
<xs:element name="parentalcontrol" type="parentalControlType"/>
<xs:complexType name="parentalControlType">
     <xs:simpleContent>
         <xs:extension base="xs:boolean">
              <xs:attribute name=”schemes” type=”xs:string”/>

         </xs:extension>
     </xs:simpleContent>
</xs:complexType>
```

For which the following semantics SHALL apply:

<parentalcontrol> - indicates whether or not the OITF supports a client controlled parental control system. If included in the OITF capability description, the value of this element SHALL be: (true|false). The <parentalcontrol> element has the following attributes:

- attribute **“schemes”**: SHALL be a non-empty space separated list of case-insensitive names of parental rating schemes registered with the platform (either by the manufacturer, or by applications where the rating scheme is associated with a recording), if the value of the <parentalcontrol> element is true. Valid rating schemes names include the ParentalRating classification scheme names as defined by property “scheme” of the `ParentalRating` object as defined in Section 7.9.4.

  Example:

```
<parentalcontrol schemes="dvb-si urn:mpeg:mpeg7:cs:MPAAParentalRatingCS:2001">
    true
</parentalcontrol>
```

9.3.6 Extended A/V API support

The OITF SHALL indicate support for the extended A/V control APIs defined in section 7.13.7 through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <extendedAVControl> element defined in Annex F:

```
<xs:element name="extendedAVControl" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.7 OITF Metadata API support

The OITF SHALL indicate support for client-side metadata processing and the APIs defined in section 7.12 through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <clientMetadata> element defined in Annex F:

```
<xs:element name="clientMetadata" type="metadataType"/>
<xs:complexType name="metadataType">
         <xs:simpleContent>
             <xs:extension base="xs:boolean">
                <xs:attribute name="type" type="xs:string"/>
             </xs:extension>
         </xs:simpleContent>
</xs:complexType>
```

This element has the following semantics:

<clientMetadata> - indicates whether or not the OITF supports a client-side metadata processing. If included in the RUI Client capability description, the value of this element SHALL be: (true|false). The <clientMetadata> element has the following attributes:

- attribute **“type”** SHALL include a non-empty space separated list of names of supported metadata systems/protocols, if the value of the <clientmetadata> element is true.

  Below is an extensible list of case insensitive metadata system/protocol names which MAY be used for this attribute:

  - **“bcg”**: indicates support for the TV-Anytime Broadband Content Guide metadata format.
  - **“sd-s”**: indicates support for the DVB SD&S metadata format.
  - **“dvb-si”**: indicates support for the DVB-SI metadata format.
  - **“eit-pf”**: indicates support for EIT present/following information as defined for DVB-SI in Section 4.1.3 of [OIPF_META2]

9.3.8 OITF Configuration API support

The OITF SHALL indicate support for modification of OITF configuration and settings by applications (via the APIs defined in section 7.3) through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <configurationChanges> element defined in Annex F:

```
<xs:element name="configurationChanges" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.9 IMS API Support

The OITF SHALL indicate support for IMS API (via the APIs defined in section 7.8) through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <ims> element defined in Annex F:

```
<xs:element name="ims" type="xs:boolean"/>
<xs:element name="communication_services" type="xs:boolean"/>
```

If included, the value of these elements SHALL be: (true|false).

9.3.10 DRM capability indication

The OITF SHALL indicate support for handling DRM-protected content through the base profile and UI profile name fragment strings as defined in section 9.2 “Default UI Profiles” and the <drm> element defined in Annex F:

```
<xs:element name="drm" type="drmType" minOccurs="0" maxOccurs="unbounded"/>
<xs:complexType name="drmType">
        <xs:simpleContent>
            <xs:extension base="xs:string">
               <xs:attribute name="DRMSystemID" type="xs:string" use="required"/>
               <xs:attribute name="protectionGateways" type="xs:string" default=""/>
            </xs:extension>
        </xs:simpleContent>
</xs:complexType>
```

And with the following semantics:

<drm> - indicates whether or not the client supports a DRM content protection system for downloading and streaming content. If included in the RUI Client capability description, the value of this element SHALL be a space separated list of zero or more case-insensitive names of supported file and/or container formats for protected content by the DRM system indicated by the "DRMSystemID" attribute, such as the OMA DRM Content Format (DCF). Valid values include: the system_format name of the first column of Table 3 of [MEDIA], and a protection format of the second column of Table 3 of [MEDIA], concatenated with an underscore ‘_’. In case of the Gateway centric approach defined by [OIPF_CSP2], this attribute indicates the protectionFormats which are supported by the combination of OITF and CSP Gateway and may be omitted.

The <drm> element has the following attributes:

- attribute **“DRMSystemID”** SHALL include a supported DRM system. Valid values for the "DRMSystemID" include the values as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is “urn:dvb:casystemid:19188”. In case of the Gateway centric

approach defined by [OIPF_CSP2], this DRMsystemID attribute indicates the DRM System(s) of UNIS-CSP-G which is supported by the combination of OITF and CSP Gateway.

- attribute **"protectionGateways"** SHALL include a space separated list of zero or more case-insensitive names of supported CSP Gateway types that are capable of supporting the DRM system indicated by attribute "`DRMSystemID`". This attribute is conditional mandatory and SHALL be specified in the case that the DRM System indicated by the "`DRMSystemID`" attribute is supported by the CSP Gateway. Valid values for the scheme for the Gateway centric approach defined by [OIPF_CSP2] are "`dtcp-ip`" and "`ci+`".

Examples:

```
<drm DRMSystemID="urn:dvb:casystemid:19188" >TS_BBTS TTS_BBTS MP4_PDCF</drm>
<drm DRMSystemID="urn:dvb:casystemid:12348" protectionGateways="ci+">TS_PF TTS_PF</drm>
<drm DRMSystemID="urn:dvb:casystemid:12348" protectionGateways="dtcp-ip">TS_PF</drm>
```

9.3.11 Media profile capability indication

If an OITF supports streaming A/V content to the client, the client SHALL indicate this by including a non-empty list of <audio_profile> and/or <video_profile> elements in the RUI client capability description. The <audio_profile> and <video_profile> elements SHALL adhere to the following requirements in addition to what has been defined by bullet v) and w) of [Req. 5.2.1.a] of CEA-2014-A:

- Valid values for the "type"-attribute of the <audio_profile> and <video_profile> elements include the MIME types given in Section 3 of [OIPF_MEDIA2].

- Valid values for the "name"-attribute include:
  - for <video_profile> elements: the system format name, the video format name and the audio format name for A/V contents, concatenated with an underscore '_', as defined in Section 3 of [OIPF_MEDIA2].
  - for <audio_profile> elements: the audio format name for pure audio contents in Table 4 of [OIPF_MEDIA2]
  - for both <video_profile>, and <audio_profile> elements, it is allowed to include multiple profile names corresponding to the same MIME type, by separating each profile name with a whitespace character.
- Valid values for the "transport"-attribute include (a space-separated list of) the protocol names as defined in the column "Name for <protocol>" in Annex E.1 of [OIPF_PROT2], whereby the value "http" as specified as default value for the "transport"-attribute in CEA-2014-A SHALL correspond to value "http-get".

- The <video_profile> and <audio_profile> elements SHALL support a new attribute called "DRMSystemID", which SHALLinclude a space separated list of zero or more DRM system IDs supported for the media profile(s), whereby the DRMSystemID SHALL correspond to a <drm> element (as defined in section 9.3.10. about DRM capability indication) with the same value for attribute "DRMSystemID". In the case the attribute "DRMsystemID" is specified, non-protected A/V contents of the media profile(s) SHALL be also supported. For non protected media profile(s), this attribute MAY be omitted (see Annex F for more information).

- Next to providing the list of supported audio and video profiles, the client SHALL include an <audio_profile> element and/or a <video_profile> element with the value "application/vnd.oipf.ContentAccessStreaming+xml" for attribute "type", to indicate support for the content access description document format as defined in 4.7.1 as value for the "data" attribute of the A/V Control object as defined by [CEA-2014-A] to initiate the streaming of content.

  Examples:

```
<video_profile type="application/vnd.oipf.ContentAccessStreaming+xml"/>

<video_profile
    name="TS_MPEG2_SD_25_AC3 TS_AVC_HD_25_HEAAC"
    type="video/mpeg"
    DRMSystemID="urn:dvb:casystemid:19188"
    transport="rtsp-rtp-udp"
/>

<video_profile
    name="MP4_MPEG2_SD_25_AC3 MP4_AVC_HD_25_HEAAC"
    type="video/mp4"
    transport="http-get"
/>

<video_profile
    name="TS_AVC_HD_25_HEAAC"
```

```
        type="application/x-dtcp1"
        DRMSystemID="urn:dvb:casystemid:12348"
        transport="http-get"
    />

    <audio_profile name="MPEG1_L3" type="audio/mpeg" transport="http-get"/>
```

9.3.12 Remote diagnostics support

The OITF SHALL indicate support for remote diagnostics (via the APIs defined in section 7.11) using the following element in the OITF's capability description (see Annex F for more information):

```
<xs:element name="remote_diagnostics" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.13 SVG

The OITF SHALL indicate support for SVG through the base profile and UI profile name fragment strings as defined in section 9.2 or as defined in section 6.4 using the Remote UI Client Capability Description defined for SVG in that section - `image/svg+xml`.

In order to determine support for video tag in SVG the `hasFeature()` method with argument "`http://www.w3.org/Graphics/SVG/feature/1.2/#Video`" shall be used. Example:

```
var hasvideo =
document.implementation.hasFeature("http://www.w3.org/Graphics/SVG/feature/1.2/#Video",
                                   null)
```

9.3.14 Third party notification support

If an OITF supports the 3rd party polling mechanism as defined in Section 5.6.2 of [CEA-2014-A], including the extensions to 5.6.2 as defined in Annex B, through the base profile and UI profile name fragment strings as defined in section 9.2 "Default UI Profiles" and the <pollingNotifications> element defined in Annex F:

```
<xs:element name="pollingNotifications" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.15 Multicast Delivery Terminating Function support

The OITF SHALL indicate support for the multicast delivery terminating function (via the APIs defined in section 7.15.1) using the following element in the OITF's capability description (see Annex F for more information):

```
<xs:element name="mdtf" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.16 HTML5 video

The OITF SHALL indicate support for HTML5 video through the base profile and UI profile name fragment strings as defined in section 9.2 and the <html5_media> element as defined in Annex F:

```
<xs:element name="html5_media" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.17 DLNA RUI Remote Control Function support

The OITF SHALL indicate support for the DLNA RUI RCF (via the APIs defined in section 7.17) using the following element in the OITF's capability description (see Annex F for more information):

```
<xs:element name="remoteControlFunction" type="xs:boolean"/>
```

If included, the value of this element SHALL be: (true|false).

9.3.18 Power Consumption

The OITF SHALL indicate support for wake-up using the following elements in the OITF's capability description (see Annex F for more information):

```
<xs:element name="wakeupApplication" type="xs:boolean"/>
<xs:element name="wakeupOITF" type="xs:boolean"/>
<xs:element name="hibernateMode" type="xs:boolean"/>
```

If included, the value of these elements SHALL be: (true|false).

9.3.19 Other capability extensions

The following extensions to the capability profile elements defined in [Req. 5.2.1.a] of CEA-2014-A SHALL be supported:

a) an additional value "0.33x0.33" for attribute "scaling" of the <video_profile> element in bullet w) of [Req. 5.2.1.a], with the following related extension to the schema for type "scalingType" (see Annex F for more information):

```
<xs:enumeration value="0.33x0.33"/>
```

9.3.20 Widgets

The OITF SHALL indicate support for Widget APIs through the base profile and UI profile name fragment strings as defined in section 9.2 "Default UI Profiles" and the <widgets> element defined in Annex F:

```
<xs:element name="widgets" type="xs:boolean"/>
```

If included, the value of these elements SHALL be: (true|false).

Widget APIs are the following Widget related methods/attributes defined in sections 7.2.1 and 7.2.2:

- `ApplicationManager.onWidgetInstallation`
- `ApplicationManager.onWidgetUninstallation`
- `ApplicationManager.installWidget`
- `Application.startWidget`
- `Application.stopWidget`
- `ApplicationManager.uninstallWidget`
- `ApplicationManager.widgets`

10 Security

10.1 Application / Service Security

This section defines the security model that applies to the privileged functionality exposed by an OITF to a server device. The main purpose of the security model is to protect local client side functionality exposed by an OITF to Javascript from unauthorized use. For example in the case of PVR control API, untrusted servers should be prevented from scheduling recordings.

The security model is quite generic, in a sense that it is not limited to particular privileged browser extensions, but can be applied to any local client side functionality exposed to any kind of networked application.

NOTE: The security model makes use of X509v3 certificates over TLS. Management of TLS root certificates, and which certificate authorities to trust is out of scope of this document.

10.1.1 OITF requirements

The following requirements SHALL apply to OITFs that expose security and/or privacy sensitive (i.e. privileged) functionality in one or more of the cases described in section 10.1.4.

- An OITF SHALL prevent a HTML document from a server from accessing the exposed security and/or privacy sensitive functionality, unless the server can be correctly authenticated (see below), and the server is granted the necessary privileges to access the security and/or privacy sensitive functionality.
- The OITF SHALL authenticate the server during a TLS handshake through a valid X.509v3 certificate, that is granted by a certificate authority that is trusted by the OITF. To this end, the OITF SHALL match the hostname or (sub)domainname of the HTML document's URI with the hostname or (sub)domainname as specified in the X.509v3 certificate, in the manner as defined in Section 3.1 of IETF RFC 2818.
- The OITF SHALL support the Online Certificate Status Protocol (OCSP), at least the Lightweight Profile as defined in RFC 5019, to determine the current validity of the X.509v3 certificate before access to privileged functionality is granted.
- The OITF MAY support a private certificate extension for X.509v3 certificates called "permissions" that specifies a set of permissions requested by a server to access privileged functionality, through zero or more permission names associated with privileges. The OITF MAY grant an authenticated server the set of permissions, which are each associated with the right to access a specific set of privileged functionality. Allowed permissions names include the permission names as defined in Section 10.1.5.
- The set of permissions granted to an authenticated server by an OITF MAY depend on the occurrence of that server on a whitelist or blacklist available to the OITF.
- NOTE: Management of whitelists and blacklists available to an OITF is out of scope of this document.
- If the server does not have the necessary privileges to access a property, method or object, or the server cannot be properly authenticated, the OITF SHALL throw an error with the name property set to the value "SecurityError". The example below shows how this can be used by applications:

```
try {
  object.foo()
} catch(e)
{
  if (e.name == "SecurityError") {
    // I am not authorised to do this
  }
}
```

- The OITF MAY inform the user of the decision to deny a server requested access to privileged functionality and MAY offer the user the option to override this decision.

10.1.2 Server requirements

The following requirements SHALL apply to servers that wish to access security and/or privacy sensitive (i.e. privileged) functionality exposed by an OITF, in one or more of the cases defined in Section 10.1.4:

- A server SHALL specify the use of TLS for each HTML document that accesses privileged functionality (i.e. by using the “https://” URI scheme for the URL of the HTML document).
- A server SHALL expose a valid X.509v3 certificate during the TLS certificate handshake.
- A server MAY request an OITF for certain permissions to access privileged functionality through a private certificate extension. If a server wants to do so, the server MAY include a private certificate extension called “permissions” as part of a valid X.509v3 certificate. If included, the “permissions” extension specifies a set of permissions through zero or more permission names. Allowed permissions names include the permission names as defined in Section 10.1.5.

10.1.3 Loading documents from different domains

The contents of an `<iframe>`, `<embed>` or `<object>` element may be retrieved from an FQDN other than the one from which the top-level document is loaded. In this case, the OITF SHALL enforce security restrictions between the contents of the element and the parent document. These restrictions may be based on the nested browsing context as defined in Section 6.1.1 of [HTML5] and the security restrictions formalised in Section 6.3.1 of [HTML5], excluding the features not included in this specification.

Documents SHALL be assigned the permissions associated with the FQDN from which they were loaded, as defined in section 10.1.1, rather than the permissions associated with the initial document of the application. For example documents loaded in an `<iframe>` element may be granted a different set of permissions from the top-level document that contains the `<iframe>` element. Similarly, following a link to a document from a different FQDN may result in the newly-loaded document having a different set of permissions than those granted to the previous document even though they are within the same application boundary.

As described in section 5.1.3, for files requested with `XMLHttpRequest`, the Same-Origin Policy SHALL be extended using the application domain as defined in section 5.1.3.

10.1.4 Specific security requirements for privileged Javascript APIs

This section defines the specific security requirements for specific privileged Javascript APIs, such as the tuner/broadcast, recording, content download and DRM related APIs as defined in Sections 7.13, 7.10, 7.4 and 7.6 in addition to the security requirements defined in sections 10.1.1 and 10.1.2.

10.1.4.1 Security requirements for tuner control and lineup

Exposure of the channel line up and the video/broadcast APIs for controlling the (local) tuner as specified in Section 7.13 SHALL adhere to the security requirements in Sections 10.1.4.1.1 and 10.1.4.1.2.

10.1.4.1.1 Security requirements for exposure of the tuner channel lineup

Exposure of the channel line up of the (local) tuner as specified in Section 7.13 SHALL adhere to the following security requirements:

the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to obtain the channel lineup of the (local) tuner. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL:

- not convey the Client Channel Listing to the server through a HTTP POST.
- not expose the Client Channel Listing to the DAE application through the `getChannelConfig()` method of the video/broadcast object. Attempts to access this method SHALL throw an error as defined in section 10.1.1.

10.1.4.1.2 Security requirements for tuner control

Control of the (local) tuner as specified in Section 7.13 SHALL adhere to the following security requirements:

- the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to control the (local) tuner. If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL deny requests to switch a local tuner to another channel by throwing an error as defined in section 10.1.1.

10.1.4.2 Security requirements for recording

The recording functionality as specified in Section 7.10 SHALL adhere to the following security requirements:

- *Recording of broadcasted content:* the OITF SHALL perform a security check (as defined by Section 10.1.1) to see if the server has the necessary privileges to schedule recordings of broadcasts. If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL deny a server's request to access the functionality of the `application/oipfRecordingScheduler` object (as defined by Section 7.10.1), and SHALL also not expose the Client Channel Listing, neither through the HTTP POST, nor through the `getChannelConfig()` method. Furthermore, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from the server attempts to access any properties or methods on the `application/oipfRecordingScheduler` object.

- *Recording of current A/V content broadcasted:* the OITF SHALL perform a security check (as defined by Section 10.1.1) to see if the server has the necessary privileges to record the current broadcast (as defined in Section 7.13.2). If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL deny a server's request to start a recording of the broadcast currently rendered by the `video/broadcast` object by throwing an error as defined in section 10.1.1.

- *Control over and exposure of scheduled recordings:* the OITF SHALL restrict the visibility and control over scheduled recordings to those scheduled recordings that were initiated through a server from the same FQDN that scheduled the recordings.

10.1.4.3 Security requirements for content download functionality

The content download functionality as defined in Section 7.4 SHALL adhere to the following security requirements:

- *Initiating a download:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to initiate a download. If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL NOT start downloading the content after receiving a content-access description document as defined in Section 4.6.2.

NOTE 1: The server is the server that served the HTML document or third-party notification that includes a link to a content-access description document. This is not necessarily the same server from which the content is downloaded.

NOTE 2: The URL from which a content item is downloaded (i.e. as specified by a <ContentURL> element in the content-access description document) does not have to be protected by TLS.

10.1.4.4 Security requirements for DRM related functionality

The DRM control functionality (i.e. the `application/oipfDrmAgent` embedded object) as defined in Section 7.6 SHALL adhere to the following security requirements:

- *Accessing the DRM agent:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the DRM agent, i.e. by accessing the DRM agent embedded object as specified in Section 7.6.1. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of its properties or methods on the DRM agent embedded object.

10.1.4.5 Security requirements for IMS functionality

The IMS functionality (i.e. the `application/oipfIMS` embedded object) as defined in Section 7.8 SHALL adhere to the following security requirements:

- *Accessing the IMS embedded object:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the IMS functionality, i.e. by accessing the IMS embedded object as specified in Section 7.8. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in Section 7.8.

10.1.4.6 Security requirements for metadata processing functionality

The metadata processing functionality (i.e. the `application/oipfSearchManager` embedded object and other APIs) as defined in Section 7.12 and 7.13.3 SHALL adhere to the following security requirements:

- *Accessing the search manager:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the search manager, i.e. by accessing the `application/oipfSearchManager` embedded object as specified in Section 7.12.1. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the properties or methods on the `SearchManager` embedded object.

- *Accessing enhanced metadata:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to access the extensions to `video/broadcast` for accessing EIT p/f information specified in section 7.13.3, in order to prevent misuse of the EIT p/f information If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access to the `programmes` property of the `video/broadcast` object specified in Section 7.13.3.

10.1.4.7 Security requirements for configuration and settings functionality

The configuration and settings functionality (i.e. the `application/oipfConfiguration` embedded object and other APIs) as defined in Section 7.3 SHALL adhere to the following security requirements:

- *Reading and modifying configuration and/or settings:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the configuration functionality, i.e. by accessing the configuration embedded object as specified in Section 7.3.1. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in section 7.3.

10.1.4.8 Security requirements for APIs for OITFs under the control of a service provider

APIs for OITFs under the control of a service provider SHALL adhere to the following security requirements:

- *Accessing the extended tuner control APIs:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the extended tuner control APIs as specified in Section 7.13.7. If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in section 7.13.7.

- *Accessing the extended PVR APIs:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the extended PVR APIs as specified in Section 7.10.4. If the server does not have the necessary privileges or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in section 7.10.4.

- *Accessing the download manager:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the download manager, i.e. by accessing the `application/oipfDownloadManager` embedded object as specified in Section 7.4.3. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods specified in Section 7.4.3.

- *Accessing all downloads:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to manage downloads not initiated by the current application, i.e. by accessing the downloads property of the `application/oipfDownloadManager` embedded object as specified in Section 7.4.3. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access this property.

10.1.4.9 Security requirements for remote diagnostics and management API

The remote diagnostics and management API (i.e. `application/oipfRemoteManagement`) as defined in Section 7.11.1) SHALL adhere to the following security requirements:

- *Accessing remote diagnostics and management parameters and/or settings:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the remote diagnostics and management functionality, i.e. by accessing the `application/oipfRemoteManagement` embedded object as specified in Section 7.11.1. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in section 7.11.1.

10.1.4.10 Security requirements for parental control manager

The parental control manager API (i.e. `application/oipfParentalControlManager`) as defined in Section 7.9.1) SHALL adhere to the following security requirements:

- *Accessing parental control manager functionality:* the OITF SHALL perform a security check (as defined in Section 10.1.1) to see if the server has the necessary privileges to interact with the parental control manager functionality, i.e. by accessing the `application/oipfParentalControlmanager` embedded object as specified in Section 7.9.1. If the server does not have the necessary privileges, or the server cannot be properly authenticated, the OITF SHALL throw an error as defined in section 10.1.1 when an application loaded from that server attempts to access any of the classes, properties or methods defined in section 7.9.1.

10.1.5 Permission names

This section describes a non-limited set of permission names that MAY be included as part of the "permissions" extension of a X.509v3 certificate as defined in Sections 10.1.1 and 10.1.2:

- "permission_tuner control_lineup": this permission name allows a server to receive/fetch the tuner's channel line-up and to switch an OITF's local tuner to another channel and to functionality as specified in Section 7.13.
- "permission_tuner_lineup" : this permission name allows a server to receive/fetch the tuner's channel line-up as specified in Section 7.13.
- "permission_tuner_control" : this permission name allows a server to switch an OITF's local tuner to another channel as specified in Section 7.13.
- "permission_recording" : this permission name allows a server to receive/fetch the tuner's channel line-up, and to instantiate the scheduler object (as defined by Section 7.10.1) and access its functionality, and to access the additional functionality as specified in Section 7.13.2 for the video/broadcast object to record and timeshift the current broadcast.
- "permission_download" : this permission name allows a server to initiate downloads.
- "permission_drmagent" : this permission name allows a server to interact with the DRM agent, i.e. by accessing the DRM agent embedded object as specified in Section 7.6.1
- "permission_metadata" : this permission name allows a server to access the DVB EIT p/f information of the current channel through the "programmes" property of the video/broadcast object, as specified in Section. 7.13.3.
- "permission_metadata_search" : this permission name allows a server to access the search functionality provided client-side metadata search functionality (as defined in section 7.12.1).
- "permission_extendedAV" : this permission name allows a server to interact with the extended A/V control functionality provided by the OITF, as defined in section 7.13.7.
- "permission_recordingsmanager" : this permission name allows a server to interact with the recording scheduler on the OITF using the APIs defined in section 7.4.3 to manage recordings initiated by the current application.
- "permission_recordingsmanager_all" : this permission name allows a server to interact with the recording scheduler on the OITF using the APIs defined in section 7.4.3 to manage all recordings, including those initiated by other applications.

- “permission_recordingsmanager_samedomain” : this permission name allows a server to interact with the recording scheduler on the OITF using the APIs defined in section 7.4.3 and manage recordings initiated by applications from the same FQDN.
- “permission_clientCOD” : this permission name allows a server to interact with the CoD catalogue browsing functionality provided by the OITF, as defined in section 7.12.
- “permission_settings” : this permission name allows a server to modify user settings and configuration using the APIs defined in section 7.3.1.
- “permission_downloadmanager” : this permission name allows a server to interact with the download manager on the OITF using the APIs defined in section 7.4.3 to control downloads initiated by the current application.
- “permission_downloadmanager_all” : this permission name allows a server to interact with the download manager on the OITF using the APIs defined in section 7.4.3 and manage all downloads, including those initiated by other applications.
- “permission_downloadmanager_samedomain” : this permission name allows a server to interact with the download manager on the OITF using the APIs defined in section 7.4.3 and manage downloads initiated by applications from the same FQDN.
- “permission ims”: this permission name allows a server to interact with an IMS Gateway using the APIs defined in section 7.8.
- “permission_remotemanagement”: this permission name allows a server to interact with an remote diagnostics and management API defined in section 7.11.
- “permission_gatewayinfo” : this permission name allows a server to interact with the gateway discovery functionality provided by the client, as defined in sections 4.2 and 7.7.
- “permission_parentalcontrolmanager” : this permission name allows a server to interact with the parental control manager on the OITF using the APIs defined in section 7.9 to override the parental control settings of an OITF.
- “permission_widget” : this permission name allows a server to interact with installed Widgets using the Widget.APIs defined in section 9.3.20.
- “permission_wakeup” : this permission name allows a server to setup wake-up requests using the APIs defined in section 7.3.3.
- “permission_set_power” : this permission name allows a server to set the power state to ON or ACTIVE_STANDBY using the setPowerState() method defined in section 7.3.3.

10.2 User Authentication

The OITF SHALL adhere to the user authentication requirements as specified in Section 5 of [OIPF_CSP2].

10.3 DLNA RUI Remote Control

The communication from the remote control device (DLNA RUIC) is secured by establishing a secure connection using SSL or TLS (i.e. HTTPS) if a `<security>` element in a DLNA RUIC Capability Description indicates that the Remote UI Client supports setting up a secure connection with the Remote UI Server (see Section 5.2.1 of [CEA-2014-A] for more information). It is the responsibility of the DAE application to require the DLNA RUIC to verify the user behind the remote control is actually the intended user. For example, this may be established by requiring a PIN number to be entered. It is outside the scope of this specification what measures are taken by the DAE application to ensure correct identification of the user.

11 DAE Widgets

DAE Widgets are a specialization of standard DAE applications. DAE Widgets are a profile of W3C Widgets. A mandatory requirement in the referenced W3C Widgets 1.0 specifications remains mandatory also for DAE Widgets and recommended and optional requirements in W3C Widgets 1.0 remain recommended and optional for the DAE Widgets, unless explicitly specified differently inside this document.

11.1 Widgets Packaging and Configuration

A Widget SHALL be packaged in order to allow a single download and installation on an OITF. The packaging format for the files of a Widget is defined in Section 5 of [Widgets-Packaging]. Content inside the Widget package has to be organized according to Section 6 of [Widgets-Packaging].

Each Widget package SHALL contain a configuration document as defined in Section 7 of [Widgets-Packaging]. All the attributes of the `<widget>` element are supported with the following exceptions and clarifications:

a) This specification does not mandate support for any view mode (as defined in 7.6.1 of [Widgets-Packaging])

b) "id" is mandatory for a DAE Widget. If this attribute is present in the manifest then the OITF SHALL use it. Otherwise the OITF should generate it internally and assign to the Widget.

Widgets also support Internationalization and Localization as defined in Section 8 of [Widgets-Packaging].

The steps for processing a Widget package and associated processing rules are described in Section 9 of [Widgets-Packaging].

11.2 Access Request

A Widget running on a OITF can request access to potentially sensitive APIs or resources. In order to avoid data leaking a security model for Widgets is imposed. DAE Widgets SHALL run in a "Widget execution scope", defined in section 2 of [Widgets-Access] as "the scope (or set of scopes, seen as a single one for simplicity's sake) being the execution context for code running from documents that are part of the Widget package". Note that Section 3 of the same specification states that "A user agent must prevent the Widget execution scope from retrieving network resources, using any method (API, linking, etc.) and for any operation, unless the user agent has granted access to an explicitly declared access request."

DAE Widgets SHALL also support mechanisms to define network permissions as defined in Section 3 and 4 of [Widgets-Access].

Note that according to [Widgets-Access] an OITF "may grant access to certain URI schemes without the need of an access request if its security policy considers those schemes benign". Furthermore a OITF "may deny access requests made via the access element (e.g. based on a security policy, user prompting, etc.)".

11.3 Widget Interface

A set of application programming interfaces (APIs) and events are defined for Widgets that enable baseline functionality such as exposing Widget metadata and runtime information.

The Widget interface primarily provides access to metadata derived from processing the Widget's configuration document. DAE Widgets SHALL support the Widgets interface as defined in Section 5 of [Widgets-APIs]. This specification doesn't define any scheme handlers for the `openURL()` method.

The Widget interface makes use of the Storage interface defined in Section 4.1 of [Web-Storage]. As an extension of that specification, Protected Keys in a Storage Area as defined in Section 6.1 of [Widgets-APIs] are also allowed.

Note that as defined in Section 6 of [Widgets-APIs] an OITF SHOULD limit the total amount of space allowed for storage areas per Widget. Furthermore an OITF SHALL support key and values at least 4kB long.

11.4 Digital Signature

Widget authors and distributors SHALL digitally sign Widgets as a mechanism to ensure continuity of authorship and distributorship. Prior to instantiation, an OITF SHOULD use the digital signature to verify the integrity of the Widget package and to confirm the signing key(s).

The process of digitally signing a W3C Widget is defined in [Widgets-DigSig].

Note that as defined in Section 7.3 of [Widgets-DigSig] in case of signature validation failure the user SHALL be notified; means or format of a failure notification are left up to implementers. The OITF MAY ask the user if the Widget should be installed even if the validation failed or if the signature is missing. If the user accept launching the Widget, it SHALL be run without access to privileged API.

Annex A. Void

Annex B. CE-HTML Profiling

This section defines a detailed set of deviations from the CEA-2014-A i-Box and 2-Box model [CEA-2014-A], in particular for those changes that are directly related to requirements in sections 5.1 through 5.10 and Annexes A through I of [CEA-2014-A]. Changes to requirements of CEA-2014-A are indicated by underlined text for text that must be added, and by strikethrough text for text that must be removed.

B.1 Changes to Section 5.2

Several new elements and new attribute/values have been added for the capability descriptions. Most of these are related to new functionality, and are defined in Section 9.3 and hence are not listed here. With respect to existing elements and attributes, the following changes apply:

- an additional value “0.33x0.33” for attribute “scaling” of the <video_profile> element in bullet w) of [Req. 5.2.1.a], with the following related extension to the schema for type “scalingType”

```
<xs:enumeration value="0.33x0.33"/>
```

- the “name”-attribute of the <audio_profile> and <video_profile> elements in CEA-2014-A are restricted to DLNA media format profiles. The forum has specified its own audio and video format profile names that can be used by the “name” attribute as well.

- new UI profiles have been defined for [Req. 5.2.1.b] that a client may choose to implement. Details are not included in this annex.

- for both <video_profile>, and <audio_profile> elements, it is allowed to include multiple profile names corresponding to the same MIME type, by separating each profile name with a whitespace character.

- element <pointer> requires some clarifications:

  m) <pointer> - indicates whether or not the Remote UI Client supports pointer-based input, such as mouse or touch. If included, the value of this element SHALL be: (true|false). A value of ‘true’ means that all mouse event types as defined in DOM level 2 Events SHALL be supported, and that server-side image maps SHALL be fully supported as defined in Section 13.6.2 of [HTML401] . Note that a value of ‘false’ still implies that ‘click’ events SHALL be supported, as per Req 5.4.1.s below.

B.2 Changes to Section 5.3

- Req. 5.3.a (5) states that if the Content-Encoding header is used, it SHALL always have case-insensitive value “identity”, unless a client/server has explicitly indicated support for other content encodings by using an Accept-Encoding header. RFC 2616 (section 3.5) states that this content-coding is used only in the Accept-Encoding header, and SHOULD NOT be used in the Content-Encoding header. We follow RFC 2616 and use the following alternative definition for Req. 5.3.a: “if this header is used, it SHALL always have a value that matches one of the content encodings as sent by an Accept-Encoding header, and SHALL adhere to Section 3.5 of RFC 2616 regarding the use of “identity” encoding”.

- Req 5.3.a (12) which states the requirements for the User-Agent header is replaced by the description in Section 8.1.

- Req 5.3.i, which limits the generated HTTP header line size to 998 bytes SHALL NOT be supported.

B.3 Changes to Section 5.4

- Since the CSS3 “image-orientation” property was defined in CSS Print/Paged Media, browsers may have difficulty implementing it for normal web pages. It is therefore made OPTIONAL. Services needing image rotation SHOULD do this at the server before sending it to the client.

- The W3C CSS working group made an official statement that the following DOM2 level Style features are considered to be problematic and have therefore been classified as obsolete.
  - The UnknownRule interface (unknown rules should be dropped by the parser and thus never reach the DOM).
  - The getPropertyCSSValue method, CSSValue interface, all interfaces inheriting from CSSValue, and the RGBColor, Rect, and Counter interfaces (the CSSValue interface is thought to be too awkward for frequent

use).

These features are OPTIONAL.

- In addition, the DocumentCSS and DOMImplementationCSS interfaces of DOM level 2 Style are also OPTIONAL.
- Compatibility with CEA-2027-A is not a requirement for the present document. Therefore, a client MAY omit the list of methods of the Window scripting object as listed by bullet 3) and the alias as defined by bullet 5) of requirement [Req. 5.4.2.a] of CEA-2014-A.
- The following methods SHALL be added to the list of supported properties and methods on the Window scripting object:
  - **Window self:** reference to the current window (returns same value as property "window").
  - **Window window:** circular reference to window object (returns same value as property "self").
  - **Number setInterval(Function f, Number d) –** call function f again after d number of seconds.
  - **void clearInterval(Number i) –** cancels the given interval timeout that has been set using "setInterval".
  - **void addEventListener(String t, EventListener l, Boolean capture) –** allows DOM 2 event listener registration on the Window object.
  - **void removeEventListener(String t, EventListener l, Boolean capture)** – allows DOM 2 event listener de-registration on the Window object.
  - **void blur()** – removes focus from current window. Calling this method on the Window object of a DAE application SHALL not deactivate the application.
  - **void postMessage(String message, String targetOrigin) –** used for cross-document messaging as defined by bullet 10 below.
  - **OipfObjectFactory oipfObjectFactory** – The global factory object which can be used to instantiate embedded object instead of using HTML <object> tags. See 7.1 for the definition of the **OipfObjectFactory** class.
  - **void close() –** closes the current window. Calling this method on the Window object of a DAE application SHALL be equivalent to calling method destroyApplication() of the DAE application (as defined in Section 7.2.2.2).
  - **Navigator navigator –** this property MAY return a Navigator object representing the OITF as defined in Section 7.15.4.
- A client MAY omit window.download() as defined in requirement [Req. 5.4.2.a] of CEA-2014-A. Applications SHOULD use registerDownloadURL as defined in Section 7.4.1 of this document.
- HTML5 cross-document messaging SHALL be supported as follows:

  10) Cross-document messaging, as defined in section 8.2 of [HTML5], a subset. The client SHALL support posting messages with the `postMessage` method as defined in Section 8.2.3 of [HTML5], prototype also listed below for reference. The `MessageEvent` interface defined in 8.1 of [HTML5] SHALL be supported, except for the `ports` value which MAY be undefined if the client does not support passing messages with ports.

     - void postMessage(any message, String targetOrigin)

- The HTML5 media elements SHALL be supported as follows::

  11) Sections 4.8.7, 4.8.8, 4.8.9 and 4.8.10 of [HTML5] SHALL be supported. Those sections cover the <source>, <audio> and <video> element, as well as the associated interfaces and processes. Only the XHTML syntax of said markup SHALL be supported. Support for this feature SHALL be indicated through the OITF's capability description by using element <html5_media> as defined in Section 9.3.16.

- Add keypress events to Requirement 5.4.1.a in the following way:

  [Req. 5.4.1.a] Every Remote UI Client SHALL support the DOM event types "keydown", "keypress" and "keyup" and the following subset of the KeyEvent interface as specified in [18], which SHALL inherit from the UIEvent interface:

  1) Properties:

     - readonly Boolean shiftKey;
     - readonly Number keyCode;

- readonly Number charCode;

2) Methods:

- initKeyEvent(DOMString eventType, Boolean canBubble, Boolean cancelable, Boolean ctrlKey, Boolean altKey, Boolean shiftKey, Boolean metaKey, Number keyCode, Number charCode), where:
- argument eventType is either “keydown”, “keypress” or “keyup”,

and

- arguments ctrlKey, altKey and metaKey MAY be ignored.

3) Constants:

- A subset of the VK_* constants as specified in Annex F, corresponding to the keys that are supported by the Remote UI Client (i.e. SHALL at least include the keys as specified by the client in the capability profile).

For “keydown” and “keyup” events, the key code as specified in Annex F that corresponds to the key that has been pressed SHALL be included in property keyCode.

For “keypress” events, if pressing a key (or sequence of keys) has resulted in generating a Unicode character, the resulting Unicode character code SHALL be included in property charCode. If no Unicode characters results from pressing the key (or sequence of keys), for example for the arrow keys, the key code as specified in Annex F SHALL be included in property keyCode.

~~Note: DOM “keypress” events are not supported.~~

- Add keypress events to Requirement 5.4.1.l:

  [Req. 5.4.1.l] A Remote UI Client SHALL generate one or more “keydown” and “keypress” events while a key is being pressed until the key is released, at a repetition rate determined by the client, and SHALL generate a “keyup” event as soon as the key is released.

  - Next to the “onkeydown” and “onkeyup” events, also add intrinsic event “onkeypress” to requirement [Req. 5.4.2.a] of CEA-2014-A:

    x) String onkeypress – read-write property that specifies the script to be called when a “keypress” event (as specified in Section 5.4.1) occurs on the window/frame that corresponds to this “window”-object.

  - Note: future revisions of CEA-2014-A or the DAE specification should consider the ability to specify a particular (maximum/minimum) size of textual or graphical labels to be inserted.

- Requirement 5.4.a.3.a SHALL be changed as follows;

  a) DOM level 2 Core [11], including the extended XML interfaces (except for Notation, Entity, EntityReference and Processing Instruction), i.e. method hasFeature(DOMString feature, DOMString version) of the DOMImplementation interface returns true for features “Core” and “XML”, and version “2.0”.

- Requirement 5.4.a.3.c SHALL be extended with the following;

  Focus events (i.e. events of type “focus”) SHALL be generated not only for <label>, <input>, <select>, <textarea>, and <button> as specified in Section 1.6.5 of [DOM 2 Events], but also at least for <a> elements, in accordance with [DOM 3 Events].

  For all elements which can receive focus events, a focus event SHALL be generated and the CSS “:focus” selector must be activated, irrespective if the focus is received through keyboard interaction, pointer interaction, calling an DOM focus() method through Javascript, or any other mechanism by which the focus can be changed.

- Requirement 5.4.a.3.d SHALL be changed as follows;

  d) DOM level 2 HTML [14] except following interfaces:

  • HTMLAppletElement,

  • HTMLFrameElement,

• HTMLFrameSetElement

The method hasFeature(DOMString feature, DOMString version) of the DOMImplementation interface returns true for features “HTML” and “XHTML”, and version “2.0”.

- Requirement 5.4.a.3.e SHALL be replaced as follows;

~~e) To distinguish between the subset as defined here for CE-HTML and full support for the DOM level 2 HTML module, the following applies:~~

~~• **hasFeature**(“CE-HTML”, “1.0”) SHALL return true if the subset of the DOM 2 HTML module is supported as defined above.~~

~~• **hasFeature**(“HTML”, “2.0”) and hasFeature("XHTML", "2.0") SHALL return true if the full DOM Level 2 HTML module is supported.~~

e) DOM level 2 Views [DOM 2 Views] with the method hasFeature(DOMString feature, DOMString version) of the DOMImplementation interface returning true for feature “Views” and version “2.0”.

f) The method hasFeature(DOMString feature, DOMString version) of the DOMImplementation interface SHALL return true for feature “CE-HTML” and version “1.0”.

- Requirement 5.4.a.6.b SHALL be replaced as follows;

~~b) If both attributes are defined and not the same, then the value defined by attribute “id” SHALL take preference.~~

b) Application authors SHOULD define both “id” and “name” on <a>, <form>, <iframe>, <img> and <map> elements as described in Section C.8 of [XHTML 1.0].

- Requirement 5.4.a.7 shall be extended with the following;
  - **nav-up, nav-down, nav-left, nav-right** as defined in Section 10.2.2 of [CSS3 UI].
  - **outline** and **outline-*** as defined in [Req. 5.4.1.q]
  - **letter-spacing** and **word-spacing** CSS2.1 [28] properties.
  - **border-top-right-radius**, **border-bottom-right-radius**, **border-bottom-left-radius**, **border-top-left-radius** and **border-radius** as defined in [CSS3 BG] with the following restrictions:
    - Only solid style is guaranteed to work in correlation with border radius: When border radius is used, non-solid border style may be ignored by an implementation and solid style be used instead.
    - If two borders are connected with a rounded corner, and those two borders have different computed colors, the OITF MAY draw both borders with the computed color of one of those two borders.
    - An implement may not trim a background image attached to the container box (or other elements contained in the container) at the outside of the rounded border. Only the background color of the container is guaranteed to be clipped to the rounded border by an implementation.

    Note that a full implementation of the border radius properties as defined in [CSS3 BG] is compliant with the subset defined above.

    The following corresponding DOM style properties (properties of the CSS2Properties interface) shall also be supported: **borderTopRightRadius**, **borderBottomRightRadius**, **borderBottomLeftRadius**, **borderTopLeftRadius**, **borderRadius**.

- Requirement 5.4.1.f SHALL be changed as follows:

If the input-focus is on any forms element except a button, a Remote UI Client SHALL not generate any VK_UP, VK_DOWN, VK_LEFT, and VK_RIGHT key-events, except at those points in time that the focus is about to move away from the form element (e.g. if VK_LEFT is pressed while the cursor is placed at the beginning of a text-entry), to allow an author of a HTML document to override the default focus navigation.

  - The client SHOULD use the same physical keys for generating the VK_UP, VK_DOWN, VK_LEFT and VK_RIGHT key events that are used for a spatial navigation mechanism provided by the client. The same keys SHOULD also be used for spatial navigation specified through the CSS properties ‘nav-up’, ‘nav-down’, ‘nav-left’ and ‘nav-right’.
  - In accordance with this requirement, the focus navigation as defined through CSS properties ‘nav-up’, ‘nav-

down', 'nav-left' and 'nav-right' SHOULD only be active at those points in time when focus can be moved away from the form-element, to not interfere with the implementation specific handling of keys inside a form-element.

- Requirement 5.4.1.m SHALL be changed as follows:

  A Remote UI Client SHALL offer a means to set focus to the following elements in a HTML document by using key-based input: <a>, <area>, all form elements, <iframe>, and <object> elements of type "video" as defined in Section 5.7.

  - Upon receiving focus, the Remote UI Client SHALL generate both a DOM 2 "focus" and a "DOMFocusIn" event for <a>, <area>, and ~~both a DOM 2 "focus" and "DOMFocusIn" event~~ for all form elements~~, for any registered event listeners~~.
  - The Remote UI Client MAY not generate DOM 2 focus and DOMFocusIn events in the following two cases. For <iframe> elements, ~~and <object> elements of type "video"~~ the Remote UI Client SHALL call the event listener that has been specified through the onfocus attribute of the "window" object (see Section 5.4.2) that is associated with the iframe. For <object> elements of type "video", it SHALL call the event listener specified through the onfocus attribute of the A/V scripting object (Section 5.7). ~~The Remote UI Client MAY not generate a DOM 2 focus events in those cases.~~

- Add a requirement 5.4.1.p that reads as follows:

  [Req. 5.4.1.p] A Remote UI Server SHOULD use the CSS properties 'nav-up', 'nav-down', 'nav-left' and 'nav-right' to override the default spatial navigation as provided by the Remote UI client, instead of defining a spatial navigation mechanism in Javascript.

- Add a requirement 5.4.1.q that reads as follows:

  [Req. 5.4.1.q] If a Remote UI Server has specified the "outline-style" attribute to be unequal to "auto" (as defined in Section 8.3 of the CSS3 Basic User Interface Module), for an element that has input focus, the Remote UI Client SHALL not draw its own focus highlight around this item, but use the focus highlight style, color and width as defined by the values given to the "outline" and/or "outline-*" attributes.

- Add a requirement 5.4.1.r that reads as follows:

  [Req. 5.4.1.r] A Remote UI Client SHALL generate the focus events as specified by [Req. 5.4.1.m] and SHALL activate the CSS ":focus" selector, for any element which can receive focus events, irrespective if the focus is received through keyboard interaction, pointer interaction, calling an DOM focus() method through Javascript, or any other mechanism by which the focus can be changed.

- Add a requirement 5.4.1.s as an extension to 5.4.1.m and 5.4.1.n

  [Req. 5.4.1.s] A Remote UI Client SHALL offer a means to activate the following elements in a HTML document by using key-based input: <a>, <area> <button>, <input type="submit">, <input type="reset"> and <input type="button">, <input type="radio">, and <select>.

  The Remote UI Client SHOULD allow the same physical key that is used to generate a VK_ENTER key event to be used to activate these elements if these elements have input focus. If an access key has been defined the Remote UI Client SHALL allow the access key to be used to activate these element.

  Upon activation, the Remote UI Client SHALL generate both a DOM 2 "DOMActivate" and a "click" event for above listed elements

B.4 Changes to Section 5.6.2

Support for this section SHALL be optional for an OITF. Support for section 5.6.2 SHALL be indicated through the OITF's capability description by using element <pollingNotifications> as defined in Section 9.3.14.

- Extend requirement 5.6.2.a as follows

  [Req. 5.6.2.a] An i-Box Remote UI Client SHALL support polling-based 3rd-party notifications from an i-Box server.

  1) To manage the polling process for a particular notification, an i-Box Remote UI Client SHALL support the following method of the Window/UIContentFrame object:
     a) Boolean **subscribeToNotifications**(String url, String name, Number period, String type) where

- ***url*** is the complete URL of the HTTP GET request made by the Remote UI Client every *period* seconds; the domain of *url* SHALL equal the domain of the current document in the CE-HTML browser window, and use SSL or TLS security[24][9][10]; if it doesn't, this method has no effect and returns *false*. If *url* equals the URL of any existing notification subscription and the value of *period* is positive, the *name* and *period* of that notification subscription is updated.
- ***name*** is the user friendly name of the notification service.
- ***period*** is the polling period of this subscription in seconds. If the value of *period* equals 0, any existing notification subscription with exactly the same URL is cancelled, and the return value indicates the former existence of such a subscription. If the value of *period* is negative, no changes are made and the return value indicates whether a subscription to the given URL already exists. If the value of *period* is positive, *true* is returned only if the Remote UI Client subscribes, or updates an existing subscription.
- ***type*** is the highest priority event type that will be sent by the notification service, and SHALL be one of the event types listed in bullet 10 of [Req 5.6.1.a], without the "upnp:"-prefix.

b) On executing the **subscribeToNotifications** method to subscribe to a new notification, the Remote UI Client SHALL alert the user to the impending new notification subscription (including information about the highest priority notification type that will be sent by the Remote UI Server), and provide the user with at least two options:
- subscribe to this notification, and
- do not subscribe to this notification.

This does not exclude an option that allows a user to always accept notifications from the same URL.

c) If the Remote UI Client does not subscribe because the user declined, the **subscribeToNotifications** method SHALL return *false*.

2) To manage the polling process for a particular notification, an i-Box Remote UI Client SHALL support the following method of the Window/UIContentFrame object:

a) Number **subscribeToNotificationsAsync**(String url, String name, Number period, String type) where
- ***url*** is the complete URL of the HTTP GET request made by the Remote UI Client every *period* seconds. *url* SHALL have the same origin as the current document in the CE-HTML browser window, and use SSL or TLS security [24][9][10]; if it doesn't, this method has no effect and an event indicating a negative response is dispatched. If *url* equals the URL of any existing notification subscription and the value of *period* is positive, the *name* and *period* of that notification subscription is updated.
- ***name*** is the user friendly name of the notification service.
- ***period*** is the polling period of this subscription in seconds. The value of period SHALL be greater than zero.
- ***type*** is the highest priority event type that will be sent by the notification service, and SHALL be one of the event types listed in bullet 9 of [Req 5.6.1.a], without the "upnp:"-prefix.
- The return value of his method indicated the ID of the subscription request. This is used when notifying the application of the result of this call, to link a response to the request that generated it.

b) On executing the **subscribeToNotificationsAsync** method to subscribe to a new notification, the Remote UI Client SHALL asynchronously alert the user to the impending new notification subscription (including information about the highest priority notification type that will be sent by the Remote UI Server), and provide the user with at least two options:
- subscribe to this notification, and
- do not subscribe to this notification.

This does not exclude an option that allows a user to always accept notifications from the same URL.

Calls to **subscribeToNotificationsAsync** return immediately. The application will be notified via the **onNotificationSubscriptionResponse** function (or corresponding DOM-2 event) user has chosen to subscribe or to not subscribe to the notification.

If two calls to **subscribeToNotificationsAsync** with the same value for *url* overlap (i.e. the notification event of the first call has not yet been dispatched), the Remote UI Client SHALL interrupt the first call and generate a response event as if the request had been declined.

3) An i-Box Remote UI Client SHALL support the following property of the Window/UIContentFrame object:

c) script **onNotificationSubscriptionResponse**

where the specified function is called with arguments id and response, which are defined as follows:

- Number **id** – the ID of the subscription request, as indicated by the return value of the **subscribeToNotificationsAsync** method.
- Boolean **response** – the response indicating whether the subscription request has been accepted. A value of *false* indicates that the request has been declined. A value of *true* indicates that the request has been accepted.

4) An i-Box Remote UI Client SHALL support the following method of the Window/UIContentFrame object:

d) void **unsubscribe**(string url, string name)

where

- ***url*** is the URL used to subscribe to a notification, which SHALL have the same origin as the current document in the CE-HTML browser window
- ***name*** is the user friendly name of the notification service.

e) On executing the unsubscribe method, the Remote UI Client SHALL unsubscribe from the specified notification service. If the application is not subscribed to the specified notification service or if the page currently loaded in the CE-HTML browser window is not from the same origin as url, this method SHALL have no effect. When this method returns, the application shall no longer be subscribed to the notification service.

5) An i-Box Remote UI Client SHALL support the following method of the Window/UIContentFrame object:

f) StringCollection **listNotificationSubscriptions**()

where the return value of this method SHALL be a collection of URLs of notification services to which HTML documents from the same origin are currently subscribed.

6) An i-Box Remote UI Client SHALL support the following method of the Window/UIContentFrame object:

g) Boolean **isSubscribed**(string url, string name)

where

- ***url*** is the URL used to subscribe to a notification, which SHALL have the same origin as the current document in the CE-HTML browser window
- ***name*** is the user friendly name of the notification service.
- The return value of this method SHALL be *true* if *url* has the same origin as the current application and application is currently subscribed to the specified notification service, or *false* otherwise.

B.5 Changes to Section 5.7

In addition to the A/V Control object extensions in Section 7.14, the following detailed modifications to Requirement 5.7.1.f SHALL apply.

- Requirement 5.7.1.f SHALL be modified as follows;

  [Req. 5.7.1.f] The following properties and methods SHALL be supported for audio objects and for video objects. Support for playlists and support for the "persist" attribute is OPTIONAL.

- Requirement 5.7.1.f bullet 1) 'data' SHALL be modified as follows;

  2) String data [RW] – media URL. If the value of data is changed while media is playing playback is stopped (resulting in a play state change). The default value is the empty string. If the value of this attribute is changed, the related data-attribute inside the DOM tree SHOULD be changed accordingly. If the value of this attribute is set to an empty string or is changed, the resources (files, server connections, etc…) currently owned by the object SHALL be released.

- Requirement 5.7.1.f bullet 2) 'playPosition' SHALL be modified as follows;

  3) Number **playPosition** [R] - the play position in number of milliseconds since the beginning as denoted by the server (i.e. in relation to NPT 0.0 as described in Section 3.6 of RFC 2326) of the media referenced by attribute data when data refers to a single media item. **playPosition** is the duration of the currently playing media item of a playlist if data refers to a playlist. The behaviour of the A/V Control object when the end of media (or the end of the currently-available media) is reached is defined in section 7.14.1 of the DAE specification. If the play position cannot be determined, the **playPosition** SHALL be `undefined`.

- Requirement 5.7.1.f bullet 3) 'playTime' SHALL be modified as follows;

3) Number **playTime** [R] - the estimated total duration in milliseconds of the media referenced by *data* when *data* refers to a single media item. **playTime** is the duration of the currently playing media item of a playlist if *data* refers to a playlist. If the duration of the media cannot be determined, the **playTime** SHALL be undefined.

- Requirement 5.7.1.f bullet 4) 'playState' SHOULD be clarified as follows to fit the state diagram as specified in Section 7.14.1;

4) Number **playState** [R] - indication of the current play state as follows:

0 - *stopped*; user (or script) has stopped playback of the current media, or playback has not yet started.

1 - *playing*; the current media pointed to by *data* is currently playing.

2 - *paused*; the current media pointed to by *data* has been paused.

3 - *connecting;* connect to media server, i.e. waiting for connection to media server to be established, upon first connection or after the connection was lost. In addition, DRM rights necessary for playback of protected content are also retrieved during this state.

4 - *buffering*; ~~the media is being buffered before playback.~~ the buffer is being filled in order to have sufficient data available to initiate or continue playback. In this state, playback is stalled due to insufficient data in the buffer to continue playback. The player waits until sufficient data has been buffered to continue playback. For video objects, whilst being in this state, the player SHOULD show the last completed video frame that was shown before entering this state. This playstate is an intermediate state to reach playState 1 ('playing'). The OITF SHOULD buffer the content in the background whilst in playState 2 ('paused'). However, this background buffering does not result into a state change to state 4.

5 - *finished;* the playback of the current media has reached the end of the media.

6 - *error*; an error occurred during media playback, preventing the current media to start/continue playing.

- Requirement 5.7.1.f bullet 5) 'error' SHALL be modified as follows;

5) Number **error** [R] - error details; only significant if the value of *playState* equals 6:

0 - A/V format not supported.

1 - cannot connect to server or connection lost.

2 - unidentified error.

3 - insufficient resources.

4 - content corrupt or invalid.

5 - content not available.

6 - content not available at given position.

- Requirement 5.7.1.f bullet 11) 'play' SHALL be modified as follows;

11) Boolean **play(Number speed)** - plays the media referenced by *data*, starting at the current play position denoted by *playPosition*, at a relative speed equal to the value of attribute *speed*. Negative speeds reverse playback. If no speed is specified, it defaults to 1. A *speed* of 0 will pause playback. If the current media can be played at the specified speed, *true* is returned. Otherwise, *false* is returned and neither the *play state* nor the *speed* is ~~not~~ changed. If the playback reached the beginning of the media at rewind playback speed, then the play state is changed to 2 ('paused'). A play speed event (see section 7.14.4.2 of the DAE specification) will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play speed.

- Requirement 5.7.1.f bullet 13) 'seek' SHALL be modified as follows;

13) Boolean seek(Number pos) – if seek() is called while the player is in state 1 ("playing", then it sets the current play position (in milliseconds) to the value of pos and MAY change play state to 4 ('buffering'). If the player

is in state 2 ('paused') then the seek() method seeks to the new position, but the play state and the rendered image is not changed. . If the new playback position is valid, the value of the playPosition attribute SHALL be set to the new value before this method returns. ~~Does not affect the play state~~. Returns *true* if the method succeeded, and *false* otherwise. A play position event (see section 7.14.4.2 of the DAE specification) will be generated when the operation has completed, regardless of the success of the operation. If the operation fails, the argument of the event SHALL be set to the previous play position.

- Requirement 5.7.1.i SHALL be modified as follows;

  [Req. 5.7.1.i] If a video object has input focus:

  - The Remote UI Client SHALL at least generate DOM Level 2 key events (as specified in Section 5.4.1) for the navigation keys: VK_UP, VK_DOWN, VK_LEFT and VK_RIGHT; for the VK_OK key; and for the transport keys: VK_PLAY, VK_PAUSE, VK_PLAY_PAUSE, VK_STOP, VK_PREV, VK_NEXT, VK_FAST_FWD and VK_REWIND.
  - The means required by [Req. 5.2.2.h] to go back to the previous UI state (e.g. by pressing a "back"-button), SHALL be included as a means to go from full-screen to windowed mode. Furthermore, note that [Req. 5.4.1.m], [Req. 5.4.1.n] and [Req. 5.4.1.o] apply w.r.t. handling focus, in order to navigate focus to and away from a video object (in addition to the use of scripting methods).
  - The Remote UI Client SHALL not block execution of scripts of the CE-HTML page from which the focus was moved to the video object, even when the video is playing full-screen and has input focus. However, the Remote UI Client MAY block execution of scripts if the CE-HTML page was explicitly closed by the user.
  - The Remote UI Client SHALL have some appropriate keys to control video playback, which SHALL at least include a key to start and stop playing.
  - ~~The Remote UI Client MAY use the VK_OK key and/or the transport keys VK_PLAY, VK_PAUSE, VK_STOP, VK_NEXT, and VK_PREV for controlling video playback. However, the Remote UI Client SHOULD not use the keys listed under the first bullet of [Req. 5.7.1.i], and SHOULD not generate DOM Level 2 key events for those keys that the client uses for A/V control, to prevent Javascript to define possibly conflicting actions.~~
  - The Remote UI Client SHALL NOT handle the VK_OK, VK_PLAY, VK_PAUSE, VK_PLAY_PAUSE, VK_STOP, VK_FAST_FWD, VK_REWIND, VK_NEXT or VK_PREV keys and no action shall be taken by the Remote UI Client when these keys have been requested by an application.

B.6 Changes to the Annexes

- In Annex C, the default value for the transport attribute of the audioProfileType and videoProfileType and for the "protocolNames" attribute of the downloadType is defined as "http". In Annex F.1 of [OIPF_PROT2] the equivalent protocol name is called "http-get". OITFs and DAE applications SHALL consider the default to be "http-get".
- In Annex F, the following key code is defined for the remote control key that allows to toggle between PLAY and PAUSE states:

```
const Number VK_PLAY_PAUSE = 463;
```

- In Annex G, the "onkeypress" events in the abbreviation section in the introduction is currently marked with a dashed blue color. This marking must be removed.
  - The following clarifications apply to inline (i.e. intrinsic) event registration using the on* attributes in (X)HTML:
    - If value `event` is used inside the script inside the on* attribute, for example as an argument to one or more functions inside the on* attribute, the associated event is in scope for the evaluation by the script once the event occurs. For example, in the following snippet, the event is passed as a parameter to function callMe, and the default action to follow the link is prevented:

```
<a href="http://www.google.com" id="clickme" onclick="callMe(event);
event.preventDefault();">Click me</a>
```

- If the event registration inside the on* attribute returns `false`, the default action for activating the (X)HTML element is prevented from occuring. For example, in the following snippet, the function callMe is called, after which the default action to follow the link is prevented:

```
<a href=" http://www.google.com  onclick="callMe(); return false;">
```

- In Annex H, as per the change to Section 5.4, the “image-orientation” CSS property is not supported.
  - The following clarification applies for the “font” CSS property: “Support for system font values (caption, icon, menu, message-box, small-caption, status-bar) is not required.”
- In Annex I:, the “onkeypress” intrinsic event handler must be added to the “window” interface. And attribute “charCode” must be added to the “KeyEvent” interface.
  - The additional implementation note for EventListener does not apply, and method handleEvent must be supported as defined in DOM 2 Events.
  - The following clarification apply to DOM 2 Events handling:

    a) The “this” keyword inside the event handler always refers to the object on which the event handler was registered (i.e. the HTML element that is currently handling the event). For example the following snippet

```
my_element.addEventListener('click',doSomething,false);
function doSomething() {
    this.style.backgroundColor = '#cc0000';
}
```

    will cause the element “my_element” to get a red background whenever the user clicks on it.

  - Full support for “DOM Level 2 HTML” specification is added except for the following interfaces: “HTMLAppletElement”, “HTMLFrameElement” and “HTMLFrameSetElement”.
  - Full support for “DOM Level 2 View” specification is added by supporting the “DocumentView” interface (implemented by the document object) and its “defaultView” attribute.

Annex C. Design Rationale (Informative)

C.1 The application model

As specified in section 4.3.2, applications are recorded within a hierarchy of applications. This hierarchy has a number of benefits for an environment where multiple applications may be executing simultaneously, including:

- Clear separation of applications so that permissions granted to one application cannot be exploited by another.
- Simpler event dispatch, whether for key events or externally triggered events such as parental control changes, caller ID integration, IM chat messaging, etc.
- The ability to deploy new applications without affecting other applications (either UI or structure).
- The ability for service providers to manage groups of applications, including invisible applications.

Each object representing an application possesses an interface that provides access to methods and attributes that are uniquely available to applications. For example, the facilities to create and destroy applications are accessed through such methods.

Development and maintenance efficiencies are gained through distinct application boundaries. Code reuse is offered through the application tree, permitting applications to export facilities as desired (for example, channel change logic may be embedded in the “zapper” application and exported to an EPG application). The paired advantages of compartmentalisation and code re-use are of increasing value as the number of authoring entities and applications grows – what is of marginal additional value for one authoring entity and three applications is of significant value for 10 authoring entities and 50 applications.

Annex D. Clarification of Download CoD, streaming CoD and CSP interfaces (Informative)

D.1 Introduction

There are many different usage models and scenarios that one can think of when dealing with protected content and the interactions the user or the device may have with a service provider. This includes usage models regarding user registration, domain management, license acquisition, downloading content, etc. This informative Annex aims to clarify the usage of the interfaces as specified in Sections 4.6, 4.7, 7.4 and 7.6. in the context of these interactions. However, this Annex will only show some of the generic mechanisms as offered by these interfaces, not only the browser interfaces, but also including some of the local interfaces on the device (that actually do not need to be standardized) In the figure below these are indicated by dotted lines.

The main scenario that we envision is the following:

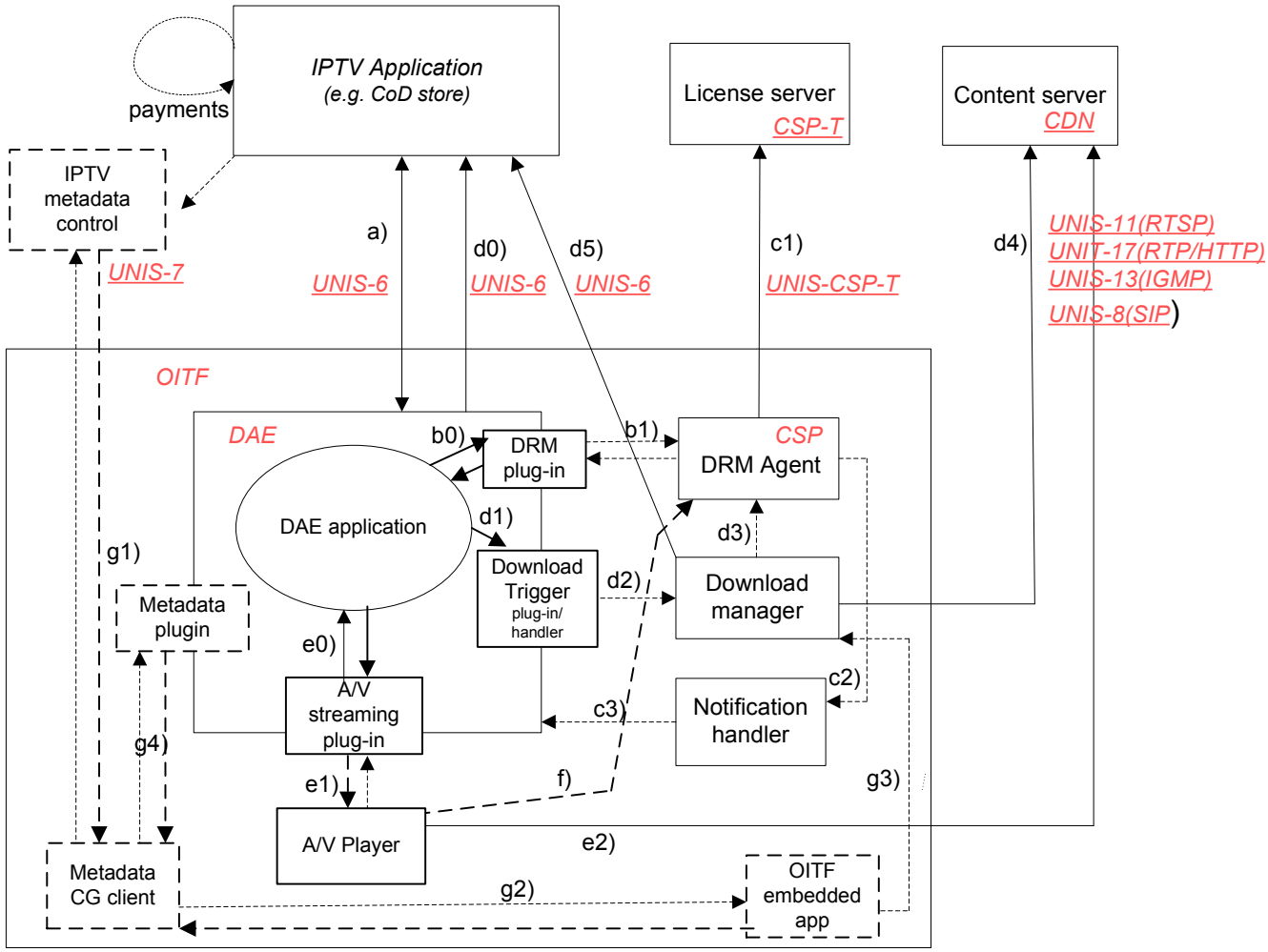


Figure 18: Main scenario

The OITF shows the UI of the CoD store. With this UI the user is able to interact with the CoD store to do things, such as user registration, browsing the content offered by the CoD store, and purchase a license.

This can be done inside the browser using a standard CE-HTML interface. In the figure above, this is identified by a).

In those deployments where the OITF supports the metadata CG client, an embedded application or a DAE application can make use of metadata provided through a metadata CG client. This is identified by g*).

After purchasing/selection of the content the selected content needs to be fetched. To this end, the download manager or the A/V embedded object needs to be triggered with information on how to fetch the content. This is done by using a special descriptor, with an easily identifiable MIME type "`application/vnd.oipf.ContentAccessDownload+xml`" in case of download, and "`application/vnd.oipf.ContentAccessStreaming+xml`" in case of streaming. This is indicated by interfaces d0, d1, d2, e0, e1), and e2).

For certain steps in these interactions, the CoD store may need to interact with the DRM agent. This can be done by talking directly to the DRM agent during a browser session using interfaces b0) and b1). Alternatively, the <DRMControlInformation> element of the content access descriptor can be used to convey DRM specific messages to the DRM agent. This is indicated by interface d3).

Note that both the DRM agent and Download manager are autonomous components that will be actively performing their duties, irrespective whether there is an active browser session or not. They will have their own interaction with e.g. the license server and download server, and possibly with the user. These interactions are identified by interfaces c1, c2, d4, d5.

The download manager or A/V player fetch the content, as indicated by interfaces d4 and e3.

Once the content is fetched, playback can be started in the A/V player. When the stream is protected, the A/V player will have to get a license from the DRM agent using interface f).

D.2 List of interfaces

Interface a: browse, select and purchase content from CoD store

This interface is used to interact with the CoD store for operations such as user registration, browsing the content offered by the CoD store, and purchase a license. This is a standard CE-HTML/HTTP interface.

Interface b*: In-session interaction from web page with underlying DRM agent

Interface b1 (and the related interface b2) is the application/oipfDrmAgent Javascript embedded object interface as defined in Section 7.3. This interface will allow messages to be exchanged between pages from the CoD store and the underlying DRM agent, whilst the user is having a user interface session with the CoD store. Examples of these messages are Marlin Action tokens. This is useful to enable scenarios, such as subscription license acquisition, registration, domain management, etc.

The interface basically consists of one method: `sendDRMMessage(String msgType, String msg)`, which is very generic in the sense that any kind of message can be exchanged. The exact payload and types of messages that could be exchanged is defined in the [OIPF_CSP2]. An example of such message could be:

```
pluginElement = document.getElementById("drmplugin");
pluginElement.sendDRMMessage("application/vnd.marlin.drm.actiontoken+xml",
                             "<marlin>…</marlin>");
…
<object id="drmplugin" type="application/oipfDrmAgent"/>
```

Note that this API is designed to be asynchronous in nature, because certain interactions may take a undeterminate amount of time. Therefore, it is not wise to make the method synchronous, since that could block the Javascript engine. To this end we have defined an event handler: `onDRMMessageResult`, to register a callback function that will be called when the DRM agent completed handling of the message. For example:

```
function callbackF(String msgID, String resultMsg) {
        …
}
document.getElementById("drmplugin").onDRMMessageResult = callbackF;
```

An equivalent DOM2 event is also generated.

Content authors SHOULD be aware of the asynchronous nature of the API. Only after having received the callback message, the web page can assume that the DRM agent has handled the DRM message. The service author may need to define some visual cues to the user if he would like the user to wait for certain actions to finish.

Interface c*: Autonomous out-of-session interaction between DRM agent and CoD store

Interface c1) is the collection of interfaces between the DRM agent, the CoD store, the license server, etc. as defined in the [OIPF_CSP2]. The interaction is typically done outside the scope of the browser, and also without the user

being involved. In the few cases where the user would be involved, the device will typically have its own “local” user interface to handle the interaction with the user. In some of these the DRM agent would need to open a web page to the originating CoD store, so that the user could resolve the issue directly with the store (e.g. using the rights URL extracted from the MPEG2_TS). Since the user could be doing other things at that moment, it may not be appropriate to popup/replace the current browser session without the user consent. Therefore, the DRM agent could issue a notification event that will get listed along similar lines to a third-party notification event. The user would be notified that his attention is required with respect to the DRM agent, and can then decide to take action and launch the browser.

In the figure above, these UI interactions are identified by interface c2) and c3). These interfaces however are typically local inside the OITF, and are not specified in more detail.

Interface d*: Downloading content

These interfaces are used for downloading content. In order to trigger the download, a special content-access descriptor (the Content Access Download Descriptor) with an easily identifiable MIME type “`application/vnd.oipf.ContentAccessDownload+xml`” is used. This descriptor contains all the relevant data related to fetch the content. This content-access descriptor is typically provided by the CoD store. A browser application can fetch this descriptor in various different ways, e.g. by following a link or through an XMLHttpRequest. This is identified by interface d0. The Content Access Download Descriptor and MIME type are defined in Annex E. It contains elements, such as <ContentURL> which indicates where the content item can be fetched, and <MetadataURL> to indicate where additional metadata, such as genre, subtitles, artwork, etc. can be retrieved from.

Interface d1) (and related interface d2) are used to trigger/register the download with the download manager. This is done by handing over the Content Access Download Descriptor to the download manager by calling method `registerDownload()` on the `application/oipfDownloadTrigger` embedded object after retrieving the content-access descriptor e.g. through XMLHttpRequest. Once the download is registered, the download manager will take care that the content is downloaded. Since this may be a lengthy task, the download manager is an independent process from the browser, that will perform its duty in the background even if the browser is closed. By making the download manager an independent process of the browser, the user can in the meantime do other things.

Interface d3) is a local interface that is used to pass optional DRM messages carried in the content-access descriptor from the Download manager to the DRM agent. These messages are included as part of one or more <DRMControlInformation> element inside the Content Access Download Descriptor (as defined by Annex E). These may include messages (such as a Marlin preview license) in cases where license information and the content to be downloaded can be packaged together.

Interface d4) is the actual interface for downloading the content. The protocols that can be used for downloading content are defined in the Open ITPV Forum Protocols specification document. The default protocol is HTTP, with support for HTTP Range requests. The HTTP Range requests are used in order for downloads to be able to resume after e.g. network failure or device power-down, because as mentioned above, the download manager is an autonomous component that must continue downloading the requested content items as a background process, even after a device power-down or network failure, until it succeeds or the user has given permission to terminate the download.

Interface d5) defines an interface to enable error recovery for the download mechanism. It could be used to recover from errors or other situations that lead to the corruption or deletion of the content/licenses or a current download to fail. An example usage is as follows: to be able to refetch the content, and its licenses from the CoD store the OITF may synchronize with the CoD store by issuing a secure HTTP GET request to the URL of element <OriginSite> concatenated with “/synchronize” as defined by the content-access descriptor, after which the IPTV application offering the content-download replies with an XML document describing the list of zero or more content IDs that had previously been downloaded by the given user (i.e. it is assumed that the IPTV application offering the content download still remembers which content a user has bought and downloaded before), using for example the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
           attributeFormDefault="unqualified">
  <xs:element name="synchronizelist" type="SynchronizeType"/>
  <xs:complexType name="SynchronizeType">
    <xs:sequence>
       <xs:element name="content" type="ContentType" minOccurs="1" maxOccurs="1"/>
     </xs:sequence>
  </xs:complexType>
```

```
    <xs:complexType name="ContentType">
       <xs:sequence>
         <xs:element name="content_ID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
       </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Example:

```
<synchronizelist>
   <content>
      <content_ID>item 1</content_ID>
      <content_ID>item 2</content_ID>
      ...
   </content>
 </synchronizelist>
```

Note: To authenticate the user, cookies or single sign on may be used.

The OITF MAY use this information to decide which content and which licenses to refetch. Refetching the content is done by issuing a secure HTTP GET request to the following URL:

`<OriginSite> + "/synchronize" + "?" + a <content_ID> value`

after which the application offering the content download replies with the appropriate information to retrigger the download by providing the appropriate Content Access Download Descriptor in order to trigger the download manager and DRM agent to redownload the content and related licenses.

Interface d6): Although the download manager is an autonomous process, the user may sometimes want to view or control the state of the download manager. To this end, the download manager will typically offer its own user interface, which allows the user to manage the ongoing downloads (e.g. suspend/resume, cancel) and monitor the progress of the items that are being downloaded. This is interface d6) in the figure above. In non-managed network deployments this is typically a local user interface, for which no protocol needs to be defined. However, since it may be useful for the user to have a quick overview of the current downloads, in Section 7.15.1 of this document a visualization embedded object called `application/oipfStatusView` has been defined by which a (third-party) server provider could include an overview of the status of the download manager as part of its UI. NOTE: for managed network deployments Javascript interfaces may be needed to have more control over the UI of the download manager. This is covered by the download manager APIs in Section 7.4.3 of the DAE specification.

Interface e*: Unicast Streaming and playback of downloaded content using A/V Control object

The CEA-2014-A A/V control object may be used to render unicast streaming content triggered by a content-access streaming descriptor (as specified in Section 7.14.2) and may be used to play back (partially) downloaded content by using the method `setSource` as specified in Section 7.14.8.

Interface e0) can be used to pass for a content access streaming descriptor to set up a protected stream, by passing through interface e1 the necessary information for the A/V player to set up the stream through interface e2), and for passing included <DRMControlInformation> messages to the DRM agent for DRM protection of the streamed content using interface f).

Interface e0) can also be used to get feedback from the A/V player (such as DRM related playback errors as defined in Section 7.13.5) in case of playing streaming content or partially downloaded content (through method `setSource()`).

Interface f: Request license

The A/V Player will render the content. When the content is protected, the A/V embedded object will have to get the necessary keys from the DRM agent using interface f) in order to decrypt the content.

If the content is played inside the browser, interface e1) defines a callback event "`onDRMRightsError`" to allow the page to handle DRM-related errors (in addition to c1).

Interface g*: Local metadata based applications

These interfaces are for use with local OITF embedded and DAE applications that may wish to use a metadata CG client for browsing and selecting the content.

D.3 Additional notes about Content-on-Demand

For a detailed specification of how devices and users are authenticated, we refer to [OIPF_CSP2]. For the security model related to accessing the DRM agent and Download manager from an external source, such as a web page (i.e. to open up the browser's sandbox), we refer to Section 10.1.

Annex E. Content Access Descriptor Syntax and Semantics

E.1 Content Access Download Descriptor Format

An OITF that supports Content Download (i.e. if the <download> element has been given value "true" in the OITF's capability profile as specified in Section 9.3.4) SHALL support parsing and interpretation of a Content Access Download Descriptor with MIME type "application/vnd.oipf.ContentAccessDownload+xml".

A valid Content Access Download Descriptor SHALL adhere to the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="urn:oipf:iptv:ContentAccessDownloadDescriptor:2008"
           xmlns:xml="http://www.w3.org/XML/1998/namespace"
           targetNamespace="urn:oipf:iptv:ContentAccessDownloadDescriptor:2008"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
     <!-- schema filename is  iptv-ContentAccessDownloadDescriptor.xsd -->
     <!-- this schema redefines the generic Content Access Descriptor Schema iptv-
          AbstractContentAccessDescriptor.xsd as defined in Annex E.3 by limiting the allowable
          values for attribute "TransferType" to "playable_download" and "full_download" -->
     <xs:redefine schemaLocation="iptv-AbstractContentAccessDescriptor.xsd">
        <xs:simpleType name="TransferTypeEnum">
           <xs:restriction base="tns:TransferTypeEnum">
              <xs:enumeration value="full_download"/>
              <xs:enumeration value="playable_download"/>
           </xs:restriction>
        </xs:simpleType>
     </xs:redefine>
</xs:schema>
```

The semantics of the allowable values for attribute TransferType as defined by simple string type TransferTypeEnum is as follows:

a) Attribute "TransferType", which indicates the type of transfer used for the content, SHALL have one of the following values:

   i) "full_download", which indicates that the content-item must be fully downloaded and stored before playback.

   ii) "playable_download", which indicates that the content-item is available for playback whilst it is being downloaded and stored by the download manager. The term "playable_download" is used solely in the context of the download manager and relates to storing the content (on persistent storage), and playing the stored version, and does not relate to buffering in the context of HTTP streaming.

The syntax and semantics of the imported elements from the generic Content Access Descriptor Schema SHALL be as defined in Annex E.3.

NOTE: An OITF SHALL silently ignore unknown elements and attributes that are part of a Content Access Download Descriptor.

E.2 Content Access Streaming Descriptor Format

An OITF SHALL support parsing and interpretation of a Content Access Streaming Descriptor with MIME type "application/vnd.oipf.ContentAccessStreaming+xml".

A valid Content Access Streaming Descriptor SHALL adhere to the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="urn:oipf:iptv:ContentAccessStreamingDescriptor:2008"
           xmlns:xml="http://www.w3.org/XML/1998/namespace"
           targetNamespace="urn:oipf:iptv:ContentAccessStreamingDescriptor:2008"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
     <!-- schema filename is  iptv-ContentAccessStreamingDescriptor.xsd -->
     <!-- this schema redefines the generic Content Access Descriptor Schema iptv-
```

```
        AbstractContentAccessDescriptor.xsd as defined in Annex E.3 by limiting the allowable
        values for attribute "TransferType" to "streaming" -->
    <xs:redefine schemaLocation="iptv-AbstractContentAccessDescriptor.xsd">
       <xs:simpleType name="TransferTypeEnum">
          <xs:restriction base="tns:TransferTypeEnum">
             <xs:enumeration value="streaming"/>
          </xs:restriction>
       </xs:simpleType>
    </xs:redefine>
</xs:schema>
```

The semantics of the allowable values for attribute `TransferType` as defined by simple string type `TransferTypeEnum` is as follows:

a) Attribute "`TransferType`", which indicates the type of transfer used for the content, SHALL have one of the following values:

   i) "`streaming`", which indicates that the content-item is streamed and should not be stored. This TransferType value is required for unicast streaming using an A/V control object as defined in Section 7.14.2.

The syntax and semantics of the imported elements from the generic Content Access Descriptor Schema SHALL be as defined in Annex E.3.

NOTE: An OITF SHALL silently ignore unknown elements and attributes that are part of a Content Access Streaming descriptor.

E.3 Abstract Content Access Descriptor Format

This section specifies the generic (i.e. "abstract") content access descriptor XML Schema that forms the basis for the XML Schemas of document types: `application/vnd.oipf.ContentAccessDownload+xml` and `application/vnd.oipf.ContentAccessStreaming+xml`.

An Abstract Content Access Descriptor SHALL adhere to the semantics as defined in the bulleted list below. In this bulleted list, optional means optional for server, but mandatory to be supported on OITFs that have indicated support for MIME type "`application/vnd.oipf.ContentAccessDownload+xml`". Mandatory means mandatory for the server to include this element in the content access descriptor.

1) <***Contents***> - mandatory element which is a container for one or more associated <`ContentItem`> elements as child element.

2) <***ContentItem***> - mandatory element which indicates a content-item. All other elements listed below are child-elements of a <`ContentItem`> element.

3) <***Title***> - mandatory element which indicates a user interpretable name to describe the content item. In case of content download, it may serve as a basis/suggestion for the actual filename used for storing the downloaded content item. It is recommended for an OITF to not require the user to enter a filename and select the storage device for storing a downloaded content item.

4) <***Synopsis***> - optional element which indicates a user interpretable description of the content item.

5) <***OriginSite***> - mandatory element which indicates the URL of the site from which this content access description document can be downloaded. Typically this is the site from which the content is/can be purchased.

6) <***OriginSiteName***> - Optional element, which gives the friendly name describing the origin site.

7) <***ContentID***> - Optional element which gives a unique identification of the content item relative to the OriginSite.

8) <***ContentURL***> - mandatory element which indicates the URL from which the content can be fetched. The element has the following attributes:

a) Optional attribute "DRMSystemID", which indicates the DRM system for which this URL applies, using a value as defined by element DRMSystemID in Table 8 of Section 3.3.2 of [OIPF_META2]. For example, for Marlin, the DRMSystemID value is "urn:dvb:casystemid:19188". This attribute is used for linking a <ContentURL> to a corresponding <DRMControlInformation> element with the same DRMSystemID value. If the "DRMSystemID" attribute is not specified or has value empty string, then this indicates that the content is not DRM protected.

b) Attribute "TransferType", which indicates the type of transfer used for the content. The concrete values that are allowed for this attribute are defined in Annexes E.1 and E.2 for document types application/vnd.oipf.ContentAccessDownload+xml and application/vnd.oipf.ContentAccessStreaming+xml.

c) Mandatory attribute "Size", which indicates the size of the content item in bytes. If the size is unknown (e.g. in case of streaming), the value of this element is -1. If the value is greater or equal to 0, the value given here SHALL correspond to the value given to the Content-Size HTTP header if the content is fetched through an HTTP ContentURL. If after downloading the content item the size of the downloaded content item does not match the indicated size parameter, the OITF SHALL report failed download (if the application/oipfDownloadManager object is used an event is dispatched to the onDownloadStateChange listener(s) with reason code 3, "The item is invalid due to bad checksum or length"). The OITF SHOULD remove the downloaded content item

d) Optional attribute "MD5Hash", which indicates the MD5 hash value [RFC1321] of the content item. This value is used to check the correctness of the downloaded file. If after downloading the content item the MD5 hash value of the downloaded content item does not match the indicated MD5 hash value, it is recommended for the OITF to remove the downloaded content item.

e) Optional attribute "Duration", which indicates the media playback duration of the media item in the following form "hh:mm:ss".

f) Mandatory attribute "MIMEType", which indicates the MIME type of the content item. It is recommended for an OITF to inform the user if the content-type of a content item being retrieved cannot be interpreted by the OITF.

g) Optional attribute "MediaFormat", which describes the media format of the content item. The value of this element should be one of the terms defined by the AVMediaFormatCS classification scheme specified in [OIPF_META2].

h) Optional attribute "VideoCoding", which describes the coding format of the video. The value of this element should be one of the terms defined by the VisualCodingFormatCS classification scheme defined in [OIPF_META2].

i) Optional attribute "AudioCoding", which describes the coding format of the audio. The value of this element should be one of the terms defined by the AudioCodingFormatCS classification scheme defined in [OIPF_META2].

Multiple <ContentURL> elements may be included for a single <ContentItem>, as long as each <ContentURL> element has a different value for the "DRMSystemID" attribute.

9) <***MetadataURL***> - optional element which indicates the URL from which additional metadata can be fetched for the content item, such as artwork, subtitle files. By default the metadata must be a text/xml document formatted according to TV anytime, as defined in [OIPF_META2].

10) <***NotifyURL***> - optional element which indicates the URL to which an HTTP GET request SHALL be made by the OITF, after the content-item has been fully and successfully fetched, in order to inform the server of the succesful completion of the transfer. If any content is returned from the <NotifyURL>, it MAY be shown in the browser.

11) <***IconURL***> - optional element which indicates the URL of an image which is a visual representation of the item that is being downloaded. Valid content types include the image formats as listed in Section 9 of [OIPF_MEDIA2].

12) <***ParentalRating***> - optional element which indicates the parental rating value (e.g. "PG-13") for this content item. The element has the following attributes:

a) Attribute “Scheme”, which indicates the name of the parental rating scheme that is used for indicating the value. Valid rating scheme names include the ParentalRating classification scheme names as identified by property “scheme” of the ParentalRating object as defined in Section 7.9.4.

b) Attribute “Region”, which indicates the region to which the parental rating applies. Valid region names include the case-insensitive alpha-2 region codes as defined in ISO 3166-1.

Multiple <ParentalRating> elements may exist, as long as each <ParentalRating> element has a different value for the “Scheme” or the “Region” attribute.

13) <***DRMControlInformation***> - optional element which allows the inclusion of DRM related information that SHALL be passed to the DRM agent. This element SHALL adhere to the DRMControlInformation Type Semantics as defined in table 8 of Section 3.3.2 of [OIPF_META2]. For Marlin, additional semantics are defined in Section 4.1.5 of [OIPF_CSP2]. This element SHALL be included for any DRM System ID for which a corresponding “DRMSystemID” value was specified as attribute of a <ContentURL> element.

Multiple <DRMControlInformation> elements MAY be included for a single <ContentItem>, as long as each <DRMControlInformation> element has a different value for its “DRMSystemID” child element.

An Abstract Content Access Descriptor SHALL adhere to the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
<!-- schema filename is  iptv-AbstractContentAccessDescriptor.xsd -->
<!-- this is the generic (i.e. "abstract") content access descriptor XML Schema that forms the
basis for the XML Schemas of document types: application/vnd.oipf.ContentAccessDownload+xml and
application/vnd.oipf.ContentAccessStreaming+xml. This schema includes the definition for
abstract type "DRMPrivateDataType" (as defined in Open IPTV Forum Solution Specification Volume
3 Metadata Release 1) and its specific instance type "MarlinPrivateDataType" or
"HexBinaryPrivateDataType" (as defined in Open IPTV Forum Solution Specification Volume 7
Authentication, Content Protection and Service Protection Release 1) -->
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
           schemaLocation="http://www.w3.org/2001/xml.xsd"/>
<xs:include schemaLocation="csp-MarlinPrivateDataType.xsd"/>
<xs:include schemaLocation="csp-DRMPrivateDataType.xsd"/>
<xs:include schemaLocation="csp-HexBinaryPrivateDataType.xsd"/>

<xs:element name="Contents" type="ContentsType"/>
  <xs:complexType name="ContentsType">
    <xs:sequence>
      <xs:element name="ContentItem" type="ContItemType" minOccurs="1"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ContItemType">
    <xs:sequence>
      <xs:element name="Title" type="TitleType" minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="Synopsis" type="SynopsisType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="OriginSite" type="xs:anyURI" minOccurs="1"/>
      <xs:element name="OriginSiteName" type="xs:string" minOccurs="0"/>
      <xs:element name="ContentID" type="xs:string" minOccurs="0"/>
      <xs:element name="ContentURL" type="ContentURLType" minOccurs="1"
                  maxOccurs="unbounded"/>
      <xs:element name="MetadataURL" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="NotifyURL" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="IconURL" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="ParentalRating" type="ParentalRatingType" minOccurs="0"
                  maxOccurs="unbounded"/>
      <xs:element name="DRMControlInformation" type="DRMControlInformationType"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TitleType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="xml:lang"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="SynopsisType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="xml:lang"/>
```

```
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="ContentURLType">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:attribute name="DRMSystemID" type="xs:string" use="optional"/>
        <xs:attribute name="TransferType" type="TransferTypeEnum" use="required"/>
        <xs:attribute name="MD5Hash" type="xs:string" use="optional"/>
        <xs:attribute name="Duration" type="xs:time" use="optional"/>
        <xs:attribute name="Size" type="xs:integer" use="required"/>
        <xs:attribute name="MIMEType" type="xs:string" use="required"/>
        <xs:attribute name="MediaFormat" type="xs:string" use="optional"/>
        <xs:attribute name="VideoCoding" type="xs:string" use="optional"/>
        <xs:attribute name="AudioCoding" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<!-- The TransferType is a string in this generic content access descriptor. The values of the
TransferTypeEnum are restricted in the document instance types
"application/vnd.oipf.ContentAccessDownloadDescriptor" and
"application/vnd.oipf.ContentAccessStreamingDescriptor" as defined in Annexes E.1 and E.2.-->
  <xs:simpleType name="TransferTypeEnum">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="ParentalRatingType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Scheme" type="xs:string" use="optional"/>
        <xs:attribute name="Region" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
</xs:complexType>

  <xs:complexType name="DRMControlInformationType">
    <xs:sequence>
       <xs:element name="DRMSystemID" type="xs:string"/>
       <xs:element name="DRMContentID" type="xs:string"/>
       <xs:element name="RightsIssuerURL" type="xs:anyURI" minOccurs="0"/>
       <xs:element name="SilentRightsURL" type="xs:anyURI" minOccurs="0"/>
       <xs:element name="PreviewRightsURL" type="xs:anyURI" minOccurs="0"/>
       <xs:element name="DoNotRecord" type="xs:boolean" minOccurs="0"/>
       <xs:element name="DoNotTimeShift" type="xs:boolean" minOccurs="0"/>
       <xs:element ref="DRMGenericData" minOccurs="0" maxOccurs="unbounded"/>
       <xs:element ref="DRMPrivateData" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

   <xs:element name="DRMGenericData" type="DRMGenericDataType"/>
   <xs:element name="DRMPrivateData" type="DRMPrivateDataType"/>

   <xs:complexType name="DRMGenericDataType">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:element name="MarlinPrivateData" type="MarlinPrivateDataType"
               substitutionGroup="DRMPrivateData"/>
   <xs:element name="HexBinaryPrivateData" type="HexBinaryPrivateDataType"
               substitutionGroup="DRMPrivateData"/>

</xs:schema>
```

An OITF SHALL silently ignore unknown elements and attributes that are part of a content-access descriptor.

Annex F. Capability Extensions Schema

This Annex contains the schema that includes the extensions and modifications to the capability negotiation mechanism as defined in Section 9.3. This schema redefines and adds the necessary extensions to the existing capability description schema as defined in Annex C of [CEA-2014-A]. The schema in this Annex SHALL be used instead of the existing capability description as defined in Annex C of [CEA-2014-A]. Note that for the additional "`0.33x0.33`" value for "`scalingType`" as defined in Section 9.3.15, a special construction has been defined. See the last two paragraphs of this Annex for more information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns="urn:oipf:config:oitf:oitfCapabilities:2009"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:oipf:config:oitf:oitfCapabilities:2009"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- schema filename is  config-oitf-oitfCapabilities.xsd -->
  <!-- Redefined uiExtensionsType of the original schema as defined in Annex C of CEA-2014
       (i.e. imports/ce-html-profiles-1-0.xsd) to add the new elements defined in Section 9.2
       of Open IPTV forum Volume 5 Declarative Application Environment Release 1 specification.
   -->
   <xs:redefine schemaLocation="imports/ce-html-profiles-1-0.xsd">
   <xs:complexType name="uiExtensionType">
      <xs:complexContent>
         <xs:extension base="uiExtensionType">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
               <xs:element name="video_broadcast" type="videoBroadcastType" minOccurs="0"
                           maxOccurs="unbounded"/>
               <xs:element name="overlaylocaltuner" type="overlayType"/>
               <xs:element name="overlayIPbroadcast" type="overlayType"/>
               <xs:element name="recording" type="pvrType"/>
               <xs:element name="parentalcontrol" type="parentalControlType"/>
               <xs:element name="extendedAVControl" type="xs:boolean"/>
               <xs:element name="clientMetadata" type="metadataType"/>
               <xs:element name="configurationChanges" type="xs:boolean"/>
               <xs:element name="ims" type="xs:boolean"/>
               <xs:element name="communication_services" type="xs:boolean"/>
               <xs:element name="drm" type="drmType" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="remote_diagnostics" type="xs:boolean"/>
               <xs:element name="pollingNotifications" type="xs:boolean"/>
               <xs:element name="mdtf" type="xs:boolean"/>
               <xs:element name="widgets" type="xs:boolean"/>
               <xs:element name="html5_media" type="xs:boolean"/>
               <xs:element name="remoteControlFunction" type="xs:boolean"/>
               <xs:element name="wakeupApplication" type="xs:boolean"/>
               <xs:element name="wakeupOITF" type="xs:boolean"/>
               <xs:element name="hibernateMode" type="xs:boolean"/>
            </xs:choice>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <!-- Redefined downloadType to add attribute manageDownloads -->
   <xs:complexType name="downloadType">
      <xs:simpleContent>
          <xs:extension base="downloadType">
             <xs:attribute name="manageDownloads" type="manageDownloadsType" default="none"/>
          </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <!-- Redefined audioProfileType to add attribute DRMSystemID -->
   <xs:complexType name="audioProfileType">
      <xs:complexContent>
         <xs:extension base="audioProfileType">
            <xs:attribute name="DRMSystemID" type="xs:string"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <!-- Redefined videoProfileType to add attribute DRMSystemID -->
   <xs:complexType name="videoProfileType">
      <xs:complexContent>
         <xs:extension base="videoProfileType">
            <xs:attribute name="DRMSystemID" type="xs:string"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   </xs:redefine>
   <!-- ADDED: type definitions for the new elements defined in Section 9.2 of the
        Open IPTV forum Volume 5 Declarative Application Environment Release 1 specification
    -->
   <xs:simpleType name="manageDownloadsType">
      <xs:restriction base="xs:string">
```

```
         <xs:enumeration value="none"/>
         <xs:enumeration value="initiator"/>
         <xs:enumeration value="samedomain"/>
         <xs:enumeration value="all"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="videoBroadcastType">
      <xs:attribute name="type" type="xs:string" use="required"/>
      <xs:attribute name="transport" type="xs:string"/>
      <xs:attribute name="nrstreams" type="xs:unsignedInt" default="1"/>
      <xs:attribute name="scaling" type="scalingType" default="arbitrary"/>
      <xs:attribute name="minSize" type="xs:unsignedInt" default="0"/>
      <xs:attribute name="postList" type="xs:boolean" default="false"/>
   </xs:complexType>
   <xs:complexType name="pvrType">
      <xs:simpleContent>
         <xs:extension base="xs:boolean">
            <xs:attribute name="ipBroadcast" type="xs:boolean" default="false"/>
            <xs:attribute name="manageRecordings" type="xs:string"/>
            <xs:attribute name="postList" type="xs:boolean" default="false"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="parentalControlType">
      <xs:simpleContent>
         <xs:extension base="xs:boolean">
            <xs:attribute name="schemes" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="metadataType">
      <xs:simpleContent>
         <xs:extension base="xs:boolean">
            <xs:attribute name="type" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="drmType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="DRMSystemID" type="xs:string" use="required"/>
            <xs:attribute name="protectionGateways" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

Due to limitations of XML Schema it is not possible to redefine/extend the enumeration of type "`scalingType`" to add the additional value "`0.33x0.33`" as defined in Section 9.3.15. Therefore, this value must be directly added to the original schema as defined in Annex C of [CEA-2014-A] (i.e. imports/ce-html-profiles-1-0.xsd), as follows:

```
 [...]

   <xs:simpleType name="scalingType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="arbitrary"/>
         <xs:enumeration value="quartersize"/>
         <xs:enumeration value="none"/>
         <xs:enumeration value="0.33x0.33"/>
      </xs:restriction>
   </xs:simpleType>
[...]
```

Annex G. Client Channel Listing Format

An OITF that supports sending the Client Channel Listing through the HTTP POST method defined in Section 7.13 SHALL adhere to the XML Schema of the Client Channel Listing defined in this annex for which the following semantics apply:

1) <***ChannelConfig***> - mandatory root element of the Client Channel Listing.

2) <***ChannelList***> - mandatory container element for zero or more <Channel> elements, the order of which corresponds to the channel order as managed by the OITF.

3) <***Channel***> - element that represents a channel that can be received by a tuner of the OITF. The element has the following attributes:

   a) Mandatory attribute "ccid" which specifies a unique identifier of the channel within the scope of the OITF. The format of ccid SHALL have a prefix 'ccid:', e.g., 'ccid:{tuner.}majorChannel{.minorChannel}'. The ccid is defined and managed by the OITF.

   b) Optional attribute "channelType" which indicates the type of media content carried over the channel. Valid values are specified in Section 7.13.12.1. If not included, the default value is "TYPE_OTHER".

   c) Mandatory attribute "idType" which specifies the type of identification that is used for the channel. Valid values are specified in Section 7.13.12.1.

   d) Optional attribute "tunerID" which specifies a unique identifier of the tuner within the scope of the OITF.

4) <***ONID***> - mandatory child element of a <Channel> element of type ID_DVB_* or ID_ISDB_* which specifies the DVB or ISDB original network ID. The value can be empty (i.e. <ONID/>) if stream does not contain an SDT_Actual.

5) <***TSID***> - mandatory child element of a <Channel> element of type ID_DVB_* or ID_ISDB_* which specifies the DVB or ISDB transport stream ID.

6) <***SID***> - mandatory element of a <Channel> element of type ID_DVB_* or ID_ISDB_* which specifies the DVB or ISDB service ID.

7) <***SourceID***> - mandatory child element of a <Channel> element of type ID_ATSC_T which specifies the ATSC source_ID.

8) <***Freq***> - mandatory child element of a <Channel> element of type "ID_ANALOG" which specifies the frequency of the content carrier in kHz.

9) <***CNI***> - optional child element of a <Channel> element of type "ID_ANALOG" which specifies the VPS/PDC confirmed network identifier.

10) <***IPBroadcastID***> - mandatory child element of a <Channel> element of type "ID_IPTV_SDS" or "ID_IPTV_URI". if the channel has type "ID_IPTV_SDS", this element denotes the DVB Textual Service Identifier of the IP broadcast service, specified in the format "ServiceName.DomainName" with the ServiceName and DomainName as defined in
    TS 102 034 V1.3.1. If the channel has type "ID_IPTV_URI", this element denotes the URI of the IP broadcast service.

11) <***MajorChannel***>> - optional child element of a <Channel> element of type "ID_ATSC_*". This element denotes the major channel number, if assigned. Value 0 otherwise.

12) <***MinorChannel***> optional child element of a <Channel> element of type "ID_ATSC_*". This element denotes the minor channel number (in relation to the major channel number as indicated through element <MajorChannel>) if assigned. Value 0 otherwise.

13) <***Name***> - mandatory child element of a <Channel> element which specifies the name of the broadcaster. May be an empty string.

14) <***Favourite***> - optional child element of a <Channel> element indicating that the user has marked this channel as a favourite. The element has the following attribute:

    a) Optional attribute "FavIDS" indicating in which favourite lists, if any, this channel is selected.

15) <***FavouriteLists***> - optional child element of the <ChannelConfig> element containing one or more <FavouriteList> elements.

16) <***FavouriteList***> - mandatory child element of the <FavouriteLists> element that represents a favourite list that is (partially) managed by the OITF. The element has the following attribute:

    a) Mandatory attribute "FavID" which specifies the unique identifier of the favourite list.

17) <***FavName***> - mandatory child element of the <FavouriteList> element specifying the name of the favourite list.

18) <***CurrentFavouriteList***> - conditionally optional child element of the <ChannelConfig> element specifying the currently active favourite list.

19) <***Recordable***> - optional child element of a <Channel> element indicating whether the channel can be recorded. Valid values include "True" or "False". If this element is not included, the default value is "False". The value SHALL be ignored if the OITF did not indicate support for control of its recording functionality.

20) <***Locked***> - optional child element of a <Channel> element indicating whether the current state of the parental control system prevents the channel from being viewed (e.g. a correct parental control pin has not been entered). Valid values include "True" or "False". If this element is not included, the default value is "False".

21) <***ManualBlock***> - optional child element of a <Channel> element indicating whether the user has manually blocked viewing of this channel. Manual blocking of a channel treats the channel as if its parental rating value always exceeded the system threshold. Valid values include "True" or "False". If this element is not included, the default value is "False".

A valid Client Channel Listing SHALL adhere to the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="ChannelConfig">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ChannelList"/>
        <xs:sequence minOccurs="0">
          <xs:element ref="FavouriteLists"/>
          <xs:element ref="CurrentFavouriteList" minOccurs="0"/>
        </xs:sequence>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ChannelList">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Channel" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Channel">
    <xs:annotation>
      <xs:documentation>
        For a DVB digital channel use ONID+TSID+SID,
        for an ISDB (ARIB) digital channel use ONID+TSID+SID,
        for a ATSC terrestrial channel use SourceID,
        for analog channel use Freq and CNI (if available).
        The IPBroadcastID element is relevant for IPTV broadcasts, as defined in Section 7.5.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:choice>
          <xs:sequence>
            <xs:element ref="ONID"/>
            <xs:element ref="TSID"/>
```

```
            <xs:element ref="SID"/>
          </xs:sequence>
          <xs:element ref="SourceID"/>
          <xs:sequence>
            <xs:element ref="Freq"/>
            <xs:element ref="CNI" minOccurs="0"/>
          </xs:sequence>
          <xs:element ref="IPBroadcastID"/>
        </xs:choice>
        <xs:element ref="Name"/>
        <xs:element ref="Favourite" minOccurs="0"/>
        <xs:element ref="Recordable" minOccurs="0"/>
        <xs:element ref="Locked" minOccurs="0"/>
        <xs:element ref="ManualBlock" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="CCID" type="xs:ID" use="required"/>
      <xs:attribute name="channelType" type="xs:string" default="TYPE_OTHER"/>
      <xs:attribute name="idType" type="xs:string" use="required"/>
      <xs:attribute name="TunerID" type="xs:ID" minOccurs="0"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="ONID" type="xs:integer"/>
  <xs:element name="TSID" type="xs:integer"/>
  <xs:element name="SID" type="xs:integer"/>
  <xs:element name="SourceID" type="xs:integer"/>
  <xs:element name="Freq" type="xs:integer"/>
  <xs:element name="CNI" type="xs:integer"/>
  <xs:element name="IPBroadcastID" type="xs:string"/>
  <xs:element name="MajorChannel" type="xs:integer"/>
  <xs:element name="MinorChannel" type="xs:integer"/>
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Favourite">
    <xs:complexType>
      <xs:attribute name="FavIDS" type="xs:IDREFS"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="FavouriteLists">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FavouriteList" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="FavouriteList">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="FavName">
          <xs:attribute name="FavID" type="xs:ID" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="FavName">
    <xs:sequence>
      <xs:element ref="FavName"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="FavName" type="xs:string"/>
  <xs:element name="CurrentFavouriteList">
    <xs:complexType>
      <xs:attribute name="FavID" type="xs:IDREF" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Recordable" type="xs:boolean"/>
  <xs:element name="Locked" type="xs:boolean"/>
  <xs:element name="ManualBlock" type="xs:boolean"/>
</xs:schema>
```

Annex H. Display model

H.1 Logical Plane Model

Digital TV terminals typically have multiple planes for displaying graphics, subtitles, video and background color. This section defines a logical plane model for OITFs. Figure 19 shows the ordering of these logical planes.

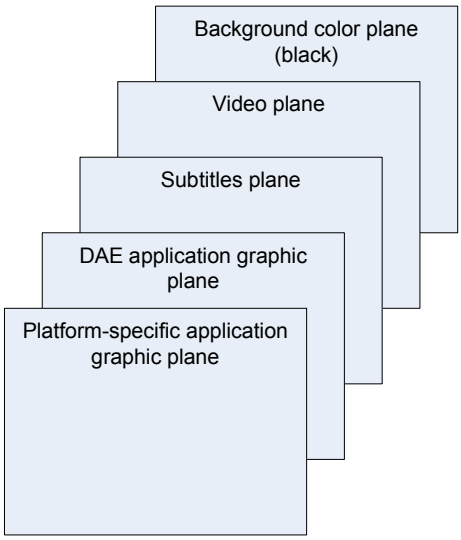


Figure 19: Logical Plane Model

This logical plane model does not imply any particular physical implementation. For instance, the presence of two graphic planes and a subtitle plane does not imply a requirement for three hardware graphic planes.

The logical planes are defined as follows:

- The “Background color plane” displays a single uniform color which shall be black. This plane is at the bottom of the logical display stack.
- The “Video plane” is used to display video. This plane is on top of the background color plane in the logical display stack. The interaction between the “video plane” and the video/broadcast object is described in Section 10.1.2. Streamed video may appear to be presented in a plane other than the logical video plane. The present document is intentionally silent about the mechanism used by an OITF to achieve this behaviour
- The “Subtitles plane” is used to display subtitles. This plane is on top of the video plane in the logical display stack.
- The “DAE application graphic plane” is used to display any running DAE applications. This plane is on top of the subtitles plane in the logical display stack. The logical resolution of this plane is given by the `<width>` and `<height>` elements of the capability description.
- The “Platform-specific application graphic plane” is used to display applications specific to the OITF such as native system menus, banners or pop-ups. This plane is on top of the DAE application graphic plane in the logical display stack.

For subtitles, the following rules apply:

- OITFs SHOULD support simultaneous display of application and subtitles. In that case, the OITF SHALL display the application over the subtitles (as shown in Figure 19). If the video is rescaled, the subtitles SHOULD be rescaled/repositioned appropriately or not displayed at all.
- If the presentation of subtitles is requested prior to the launch of an application, then OITFs which cannot support simultaneous display of applications and subtitles SHALL display subtitles in preference to running the application. The OITF may offer the end-user the opportunity to disable subtitles and run the application instead.

- If the presentation of subtitles is requested while an application is running, OITFs which cannot support simultaneous display of applications and subtitles shall display applications in preference to the presentation of subtitles.

H.2 Interaction with the video/broadcast and A/V Control objects

The behaviour of the `video/broadcast` object is defined in section 7.13.1.1. When no `video/broadcast` object is instantiated, or when all `video/broadcast` objects are in the `Unrealized` state, broadcast video presentation SHALL be under the control of the OITF. When video is under the control of the OITF:

- Any broadcast video being presented SHALL be displayed in the logical video plane.
- The complete logical video plane SHALL be filled.
- The OITF MAY scale and/or position video, for example to remove black bars.

For broadcast related applications as defined in section 5.2.3, broadcast video presentation SHALL initially be under the control of the OITF. Applications wanting to control video presentation SHALL create a video/broadcast object.

When a `video/broadcast` object is in any state other than the `Unrealized` state, broadcast video presentation SHALL be under the control of the application. When video is under the control of the application:

- When the `video/broadcast` object or A/V Control object is not in "full-screen mode", any video being presented SHALL be scaled and positioned to fit the object. The area of the video plane not containing video SHALL be transparent.
- When the `video/broadcast` object or A/V Control object is in "full-screen mode", presented video SHALL be scaled to fill the entire logical video plane. The OITF MAY further scale and/or position video, for example to remove black bars.
- Depending on the Z index of the `video/broadcast` or A/V Control object with respect to other HTML elements (regardless of whether the object is in "fullscreen mode" or not), presented video may fully or partially obscure other HTML elements with a lower Z index, and may in turn be fully or partially obscured by HTML elements with a higher Z index. As a result of this, video may appear to be presented in a plane other than the logical video plane. This specification is intentionally silent about the mechanism used by an OITF to achieve this behaviour.
- Calling the `Application.hide()` method SHALL cause video (and any subtitles) being presented under the control of that application to be hidden, and any audio being presented by the `video/broadcast` or A/V Control object under the control of that application to be muted. Calling `Application.show()` SHALL cause video and audio presentation to be restored.

If the `release()` method is called on a `video/broadcast` object, or if the object is garbage collected, control of broadcast video presentation SHALL be returned to the OITF and video SHALL be re-scaled and re-positioned (if necessary).

H.3 Graphic safe area (Informative)

Figure 20 shows the recommended safe area for content authoring for the OITF_HD_UIPROF default profile:

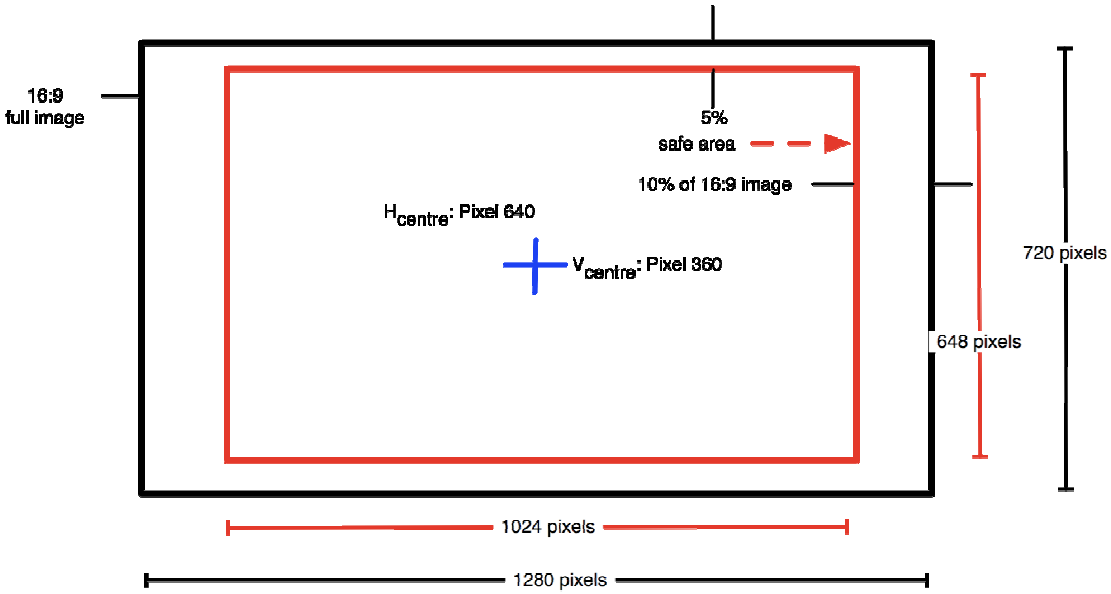


Figure 20: Graphic Safe Area

Annex I. HTML 5 Video Tag Support

This section provides a comparison between HTML5 <video> and the visual objects defined in this specification. When not supported by HTML5 it is indicated with NS (Not Supported). When not in the scope of HTML 5 it is indicated with NA (Not Applicable). If there are differences in values or behavior additional information is provided under the Comments column.

	A/V Control Object	Broadcast object	HTML5 IDL attributes	Comments
General	Number width	Integer width	video.videoWidth	
	Number height	Integer height	video.videoHeight	
	readonly Boolean fullScreen	readonly Boolean fullScreen	NS	Not in HTML5 because of security issues
	setFullScreen (Boolean fullscreen)	void setFullScreen(Boolean fullscreen)	NS	Not in HTML5 because of security issues
	focus ()		window.focus()	
	Object onfocus	function onfocus	onfocus	
	Object onblur	function onblur	onblur	
	Object onFullScreenChange	function onFullScreenChange	NS	Not in HTML5 because of security issues
Volume	Boolean setVolume(Number volume)	Boolean setVolume(Integer volume)	float media.volume	The HTML5 value is in a range between 0 and 1, whereas the DAE visual objects are between 0 and 100
			boolean media.muted	
			boolean media.controls	true if the user agent should provide its own set of controls
			onvolumechange	
		Integer getVolume()	float media.volume	
Components (ex. subtitles, languages)	AVComponentCollection getComponents(Integer componentType)	AVComponentCollection getComponents(Integer componentType)	NS	Subtitles and media annotations not currently in HTML5 (but proposals exist)
	AVComponentCollection	AVComponentCollection	NS	

	getCurrentActiveComponents(Integer componentType)	getCurrentActiveComponents(Integer componentType)		
	void selectComponent(AVComponent component)	void selectComponent(AVComponent component)	NS	
	void unselectComponent(AVComponent component)	void unselectComponent(AVComponent component)	NS	
Broadcast specific		function onChannelChangeError(Channel channel, Number errorState)	NA	
		Integer playState	NA	
		function onPlayStateChange(Number state, Number error)	NA	
		Channel bindToCurrentChannel()	NA	
		void setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)	NA	
		void prevChannel()	NA	
		void nextChannel()	NA	
		void release()	NA	
		void setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL, Integer offset)	NA	
		readonly Channel currentChannel	NA	
Playback control	String data			video.url
	readonly Number playPosition	readonly Integer playPosition	attribute float currentTime; (get)	
			readonly attribute float startTime;	
	readonly Number playTime		readonly attribute float duration	
	readonly Number playState		readonly attribute boolean paused; readonly attribute	

			boolean ended;	
	readonly Number error		attribute int media.error and error / abort events	
	readonly Number speed	readonly Number playSpeed	attribute float defaultPlaybackRate; attribute float playbackRate;	

	Boolean play (Number speed)	Boolean resume()
		Boolean pause()
			void play();
			void pause();
			attribute boolean autoplay;
			attribute boolean loop;
				Recording aspects not covered

		Boolean setSpeed(Number speed)	attribute float playbackRate;	
	Boolean stop ()	void stopRecording()	NA (no recording support)	Stop functionality can be implemented with pause();currentTime=0;
		Boolean stopTimeshift()	NA (no recording support)	
	Boolean seek (Number pos)	Boolean seek(Integer offset, Integer reference)	attribute float currentTime; (set)	The HTML5 values are in seconds, whereas the DAE values are in milliseconds.
	Boolean next ()		NS (no playlist support)	
	Boolean previous ()		NS (no playlist support)	
			readonly attribute TimeRanges played;	
			readonly attribute TimeRanges seekable;	
	function onPlaySpeedChanged(Number speed)	function onPlaySpeedChanged(Number speed)	events: ratechange durationchange	

	script onPlayPositionChanged(Integer position)	function onPlayPositionChanged(Integer position)	event: timeupdate	
	readonly Number playSpeeds[ ]	readonly Number playSpeeds[ ]	NS	
	readonly String oitfSourceIPAddress		NA	□
	readonly String oitfSourcePortAddress		NA	
	Boolean oitfNoRTSPSessionControl		NA	
	String oitfRTSPSessionId		NA	
Recording specific		String recordNow(Integer duration)	NA	□
		readonly Integer playbackOffset	NA	
		readonly Integer maxOffset	NA	
		readonly Integer recordingState	NA	
		function onRecordingEvent	NA	
		readonly Integer state	NA	
		readonly Integer error	NA	
		readonly String recordingId	NA	

Annex J. DLNA RUI Remote Control Function Sequences

There are two cases to send the control UI to the Remote Control Device:

- First, when the DAE application is created (for example, when loaded in response to a request from the Remote Control Device), the DAE application SHALL try to give a proper control UI to the Remote Control Device (Creating DAE app → finding the Remote Control Device handle → giving the control UI). See Annex J.1.

  The DAE application is launched in response to an HTTP request from an OITF control UI being rendered in the Remote Control Device. The DAE application checks the `currentRemoteDeviceHandle` property when it has completed loading. If this property returns "`undefined`", it means that the current DAE application wasn't launched by a Remote Control Device (but by some other means), whereas if this property returns a value (the Remote Control Device handle), the DAE application knows that it must send its Control UI to the Remote Control Device.

  This scenario is made based on Section 10.6 of [OIPF_ARCH2].

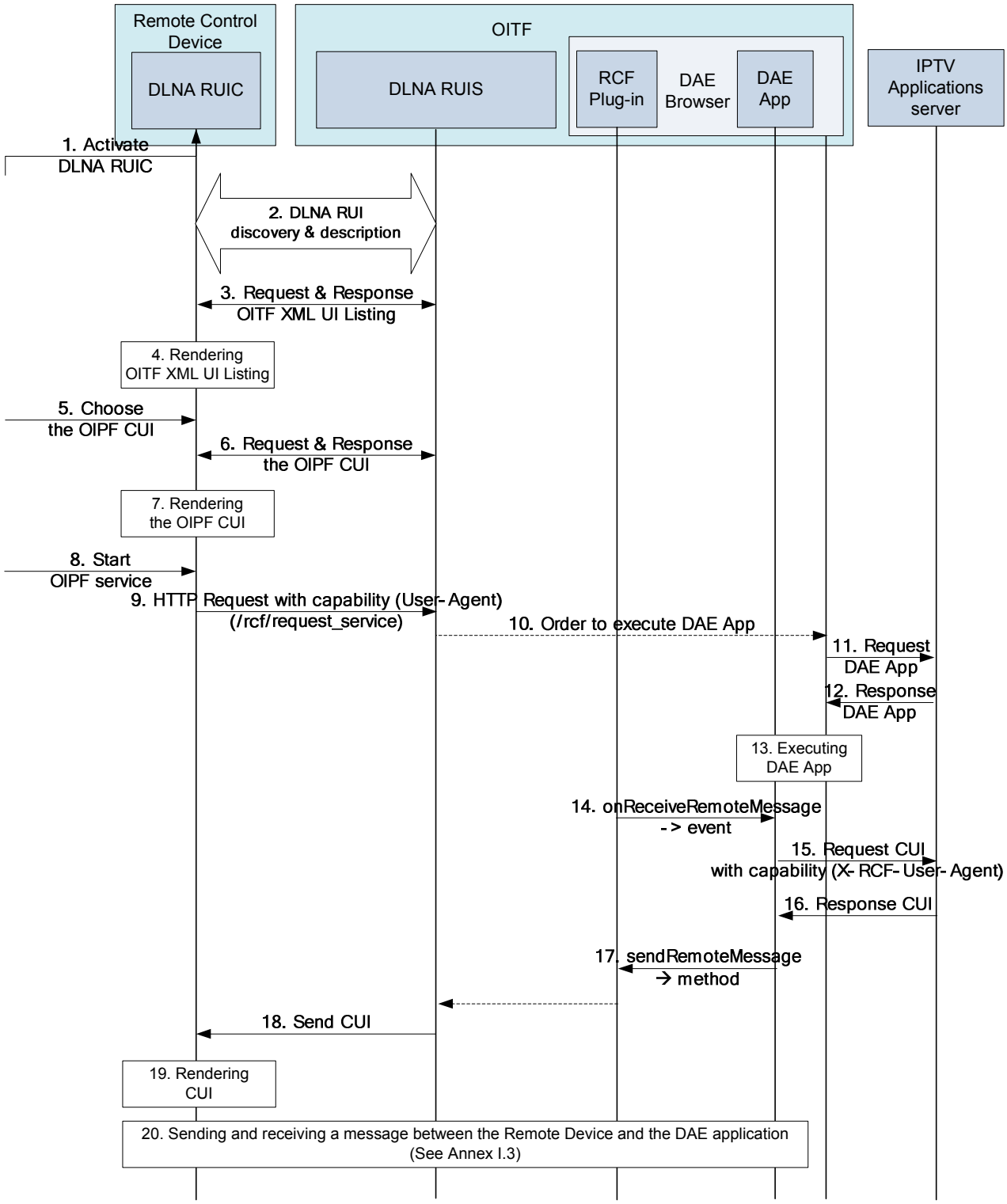
- Second, when the DAE application is already running, the DAE application sends a control UI in response to a control UI request (DAE app running → getting the CUI request event → giving the control UI). See Annex J.2.

  The DAE application is currently being executed in the OITF and during this time the Remote Control Device requests the control UI from it. In this case, the OITF generates the `ReceiveRemoteMessage` event to the DAE application with type set to 0. Then the DAE application retrieves the control UI from the IPTV Applications server and returns it to the Remote Control Device.

Annex J.3 shows the message flow for sending and receiving messages between control UI in the Remote Control Device and the DAE application.

NOTE: Dotted lines in the diagrams below indicate internal operations.

J.1 Launching a DAE application to obtain the Control UI



The following is a brief description of the steps in the flow:

Note: The dotted line is an internal operation.

1) The user activates a DLNA RUIC function.

2) The DLNA RUIC discovers the DLNA RUIS in the OITF as defined in Section 5.1 of [CEA-2014-A] , and the DLNA RUIC and the DLNA RUIS perform capability profile matching using the mechanism defined in Section 5.2 of [CEA-2014-A].

3) DLNA RUIC requests XML UI Listing to DLNA RUIS, and gets it.

4) DLNA RUIC renders XML UI Listing in its own screen.

5) The user chooses the OIPF CUI in the XML UI Listing.

6) DLNA RUIC requests the OIPF CUI and gets it. (This OIPF CUI could be made based on Open IPTV Forum Metadata information)

7) DLNA RUIC renders the OIPF CUI.

8) The user starts OIPF service with the OIPF CUI which came from DLNA RUIS in the OITF.

Note: The steps from step 1 to step 8 conform to the normal DLNA RUI sequence.

9) The OIPF CUI in the DLNA RUIC sends the OIPF service HTTP Request with capability in the User-Agent to DLNA RUIS. The OIPF service HTTP Request is vender specific URI to create DAE application.

10) DLNA RUIS orders the DAE Browser to execute the requested DAE application.

11) DAE Browser requests the DAE application.

12) IPTV Applications server sends the requested DAE application.

13) DAE Browser executes the DAE application.

14) When the DAE application is loaded, the OITF dispatches a `ReceiveRemoteMessage` event with type `CREATE_APP` to the `application/oipfRemoteControlFunction` object in the DAE application.

15) The DAE application requests the CUI by using XMLHTTPRequest object with capability of DLNA RUIC.

16) The IPTV Applications server sends the CUI.

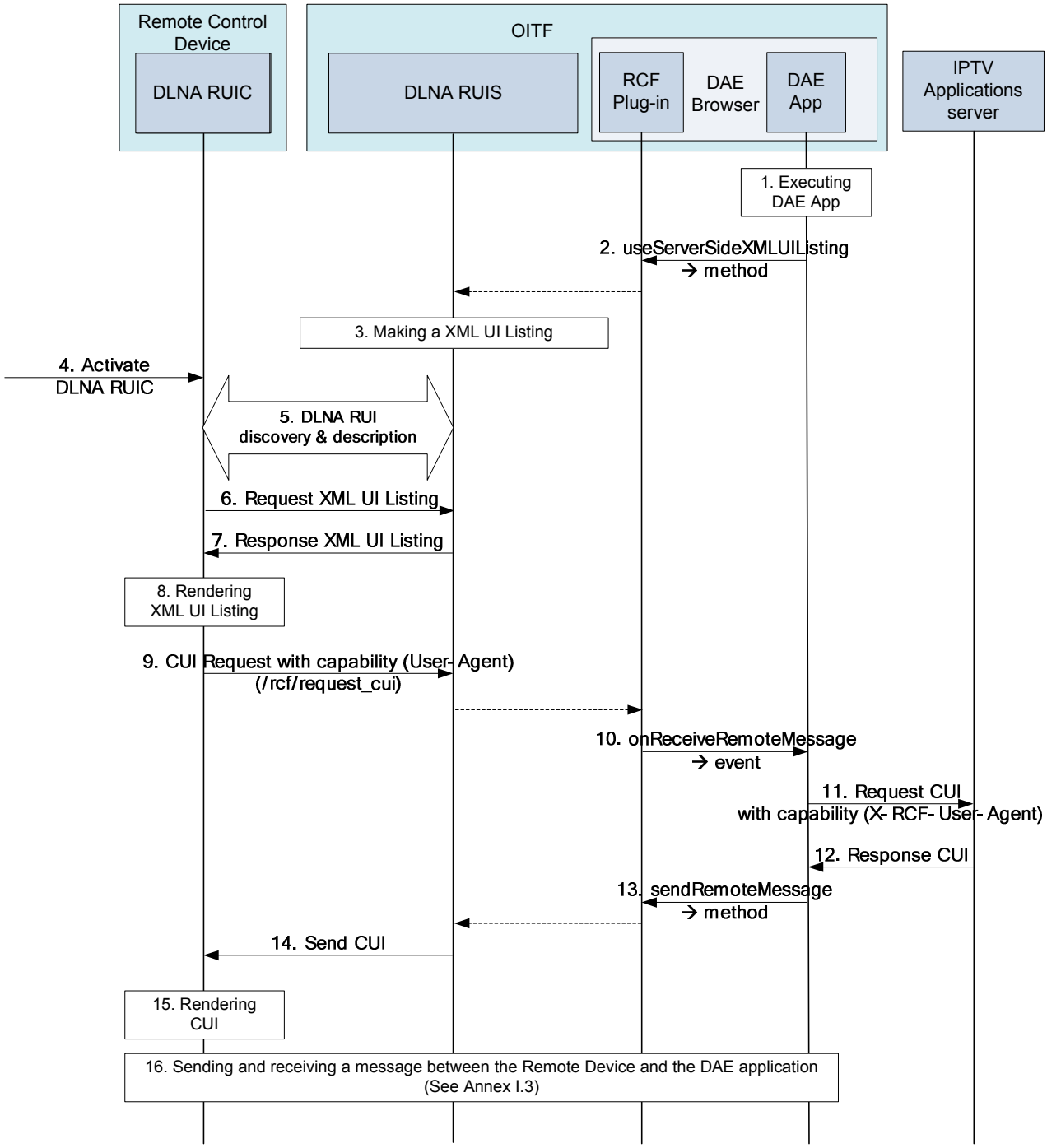
17) The DAE application sends the CUI to the `application/oipfRemoteControlFunction` object by using the `sendRemoteMessage()` method.

18) DLNA RUIS sends the content of the CUI CE-HTML document to DLAN RUIC through a HTTP Response body.

19) DLNA RUIC renders the CUI. DLNA RUIC fetches resources (images/css/js) directly from the IPTV application server.

20) DLNA RUIC sends a message to the DAE application and receive the response message.

J.2 Obtaining the control UI from a running DAE application



The following is a brief description of the steps in the flow:

Note: The dotted line is an internal operation.

1) DAE application which has the `application/oipfRemoteControlFunction` object is being executed.

2) The Server Side XML UI Listing is updated in the DLNA RUIS through the `useServerSideXMLUIListing()` method.

3) The user activates a DLNA RUIC function.

4) The DLNA RUIC discovers the DLNA RUIS in the OITF as defined in Section 5.1 of [CEA-2014-A] , and the DLNA RUIC and the DLNA RUIS perform capability profile matching using the mechanism defined in Section 5.2 of [CEA-2014-A].

5) DLNA RUIC requests XML UI Listing to DLNA RUIS.

6) DLNA RUIS sends the Server side XML UI Listing to the DLNA RUIC.

7) DLNA RUIC renders XML UI Listing in its own screen.

8) When a user chooses one of the CUIs in the XML UI Listing, DLNA RUIC sends the HTTP request message (`/rcf/request_cui`) with the RUIC capability information in the User-Agent to DLNA RUIS to get the CUI.

9) The `application/oipfRemoteControlFunction` object dispatches a `ReceiveRemoteMessage` event with type `REQUEST_CUI` to the DAE application.

10) The DAE application requests the CUI using XMLHTTPRequest object, including the capability description received from the RUIC in the request.

11) The IPTV Applications server sends the CUI.

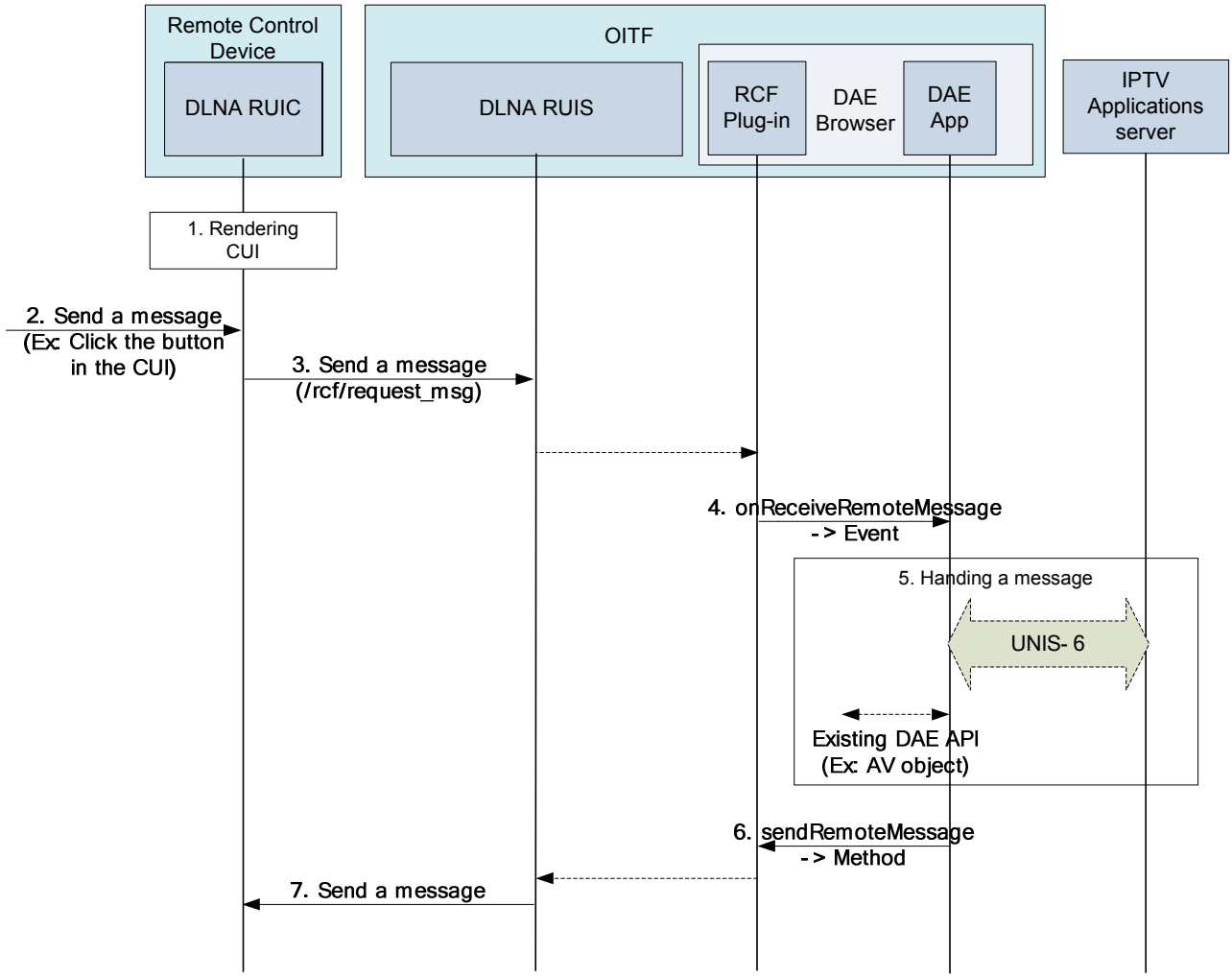
12) The DAE application sends the CUI to the `application/oipfRemoteControlFunction` object by using the `sendRemoteMessage()` method.

13) DLNA RUIS sends the content of the CUI CE-HTML to DLAN RUIC (+RUIPL+) by using HTTP Response body

14) DLNA RUIC renders the CUI. DLNA RUIC fetches resources (images/css/js and any other HTML documents) directly from the IPTV application server.

15) DLNA RUIC sends a message to the DAE application and receive the response message as described in annex I.3.

J.3 Sending and receiving messages between the Remote Control Device and DAE application



The following is a brief description of the steps in the flow:

1) DLNA RUIC renders the CUI.

2) User sends a message to the DAE application. For example, user clicks a button which could send a specific message to the DAE application.

3) The CUI sends a message to the DLNA RUIS by using a pre-defined URL (`/rcf/request_msg`).

4) The `application/oipfRemoteControlFunction` object dispatches a `ReceiveRemoteMessage` event with type `REQUEST_MSG` to the DAE application.

5) The DAE application handles the message received from the DLNA RUIC.

6) The DAE application sends a message to the `application/oipfRemoteControlFunction` object by using a `sendRemoteMessage()` method.

7) DLNA RUIS sends a message to DLNA RUIC.

Annex K. ECMAScript Conventions

In the documented APIs ECMAScript attributes are read-write unless otherwise specified.

The type "`Integer`" is not a valid Javascript type as is. It is used as a short hand notation for a subset of type "`Number`" which includes only the numbers that can be written without a fractional or decimal component.

K.1 Collections

This document defines a number of ECMAScript collections, used by APIs to return lists of objects from the OITF to applications (e.g. lists of channels or EPG search results). Many of these collections have identical semantics, and so for the sake of brevity, the following notation is used to define these collections.

Each collection is an instance of the `Collection<T>` parameterized class (see Section K.1.1), and is defined in the following way:

```
typedef Collection<Foo> FooCollection
typedef Collection<Bar> BarCollection
```

where `Foo` or `Bar` is the name of the class that may be stored in the collection. For example:

```
typedef Collection<String> StringCollection
typedef Collection<Channel> ChannelList
```

Collections defined in this way SHALL follow the semantics defined in Section K.1.1, and may be extended with additional properties and methods as necessary.

Collections defined in this way always represent snapshots of the state of the OITF at a given time. They are not updated automatically if the state of the OITF changes. This means that different instances of a specific type of collection may contain different values.

K.1.1 The Collection template

The `Collection<T>` class is a parameterized class whose instances are (possibly zero-length) collections of values of type *T*. The properties and methods defined below SHALL be present on any instance of a `Collection<T>` class. Instances of a `Collection<T>` class SHALL support the use of array notation to access objects in the collection.

Instances of a `Collection<T>` class SHALL be considered to be immutable, except by APIs defined on the collection. Attempts to insert items into instances of a `Collection<T>` class using array notation SHALL fail.

11.4.1.1 K.1.1.2 Properties

`readonly Integer` **`length`**
The number of items in the collection

11.4.1.2 K.1.1.2 Methods

<T> **`item`**`( Integer index )`		
Description	Return the item at position index in the collection, or `undefined` if no item is present at that position.	
Arguments	*index*	The index of the item that SHALL be returned

Annex L. SVG Video Tag Support

This section provides a comparison between SVG <video> and the visual objects defined in this specification. When not supported by SVG it is indicated with NS (Not Supported). When not in the scope of SVG it is indicated with NA (Not Applicable). If there are differences in values or behavior additional information is provided under the Comments column.

	A/V Control Object	Broadcast object	SVG IDL attributes	Comments
General	Number width	Integer width	Video element: width attribute	
	Number height	Integer height	Video element: height attribute	
	readonly Boolean fullScreen	readonly Boolean fullScreen	Video element: viewbox attribute	
	setFullScreen (Boolean fullscreen)	void setFullScreen(Boolean fullscreen)	Video element: viewbox attribute	
	focus ()		NS	
	Object onfocus	function onfocus	DOM2 Event Model: DOMFocusIn	
	Object onblur	function onblur	DOM2 Event Model: DOMFocusOut	
	Object onFullScreenChange	function onFullScreenChange	NS	
Volume	Boolean setVolume(Number volume)	Boolean setVolume(Integer volume)	Audio element: audio-level attribute	The range specified in SVG is 0 to 1.0 with 0 silencing the audio.
		Integer getVolume()		
Components (ex. subtitles, languages)	AVComponentCollection getComponents(Integer componentType)	AVComponentCollection getComponents(Integer componentType)	audioLanguage = ‘auto’ | <list-of-language-ids> subtitleLanguage = ‘auto’ | <list-of-language-ids> audioType = ‘auto’ | ‘normal’ | ‘descriptive’ subtitleType = ‘auto’ | ‘normal’ | ‘hearingImpaired’	

			| ‘none’ teletextType = ‘auto’ | ‘normal’ | ‘none’	
	AVComponentCollection getCurrentActiveComponents(Integer componentType)	AVComponentCollection getCurrentActiveComponents(Integer componentType)		
	void selectComponent(AVComponent component)	void selectComponent(AVComponent component)		
	void unselectComponent(AVComponent component)	void unselectComponent(AVComponent component)		
Broadcast specific		function onChannelChangeError(Channel channel, Number errorState)	NA	
		Integer playState	NA	
		function onPlayStateChange(Number state, Number error)	NA	
		Channel bindToCurrentChannel()	NA	
		void setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL)	NA	
		void prevChannel()	NA	
		void nextChannel()	NA	
		void release()	NA	
		void setChannel(Channel channel, Boolean trickplay, String contentAccessDescriptorURL, Integer offset)	NA	
		readonly Channel currentChannel	NA	
Playback control	String data		Video element: xlink:href attribute	
	readonly Number playPosition	readonly Integer playPosition	NS	
	readonly Number playTime		NS	
	readonly Number playState		NS	

	readonly Number error			
	readonly Number speed	readonly Number playSpeed	NS	
	Boolean play (Number speed)	Boolean resume() Boolean pause()	Media element: pause/resume attributes SMIL: speed attribute	
		Boolean setSpeed(Number speed)	NA	
	Boolean stop ()	void stopRecording()	Media element: end attribute stopRecording is NA	
		Boolean stopTimeshift()	NA	
	Boolean seek (Number pos)	Boolean seek(Integer offset, Integer reference)	Media element: begin attribute	
	Boolean next ()		NS	
	Boolean previous ()		NS	
	function onPlaySpeedChanged(Number speed)	function onPlaySpeedChanged(Number speed)	NS	
	script onPlayPositionChanged(Integer position)	function onPlayPositionChanged(Integer position)	NS	
	readonly Number playSpeeds[ ]	readonly Number playSpeeds[ ]	NS	
	readonly String oitfSourceIPAddress		NA	
	readonly String oitfSourcePortAddress		NA	
	Boolean oitfNoRTSPSessionControl		NA	
	String oitfRTSPSessionId		NA	
Recording specific		String recordNow(Integer duration)	NA	
		readonly Integer playbackOffset	NA	
		readonly Integer maxOffset	NA	
		readonly Integer recordingState	NA	

		function onRecordingEvent	NA	
		readonly Integer state	NA	
		readonly Integer error	NA	
		readonly String recordingId	NA	