Test Specification

For HbbTV Test Suite Version 2025-2

Version 1.0

July 2025

Contents

1 General

1.1 Scope

The scope of this Test Specification is to describe the technical process for verification of HbbTV Devices for conformance with the following HbbTV Specifications:

- ETSI TS 102 796 V1.1.1 (informally known as HbbTV 1.0) + errata
- ETSI TS 102 796 V1.2.1 (informally known as HbbTV 1.5) + errata
- ETSI TS 102 796 V1.3.1 (informally known as HbbTV 2.0)
- ETSI TS 102 796 V1.4.1 (informally known as HbbTV 2.0.1)
- ETSI TS 102 796 V1.5.1 (informally known as HbbTV 2.0.2)
- ETSI TS 102 796 V1.6.1 (informally known as HbbTV 2.0.3)

This document targets HbbTV Testing Centers and Quality Assurance Departments of HbbTV Licensees.

The HbbTV Test Specification contains five parts:

1) A part that describes the HbbTV Test System and its components.
2) A Test Case creation part that describes the Test Case creation process.
3) HbbTV JavaScript API's and playout definitions used for Test Case implementation
4) Information about the versioning of HbbTV Test specifications.
5) The description of the Test Report format.

The process that is used to define the HbbTV Test Specification, Test Cases and implementation of Test Tools from the HbbTV Specification is depicted in Figure 1.

1) From the HbbTV Specification, Test Cases are defined
2) From the total set of Test Cases the HbbTV Test Specification can be generated
3) From the HbbTV Test Specification the design and implementation of Test Tools will be designed

The grey highlighted elements in Figure 1 are provided by the HbbTV Testing Group.

1.2 Conformance and reference

HbbTV devices shall conform to the applicable parts of the relevant HbbTV Specification.

2 Conditions of publication

2.1 Copyright

The TEST SPECIFICATION for HbbTV is published by the HbbTV Association. All rights are reserved. Reproduction in whole or in part is prohibited without express and prior written permission of the HbbTV Association.

2.2 Disclaimer

The information contained herein is believed to be accurate as of the data of publication; however, none of the copyright holders will be liable for any damages, including indirect or consequential from use of the TEST SPECIFICATION for HbbTV or reliance on the accuracy of this document.

2.3 Classification

The information contained in this document is marked confidential and shall be treated as confidential according to the provisions of the agreement through which the document has been obtained.

2.4 Notice

For any further explanation of the contents of this document, or in case of any perceived inconsistency or ambiguity of interpretation, contact:

HbbTV Association

Contact details: Nguyen Thi Thanh Van (HbbTV Testing Group Chair) van.nguyen@samsung.com

Web Site: http://www.hbbtv.org

E-mail: info@hbbtv.org

3 References

3.1 Normative references

The following referenced documents are required for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI TS 102 796 “Hybrid Broadcast Broadband TV”; V1.4.1. Including all approved Erratas.

[2] VOID

[3] VOID

[4] ETSI TS 102 809 “Digital Video Broadcasting (DVB); Signalling and carriage of interactive applications and services in hybrid broadcast / broadband environments", V1.2.1

[5] VOID

[6] VOID

[7] VOID

[8] RFC2616, IETF: “Hypertext transport protocol – HTTP 1.1”

[9] VOID

[10] VOID

[11] VOID

[12] VOID

[13] VOID

[14] CI Plus Forum, CI Plus Specification, “Content Security Extensions to the Common Interface”, V1.3 (2011-01).

[15] VOID

[16] VOID

[17] VOID

[18] ETSI EN 300 468 “Specification for Service Information (SI) in DVB systems”, V1.10.1

[19] VOID

[20] VOID

[21] ISO 23009-1 (2012): “Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats”

[22] RFC3986, IETF: “Uniform Resource Identifier (URI): Generic Syntax”

[23] RFC4337, IETF: “MIME Type Registration for MPEG-4”

[24] VOID

[25] Document Names for specReference "name" attribute. https://redmine.hbbtv.org/projects/hbbtv-ttg/wiki/DocumentNames

[26] Document Names for specReference "name" attribute. https://redmine.hbbtv.org/projects/hbbtv-ttg/wiki/AppliesToSpecNames

[27] Test Material Challenging Procedure (TMCP), V2.0

[28] W3C Recommendation (16 January 2014): "Cross-Origin Resource Sharing". Available at: http://www.w3.org/TR/2013/PR-cors-20131205/

[29] Open IPTV Forum Release 1 specification, "CSP Specification V1.2 Errata 1", March 2013. Available from http://www.oipf.tv/specifications

[30] DIAL 2nd Screen protocol specification v1.7 – 19 November 2014

NOTE: Available from http://www.dial-multiscreen.org/dial-protocol-specification

[31] RFC6455, IETF: "The WebSocket protocol"

NOTE: Available at https://tools.ietf.org/html/rfc6455

[32] VOID

[33] ECMAScript Language Specification (Edition 5.1), June 2011

NOTE: Available at http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262%205.1%20edition%20June%202011.pdf

[34] ETSI TS 103 286-2 (V1.1.1): "Companion Screens and Streams; Part 2: Content Identification and Media Synchronisation"

[35] UPnP Device Architecture 1.0 - 24 April 2008

NOTE: Available at http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf

[36] UPnP Device Architecture 1.1 - 15 October 2008

NOTE: Available at http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[37] HbbTV Operator Applications Specification 2017-12-15

[38] OIPF 2.3 volume 5a Web Standards TV Profile

[39] Julian Aubourg et al. XMLHttpRequest.. 6 December 2012. W3C Working Draft. URL: http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/

[40] RFC3875, IETF:"The Common Gateway Interface (CGI)" Version 1.1

[41] IP-Delivered Broadcast Channels and Related Signalling of HbbTV Applications 2017-04-07

[42] Optional feature strings. https://redmine.hbbtv.org/projects/hbbtv-ttg/wiki/OptionalFeatures

NOTE: Available at https://www.hbbtv.org/wp-content/uploads/2017/04/HbbTV-SPEC-00012-017-iptv-specification.pdf

3.2 Informative references

[i.1] CEA-2014 revision A, "Web-based Protocol and Framework for Remote User Interface on UPnP™ Networks and the Internet (Web4CE)"

[i.2] ETSI ES 202 130 "Human Factors (HF); User Interfaces; Character repertoires, orderings and assignments to the 12-key telephone keypad (for European languages and other languages used in Europe)", V2.1.2

[i.3] ETSI TS 102 757 "Digital Video Broadcasting (DVB); Content Purchasing API", V1.1.1

[i.4] ETSI TS 101 231 “Television systems; Register of Country and Network Identification (CNI), Video Programming System (VPS) codes and Application codes for Teletext based systems”, V1.3.1

[i.5] ISO/IEC/IEEE 9945:2009 Information technology – Portable Operating System Interface (POSIX®) Base Specifications, Issue 7

[i.6] I. Hickson. The WebSocket API. 20 September 2012. W3C Candidate Recommendation.

NOTE: Available at http://www.w3.org/TR/2012/CR-websockets-20120920/

[i.7] Using the DVB CSA algorithm; ETSI. http://www.etsi.org/about/what-we-do/security-algorithms-and-codes/csa-licences

[i.8] Common Scrambling Algorithm; Wikipedia. https://en.wikipedia.org/wiki/Common_Scrambling_Algorithm

[i.9] W3C Candidate Recommendation (11 December 2008): “Web Content Accessibility Guidelines (WCAG) 2.0”

[i.10] JSONP; Wikipedia https://en.wikipedia.org/wiki/JSONP

[i.11] Wireshark network protocol analyzer https://www.wireshark.org/

[i.12] Netcat networking utility; Wikipedia https://en.wikipedia.org/wiki/Netcat

NOTE: Available at http://www.w3.org/TR/2008/REC-WCAG20-20081211/

4 Definitions and abbreviations

4.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

Application data: set of files comprising an application, including HTML, JavaScript, CSS and non-streamed multimedia files

Assertion: Testable statement derived from a conformance requirement that leads to a single test result
Broadband: An always-on bi-directional IP connection with sufficient bandwidth for streaming or downloading A/V content

Broadcast: classical uni-directional MPEG-2 transport stream based broadcast such as DVB-T, DVB-S or DVB-C

Broadcast-independent application: Interactive application not related to any broadcast channel or other broadcast data

Broadcast-related application: Interactive application associated with a broadcast television, radio or data channel, or content within such a channel

Broadcast-related autostart application: A broadcast-related application intended to be offered to the end user immediately after changing to the channel or after it is newly signalled on the current channel. These applications are often referred to as “red button” applications in the industry, regardless of how they are actually started by the end user

Conformance requirement: An unambiguous statement in the HbbTV specification, which mandates a specific feature or behaviour of the terminal implementation

Digital teletext application: A broadcast-related application which is intended to replace classical analogue teletext services

HbbTV application: An application conformant to the present document that is intended to be presented on a terminal conformant with the present document

HbbTV test case XML description: The HbbTV XML document to store for a single test case information such as test assertion, test procedure, specification references and history. There exists a defined XSD schema for this document format.

HbbTV Test Specification: Refers to this document.

HbbTV Technical Specification: Refers to [1] with the changes as detailed in [20]

Hybrid terminal: A terminal supporting delivery of A/V content both via broadband and via broadcast

Linear A/V content: Broadcast A/V content intended to be viewed in real time by the user

Non-linear A/V content: A/V content that which does not have to be consumed linearly from beginning to end for example, A/V content streaming on demand

Persistent download[1]**:** The non-real time downloading of an entire content item to the terminal for later playback

Playout: Refers to the modulation and playback of broadcast transport streams over an RF output.

Test Assertion: A high level description of the test purpose, consisting of a testable statement derived from a conformance requirement that leads to a single test result.

[1] Persistent download and streaming are different even where both use the same protocol - HTTP. See clause 10.2.3.2 of the HbbTV specifications, ref [1]

NOTE: Not to be confused with the term “assertion” as commonly used in test frameworks e.g. JUnit.

Test Case: The complete set of documents and assets (assertion, procedure, preconditions, pass criteria and test material) required to verify the derived conformance requirement.

NOTE 1: This definition does not include any test infrastructure (e.g. web server, DVB-Playout…) required to execute the test case.

NOTE 2: For the avoidance of doubt, Test Material must be implemented in a way that it produces deterministic and comparable test results to be stored in the final Test Report of this Test Material. Test Material must adhere to the HbbTV Test Specification as defined by the Testing Group.

Test framework/Test tool/Test harness: The mechanism (automated or manually operated) by which the test cases are executed and results are gathered. This might consist of DVB-Playout, IR blaster, Database- and Webservers.

Test material: All the documents (e.g. HTML, JavaScript, CSS) and additional files (DVB-TS, VoD files, static images, XMLs) needed to execute the test case.

Test Procedure: A high level textual description of the necessary steps (including their expected behaviour) to follow in order to verify the test assertion.

Terminal specific applications: Applications provided by the terminal manufacturer, for example device navigation, set-up or Internet TV portal.

Test Report: A single file containing the results of executing one or more Test Suites in the format defined in section 9. This is equivalent to the “Test Report” referred to in the HbbTV Test Suite License Agreement and HbbTV Full Logo License Agreement.

Test Repository: Online storage container for HbbTV test assertions and Test Cases. Access is governed by the TRAA. Link: https://www.hbbtv.org/pages/about_hbbtv/hbbtv_test_repository.php

Test Suite: This means the collection of test cases developed against the HbbTV Specifications.

NOTE 1: The Testing Group specifies and approves which test cases are parts of the HbbTV Test Suite.

NOTE 2: Only Approved HbbTV Test Material shall be a part of the HbbTV Test Suite used for the Authorized Purpose.

NOTE 3: Upon approval of HbbTV, new HbbTV Test Material may be added from time to time at regular time-intervals.

NOTE 4: The HbbTV Test Suite does not include any physical equipment such as a Stream Player/Modulator, IP Server

Test Harness: The Test Harness is a system which orchestrates the selection and execution of Test Cases on the DUT, and the gathering of the Test Case Results for the Test Report.

Standard Test Equipment: The Standard Test Equipment is the collection of all “off the shelf” tools, which are needed to store, serve, generate, and play out the Test Cases on the DUT.

4.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AIT	Application Information Table
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CEA	Consumer Electronics Association
CICAM	Common Interface Conditional Access Module
CSS	Cascading Style Sheets
CSS-CII	Interface for Content Identification and other Information [30]

CSS-TS	Interface for Timeline Synchronisation [30]
CSS-WC	Interface for Wall Clock [30]
DAE	Declarative Application Environment
DLNA	Digital Living Network Alliance
DOM	Document Object Model
DRM	Digital Rights Management
DSM-CC	Digital Storage Media – Command and Control
DVB	Digital Video Broadcasting
DUT	Device under Test
EIT	Event Information Table
EIT p/f	EIT present/following
EPG	Electronic Program Guide
GIF	Graphics Interchange Format
HE-AAC	High-Efficiency Advanced Audio Coding
FQDN	Fully Qualified Domain Name
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDTV	Integrated Digital TV
IP	Internet Protocol
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MPEG	Motion Picture Experts Group
MSB	Most Significant Bit
OIPF	Open IPTV Forum
OpApp	Operator Application
PMT	Program Map Table
PNG	Portable Network Graphics
PVR	Personal Video Recorder
RCU	Remote Control Unit
RTP	Real-time Transport Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TV	Television
UI	User Interface
URL	Uniform Resource Locator
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

4.3 Conventions

MUST	This, or the terms "REQUIRED" or "SHALL", shall mean that the definition is an absolute requirement of the specification.
MUST NOT	This phrase, or the phrase "SHALL NOT", shall mean that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "RECOMMENDED", mean that there may be valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.
MAY	This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same manner an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

5 Test system

An HbbTV Test System consists of:

1) A suite of Approved Test Cases, obtained from and maintained by HbbTV Association,

2) A Test Environment used to execute the test cases and record the results of testing in a standard Test Report format. The Environment consists of:

    a. The Device Under Test (DUT)

    b. A Test Harness (TH), which acts as test operation coordinator and manages Test Cases and Execution Results

    c. Standard Test Equipment (STE) required for all Test Environments, as described in 5.2.1

    d. Optional Test Equipment (OTE) which may be used to simplify or allow automation of some Test System tasks, perhaps through proprietary terminal interfaces.

    e. RF and IP connections between the DUT and other elements of the Test Environment

    f. TLS test server used to provide a server-side environment for execution of some TLS related test cases. This is provided on behalf of the HbbTV Association and does not need to be supplied separately.

Figure 1 shows the interconnectivity between these required elements.

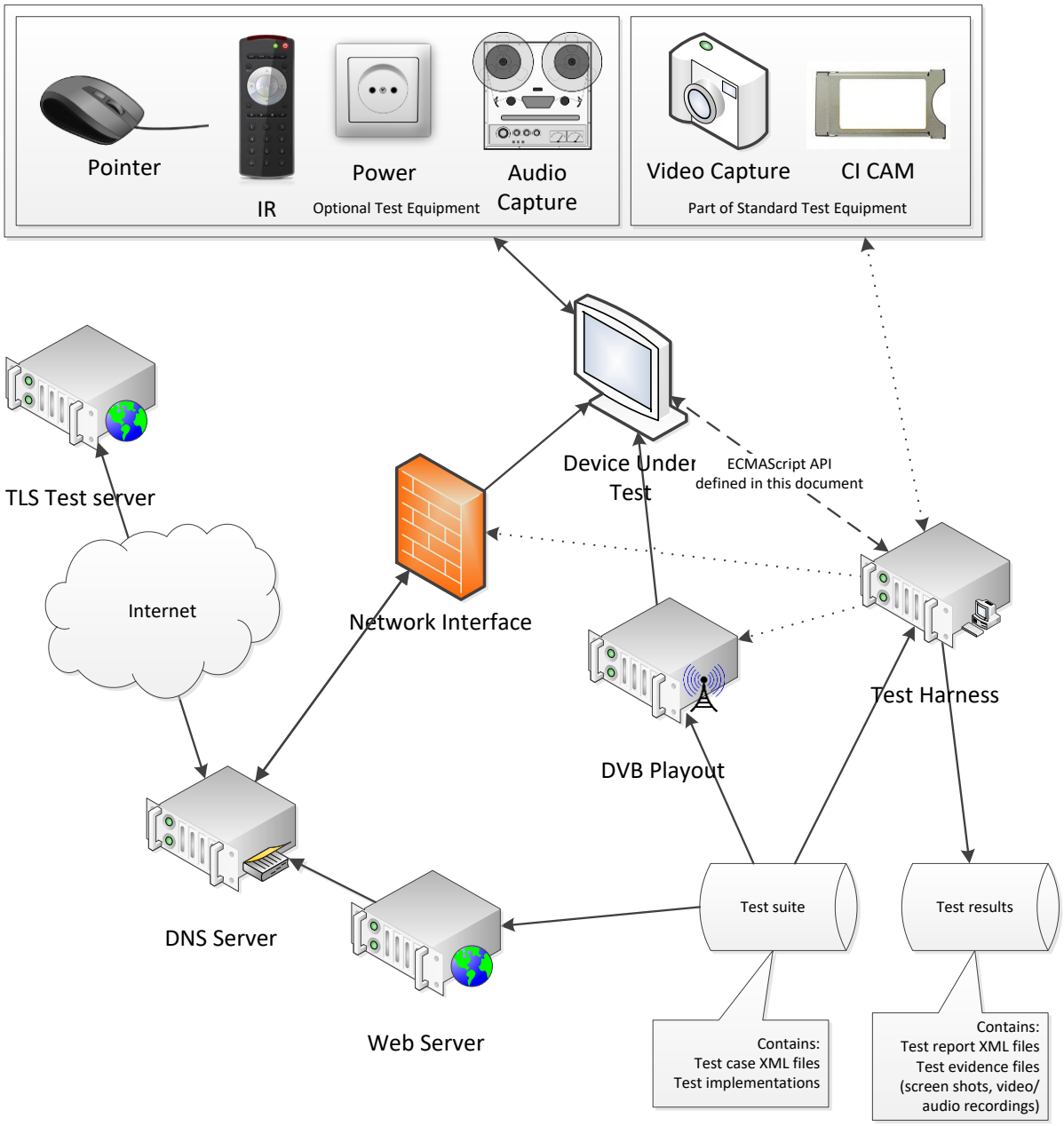


Figure 1: Outline of an example test system overview

5.1 Test Suite

The HbbTV Test Suite consists of a number of Test Cases. Each Test Case is uniquely identified and is intended to verify a specific requirement of the HbbTV specification. The Test Case consists of the following items, mostly stored in a single folder named for the Test Case identifier:

- Test Case XML. This is a multi-purpose document containing:
  - Specification references and applicability
  - Assertion text
  - Procedure and expected results
  - Test Case development history record

- Test implementation files. Where a number of Test Cases use the same files these may be in a shared resource folder.
- Configuration files required to execute the Test Case on a Test Harness.
- License and distribution information.

The Test Cases in a Test Suite can have one of two statuses:

- Approved Test Cases are approved by the HbbTV Testing Group and form part of the requirements for HbbTV compliance
- Additional Test Cases are not approved the HbbTV Testing Group for compliance. They are distributed because they may be of use to developers. Test Cases may be changed from Additional to Approved if they are valid for HbbTV requirements and have been successfully reviewed for approval by the Testing Group.

5.2 Test Environment

The Test Environment for executing the HbbTV Test Suite on a DUT consists of two major components:

- Standard Test Equipment: The Standard Test Equipment is the collection of all "off the shelf" tools, which are needed to store, serve, generate, and play out the Test Cases on the DUT. This includes web servers, and the DVB Playout System. The components of the Standard Test Environment are not included in the Test Suite delivery, but may be provided by commercially available test tools.
- Test Harness: The Test Harness is a system which orchestrates the selection and execution of Test Cases on the DUT, and the gathering of the Test Case Results for the Test Report.

The Test Harness:

1) Uses the information in the HbbTV test case XML description and in the Test Material to initiate the execution of all necessary steps prior to execution (e.g. final test stream generation).
2) initiates the execution of the Test Case on the DUT, causing changes in the environment during execution based on timing points defined in the Test Material and in response to API calls from the Test Case,
3) Collects the test results from the Test Case running on the DUT.

5.2.1 Standard Test Equipment

This chapter further lists the components and requirements for the Standard Test Equipment parts of the HbbTV Test Environment.

The implementation of these components is a proprietary decision. For example, each of these components may or may not be physically installed on a single piece of hardware for each DUT, they may or may not be installed on commonly accessible machine, or they may be virtualized. They may also be already integrated into commercially available tools.

NOTE 1: None of the components defined in this section are delivered with the HbbTV Test Suite.

NOTE 2: There may be more than one DVB Playout device and DUT attached to the Test Harness to allow concurrent testing of multiple devices. There may also be one or more devices (such as a WLAN access point, an Ethernet switch or an IP router) connected between the web server and the DUT. These are implementation decisions which are beyond the scope of this document.

5.2.1.1 Web Server

The Standard Test Equipment shall include an HTTP web server serving the Test Material to the DUT. The web server shall be able to negotiate CORS requests; see [28].

NOTE: There may be test cases in the future that will require the ability to control this operation.

The web server shall accept HTTP requests on port 80. The web server shall be set up in a way that allows reaching the complete test suite under the directory "/_TESTSUITE". For example, the "index.html" file of the test case 00000010 shall be reachable via the URL:

http://hbbtv1.test/_TESTSUITE/TESTS/00000010/index.html

The server shall not use port 81, connections to port 81 shall be refused. This port is used in test cases requiring a refused TCP/IP connection error.

The network connection between the web server and the DUT is controlled by the Test Harness as defined in the playout sets of the Test Cases executed, either directly or via instructions delivered to the operator, to allow disconnection of the network from the DUT, see section 7 for more details.

The web server shall provide support for both IPv4 and IPv6. The web server shall have both an IPv4 and IPv6 address on the network connection between the web server and the DUT. The DUT shall be able to obtain both an IPv4 (always) and IPv6 address on that network connection (if the DUT supports IPv6, which is designated as Optional Feature +IPV6, but the network configuration does not need to be directly driven by the Optional Feature). The web server shall be configured to accept and process incoming requests from clients (usually the DUT) that address either its IPv4 or IPv6 addresses (either directly, or after DNS name resolution).

5.2.1.1.1 PHP

The server shall include a valid installation of the PHP Hypertext Pre-processor to allow dynamic generation of content. Any version of PHP 5.2 or newer may be installed on the server, no special modules are required. Only files with the extension .php shall be processed by the PHP Hypertext Pre-processor.

The Test Harness shall set the PHP Hypertext Pre-processor's "default_mimetype" option to "application/vnd.hbbtv.xhtml+xml; charset=UTF-8".

NOTE: PHP scripts can control the MIME type used for their output, by using a command such as "header('Content-type: video/mp4');". If a PHP script does not specify a MIME type for its output, then PHP uses the MIME type specified by the "default_mimetype" option.

The Test Harness shall set the PHP Hypertext Pre-processor's "short_open_tag" option to "Off"

NOTE: This is so that the test cases can use PHP in combination with XML.

The Test Harness shall set the PHP Hypertext Pre-processor's "allow_url_fopen" option to "On".

NOTE: This is so that the test cases can use PHP to fetch json log files from the secure TLS server (specified in section 5.2.1.14 TLS test server) via file_get_contents() call.

NOTE: These PHP options are usually set in php.ini, but there are often web-server-specific ways to set them too, such as php_admin_value in Apache's mod_php.

Test authors shall ensure that content is served with cache control headers when needed. To set headers, a test will need to use a PHP script to serve the content.

EXAMPLE: when serving a dynamic MPD file for DASH streaming, the PHP script should set the "Cache-Control: no-cache" header.

The $_SERVER shall contain the variables required by the CGI/1.1 Specification [40].

NOTE: That does not necessarily mean that the PHP has to be launched by CGI. The mechanism used to run PHP code is a harness implementation detail.

For each request, PHP shall be allowed to use at least 160MB of memory. Normally this means that the PHP memory_limit option shall be 160M or greater.

5.2.1.1.2 File Extensions and MIME types

The file extensions to be served by the web server with their respective MIME types are defined in table 1:

Extension	MIME type
html	application/vnd.hbbtv.xhtml+xml; charset=UTF-8
html5	text/html; charset=UTF-8
txt	text/plain; charset=UTF-8
xml	text/xml; charset=UTF-8
cehtml	application/vnd.hbbtv.xhtml+xml; charset=UTF-8
js	application/x-javascript; charset=UTF-8
css	text/css; charset=UTF-8
aitx	application/vnd.dvb.ait+xml; charset=UTF-8
mp4, m4s	video/mp4
ts	video/mpeg
m4a, mp4a, aac	audio/mp4
mp3	audio/mpeg
bin	application/octet-stream
casd	application/vnd.oipf.contentaccessstreaming+xml; charset=UTF-8
cadd	application/vnd.oipf.contentaccessdownload+xml; charset=UTF-8
mpd	application/dash+xml; charset=UTF-8 [21]
xse	application/vnd.dvb.streamevent+xml; charset=UTF-8
png	image/png
jpg	image/jpeg
gif	image/gif
wav	audio/x-wav
tts	video/vnd.dlna.mpeg-tts
dcf	application/vnd.oma.drm.dcf
ttml	application/ttml+xml; charset=UTF-8
woff	application/font-woff
ttf, cff	application/font-stnt
sub	image/vnd.dvb.subtitle
map	application/octet-stream
pkg	application/vnd.hbbtv.opapp.pkg[2]
otf	application/font-sfnt

Table 1: Web server MIME types

Additionally[3], the harness web server shall serve files with any of the following filenames with the "image/png" Content-Type:

- dvb.icon.0001
- dvb.icon.0002
- dvb.icon.0004
- dvb.icon.0008
- dvb.icon.0010
- dvb.icon.0020
- dvb.icon.0040
- dvb.icon.0080
- dvb.icon.0100
- dvb.icon.0200
- dvb.icon.0400
- dvb.icon.0800

NOTE: These are application icons as defined in TS 102 809 [4] v1.3.1 section 5.2.8. The filenames are mandated by that specification. The OpApps spec notes that the bilateral agreement may require icons.

[2] Only required for harnesses that support Operator Application testing as defined in Appendix D.

[3] Only required for harnesses that support Operator Application testing as defined in Appendix D.

5.2.1.1.3 Handling fragmented MP4 file requests

The web server shall support special processing for files served from the test suite that end in the extension '.fmp4'.

If the Path section [22] of HTTP requests matches the POSIX basic regular expression[4] [i.5]:

```
^/_TESTSUITE/TESTS/\([a-z0-9][a-z0-9\-]*\(\.[a-z0-9][a-z0-9\-]*\)\{1,\}_[A-Z0-9][A-Z0-9_\-
]*\)/\(.*\.fmp4\)/\(.*\)$
```

And the first capture group:

```
[a-z0-9][a-z0-9\-]*\(\.[a-z0-9][a-z0-9\-]*\)\{1,\}_[A-Z0-9][A-Z0-9_\-]*
```

Matches the ID of a test case in the test suite, and the third capture group:

```
.*\.fmp4
```

Matches the path of a file in that test case's directory (referred to as the container file), then the fourth capture group:

```
.*
```

(Referred to as the segment path) shall be used to extract a subsection of the container file, as follows:

1) The web server shall look in the directory holding the container file for a file named 'seglist.xml'. If this file is not found the server shall return an HTTP response to the client with status 404. This file, if it exists, shall conform to the XSD in SCHEMAS/seglist.xsd. The contents of the seglist.xml file shall not vary during execution of a test. The server shall parse the seglist.xml file, and locate the 'file' element with a 'ref' attribute matching the segment path and a 'video' element matching the container file. If no such element is found the server shall return an HTTP response to the client with status 404. The server shall then read the number of bytes given by the 'size' element from the container file, starting at the offset given by the 'start' element (where an offset of 0 is the first byte in the file.) The HTTP response containing the data shall have a status of 200, and shall have the 'content-type' header [8] set to the value specified by the 'mimetype' element in the XML, or 'video/mp4' if that element is not present [23].

2) If an error occurs reading the file, the server shall return an HTTP response with status 500. If there is not enough data in the file to service the request, the server shall return an HTTP response with status 404.

5.2.1.1.4 HTTPS

The web server shall also serve HTTP over TLS (HTTPS). The domain name used for this can be chosen by the tester. The tester must acquire a TLS certificate and corresponding private key, and the certificate must be trusted by the terminal. For example, the tester might buy an Internet domain name, and acquire a corresponding TLS certificate from a Certificate Authority on the HbbTV Root Certificate list.

The HTTPS server shall support HTTP and TLS versions as follows:

- HTTP 1.1 shall be supported
- If testing a HbbTV 1.1.1 or 1.2.1 device, TLS 1.0, TLS 1.1, and TLS 1.2 shall all be supported
- If testing a HbbTV 1.3.1 or newer device, TLS 1.2 shall be supported
- If testing an OpApp device, TLS 1.3 shall NOT be supported, and HTTP 2 shall NOT be supported.[5]
- Except as prohibited above, the HTTPS server may support newer versions of TLS and HTTP.

---

[4] An equivalent Perl compatible regular expression (PCRE) is;

^/_TESTSUITE/TESTS/([a-z0-9][a-z0-9\-]*(\.[a-z0-9][a-z0-9\-]*)+_[A-Z0-9][A-Z0-9_\-]*)/(.*\.fmp4)/(.*)$

[5] The section of this test specification dealing with OpApp HTTPS client certificates was written before TLS 1.3 or HTTP 2 were standardized, and is not compatible with those protocols. A future version of this test specification may correct that.

The HTTPS server shall serve the same paths as the HTTP server, with the same content. All the rest of this section "5.2.1.1 Web Server" applies equally to the HTTP and HTTPS servers.

Test cases that use the harness HTTPS server must indicate that by either:

- including this tag in their implementation.xml file, directly inside the <testImplementation> element (NOT in the <opAppDiscovery> element):

```
<httpsServerConfig/>
```

- OR, for OpApp tests only, including either the <opAppDiscovery> or <runOpAppPage> tags
- OR, for ADB tests only, including the <adbAitDiscovery> tag. In this case the required "target" attribute of the <adbAitDiscovery> tag will determine the PHP script which will be run when trying to retrieve the path "/xml.aitx".

NOTE: If the test does not use any of those tags, then:

- The test harness does not need to start the HTTPS server, and
- If the harness starts the HTTPS server anyway, then that HTTPS server does not need to be correctly configured.

5.2.1.2 DNS Server

The Standard Test Equipment shall include a DNS (Domain Name System) server that can be accessed by the DUT over its network connection. The DUT shall be configured to use only the Standard Test Equipment, and not any other DNS server. This can be done by manual configuration of the DUT by the tester, or by automatic configuration via a specially configured DHCP, or by router / firewall configuration to direct DNS requests to the DNS server on the test harness server.

The DNS server shall both have both an IPv4 and IPv6 address on the network connection between the DNS server and the DUT. The DNS server shall be configured to accept and process incoming requests from clients that address either its IPv4 or IPv6 addresses. The DNS server shall be capable of processing both IPv4 (QTYPE: A) requests and IPv6 (QTYPE: AAAA) requests.

The DNS server shall resolve the following domain names to IPv4 addresses:

- hbbtv1.test, hbbtv2.test, hbbtv3.test, a.hbbtv1.test, b.hbbtv1.test, c.hbbtv1.test, shall be resolved to the IP address of the Standard Test Equipment web server.
- For test cases that indicate the use of the harness HTTPS server in their implementation.xml file, the domain name of the Standard Test Equipment HTTPS server shall be resolved to the IP address of that server.
- server-na.hbbtv1.test shall be resolved to an IP address that is not reachable
- dns-failure.hbbtv1.test shall not be resolved, the DNS server shall reply with NXDOMAIN error.
- External (on Internet) hbbtvtest.org domain and all its sub-domains must be reachable by the DUT and PHP interpreter via both HTTP (port 80) and HTTPS (port 443) protocols
- DUT must be able to access hbbtvtest.org domain via its IP address directly via the HTTP protocol over TLS
- DUT must be able to access domains specified in Annex C

The following domain names shall be resolved to the IP address of the Standard Test Equipment web server, using the specified version of the Internet Protocol (IP):

- ipv4.hbbtv1.test shall be resolvable exclusively to the IPv4 address
- ipv6.hbbtv1.test shall be resolvable exclusively to the IPv6 address
- dual-stack.hbbtv1.test shall be resolvable to both the IPv4 and IPv6 addresses. The DNS server shall include both an A record and an AAAA record for this domain name.

For the .test domain:

- if a lookup succeeds, the DNS server response shall indicate a 30 second TTL.
- if a lookup fails with NXDOMAIN, it shall indicate a 30 second negative-cache TTL.

DNS queries to resolve the Authoritative FQDN under '*.hbbtvdns.org' should resolve to the IP address of the harness HTTPS server with a 30 second TTL. The following special cases apply:

- '3bd613.a336.watermark.hbbtvdns.org' should return a DNS record with a TTL of 300 seconds (5 minutes).
- The subdomains '2.a336.watermark.hbbtvdns.org', '3bd6f0.a336.watermark.hbbtvdns.org', and '3bd6f1.a336.watermark.hbbtvdns.org' should return a name error (NE).
- The subdomains '3bd6f2.a336.watermark.hbbtvdns.org' and '3bd6f3.a336.watermark.hbbtvdns.org' shall be resolved to an IP address that is not reachable.

5.2.1.3 Network Interface

There shall be some mechanism by which the application layer throughput of the network interface connected to the DUT may be “throttled” to a defined maximum throughput. This shall be controlled by the Test Harness, either directly or via instructions delivered to the operator depending on the implementation of the Test Harness. By default, the maximum "nominal bitrate" (as that is defined in section 7.4.6) shall be 8Mbps, but this may be changed by the use of the setNetworkBandwidth() (see 7.4.6) function. At the beginning of each test case, the maximum "nominal bitrate" shall be returned to the default, regardless of any throttling that was applied during the previous test.

NOTE: This functionality may be integrated into a Test Harness; however this is outside of the scope of this document.

5.2.1.4 DVB Playout

The DVB Playout mechanism shall be controlled by the Test Harness, either directly or via instructions delivered to the operator. The transport stream data played out by the DVB Playout mechanism is created from the Playout Set definition of the test that is currently being executed on the DUT.

DVB Playout shall implement TDT/TOT adaptation such that the first TDT/TOT in the base stream is used unaltered the first time that the base stream is played out, and all subsequent TDT/TOTs (including in any repetitions of the base stream) shall be replaced such that the encoded time advances monotonically. For non-integrated playout solutions this means that a playout mechanism which supports TDT/TOT adaptation must be used, and this feature must be activated.

The DVB Playout mechanism shall support up to two DVB multiplexes.

The DVB multiplexes are typically transmitted using multiple modulators (one for each multiplex) and a RF mixer, but may also be transmitted using a single advanced modulator that supports generating multiple signals at once on different RF channels. This specification uses the phrase “modulator channel” to refer to either a modulator outputting a single multiplex, or one channel on an advanced modulator capable of generating multiple signals at once.

When the Test Harness has two modulator channels configured and the Test Harness attempts to execute a test case that specifies two DVB multiplexes, then:

- the first multiplex shall be delivered by the first configured modulator channel;
- the second multiplex shall be delivered by the second configured modulator channel.

When the Test Harness has two modulator channels configured and the Test Harness attempts to execute a test case that specifies a single DVB multiplex, then:

- the multiplex shall be delivered using the first modulator channel configured in the Test Harness;

the second modulator channel shall completely stop outputting a signal. Any two supported modulator settings may be configured in the Test Harness – there is no restriction on, for example, configuring a DVB-S modulator and a DVB-T modulator.

5.2.1.5 Image Capture

When test cases call either of the API calls analyzeScreenPixel() (7.3.3) or analyzeScreenExtended (7.3.4), an image must be taken of the DUT display. Each image taken shall be referenced in the relevant <test case id>.result.xml file in the testStepData element as defined in 9.1.3.4. These images shall be stored within the Test Report as defined in section 9.

The mechanism for this image capture is proprietary and outside of the scope of this specification. It may be automated by the Test Harness, or may be implemented manually by testers using image capture technology, webcam, digital camera or by some other capture mechanism. The format of the image stored is defined in 9.1.3.4.

5.2.1.6 ECMAScript Environment

The standard test equipment shall include an ECMAScript application environment in which tests can be run directly by the Test Harness (referred to as 'harness-based tests.') The file containing the code to be run is signalled by the implementation.xml file for the current test. The application environment shall support ECMAScript version 5.1 [33], and shall provide an implementation of the test API for the test to use. It is not required that the ECMAScript environment implement the DAE or DOM.

This ECMAScript environment shall also provide the XMLHttpRequest [39] API, with the following modifications:

- Support for Document response type is optional.
- The request URL passed to this API must be an absolute URL.
- All requests are considered to be "same origin".
- It is unspecified what, if any, Referer header is sent.

Note: The referenced version of the XMLHttpRequest specification, and the first modification listed above, are the same as the APIs defined in the OIPF Web Standards TV Profile [38] section A.3.1.

The file containing the application or the harness-based test shall be defined using the harnessTestCode element of the implementation.xml file. If a server side test is defined then the transport stream may include an application, but the application shall not instantiate the test API object or make calls to the test API.

Harness-based test applications shall be defined as an ECMAScript application, conforming to the ECMAScript profile defined above. The application shall be contained in a text file encoded in UTF-8. The application environment shall include the test API object as a built-in constructor, which may be instantiated by applications in order to communicate with the test harness. All API calls shall behave as defined for running in the DAE.

When the test harness executes a test with the harnessTestCode element defined, it shall follow the following steps:

1) Commence play out of playoutset 1, if defined
2) Execute application defined in harnessTestCode element

The Test Harness may terminate the test at any time without notifying the test application.

5.2.1.7 CSPG-CI+ Test Module

Several test cases require a custom CI+ CAM ("CSPG-CI+ module") which acts like a standard Content and Service Protection Gateway (CSPG-CI+) module but additionally has an interface with the test harness allowing tests to control certain functions of the module (such as descrambling), and to query the module's status at various points (such as whether the DUT is forwarding content to the CAM for descrambling).This section describes the custom CSPG-CI+ test module functions that need to be implemented (in the CAM) to enable the tests, and other requirements on the test framework.

NOTE: This section focuses on additional functions required of the test CSPG-CI+ module over a normal CSPG-CI+ module.

The CSPG-CI+ module will have to operate with test certificates (not production certificates) because it implements custom test behaviour. This implies that DUTs will need to be configured with test certificates also, during testing.

5.2.1.7.1 Communication between test harness and CSPG-CI+ module

5.2.1.7.1.1 Requirement

Several CI+-related tests require a mechanism for the test harness to:

- configure CAM behaviour (for example put it into a mode where it will respond in a certain way to future DUT/CAM interactions)
- Trigger the CSPG-CI+ CAM to perform an action now (for example send a message informing the DUT of a parental rating change event).

In addition the CSPG-CI+ test CAM needs a mechanism to:

- Indicate to the test harness that, according to its configuration, the host has not behaved as expected/required and hence the test has failed.

5.2.1.7.1.2 Configuration/action required

The following CAM functions need to be controlled by the Test Harness:

1) Do/do not perform CSPG-CI+ discovery on next CAM insertion (i.e. switch between CSPG-CI+ and standard CI+ modes) (accept session establishment to the CI+ SAS resource and OIPF private application)
2) Expect certain DRM messages from the host and, when received, respond to each with a specific configured response
3) Assert that certain DRM messages were indeed received (in a specific order) by the CAM (since some configurable recent time/event)
4) Stop/start descrambling and send a rights_info message to the host
5) Stop/start descrambling and send a parental_control_info message to the host
6) Assert content is/is not currently being received from the host for descrambling
7) Respond to OIPF private message with configured response
8) Set URI and confirm delivery to DUT

5.2.1.7.2 CICAM functionality

The combination of CICAM and harness used with these test APIs shall meet the following requirements, in addition to the requirements implied by the functions specified in 7.4.7.

1) Correct implementation of CI plus [14] and 4.2.3 of [29], except where behaviour needs to be modified to implement other functionality described in this document.
2) Able to decode the CA system as specified in the next Section.

NOTE 1: These requirements apply to the combination of test harness implementation and CICAM, this document does not specify which component within the harness implements the requirements in this document.

NOTE 2: The CICAM may be equipped with either development or production certificates.

5.2.1.7.3 CA System

The CICAM shall support descrambling of content encoded using DVB CSS when the CA_system_ID signalled in the CAT has a value of 4096, 4097 or 4098.

NOTE: The DVB CSA specification is only available to licensees. Information on licensing it is available at [i.7]. Non-licensees may find the Wikipedia page [i.8] helpful.

To descramble a selected service, the CAM shall parse the CA_descriptor from the CAPMT and parse the PID identified by the CA_PID. These PIDs identify the ECMs for the selected service. The format of the ECM (excluding transport stream packet and section headers and stuffing bytes) is shown in the table below:

Syntax	No. of bits	Mnemonic
control_word_indicator	8	bslbf
Reserved	16	
encrypted_control_word	64	bslbf
Reserved	40	

Table 2: CA System ECM format

control_word_indicator – 0x80 indicates that encrypted_control_word is the 'odd' key, 0x81 indicates that encrypted_control_word is the 'even' key

encrypted_control_word – value to be used as either the odd or even control word, 'encrypted' by being XORed with the value 0xAB CD EF 01 02 03 04 05.

5.2.1.7.4 Extensions to Implementation XML schema to support configuration of CICAM state

The harness shall have the capability to present to tests a CICAM in each of the following three states:

1) CI+ CAM with OIPF application conforming to 4.2.3 of [29] profiled as in 11.4.1 [1]

2) CI+ CAM without OIPF application

3) CI CAM (i.e. CAM which does not implement CI+)

A harness may do this either using multiple CAMs or by using a single reconfigurable CAM (or by any other suitable mechanism.)

The test's requirements (if any) for use of a CAM shall be declared using an extension to the existing implementation XML syntax (see below).

If the XML element is not present then the test is assumed not to require the use of a CAM and test cases shall not use the functionality described in this section.

5.2.1.7.5 Implementation XML extension

A 'CAM' element is defined, with an optional 'type' attribute. If the element is absent then the harness may assume that no CAM is required by the test. In this case the behaviour of the JS function calls defined in 7.4.7 is undefined.

The 'type' attribute shall have one of the following values:

- cspgcip (default)
- cip
- ci
- none

Where the meanings of the values are as follows:

- cspgcip – the CICAM shall behave as described in Section 5.2.1.7.2 above.
- cip – The CAM shall respond to SAS_connect_rqst APDUs specifying the OIPF application ID (4.2.3.4.1.1 of [29]) with session_status 0x01 (Connection denied - no associated vendor-specific Card application found)

- ci – The CAM shall be configured such then whenever the CAM is powered up or inserted the CAM behaves as a CI, rather than CI plus, device for the purposes of host shunning behaviour (§10 in [14].) There are multiple mechanisms by which this may be achieved, including removal of the 'ciplus' compatibility indicator from the CIS information string, disabling the CA resource, etc.

- none – The test requires that no CAM is inserted in the terminal at the start of the test. This is distinct from the case where the test is not concerned with the presence or absence of a CAM (in which case the 'CAM' element should be omitted altogether.) See also note below.

An example of the XML syntax would be:

```
<?xml version="1.0" encoding="utf-8"?>
<testImplementation id="com.example_00000000"
                    xmlns="http://www.hbbtv.org/2012/testImplementation">

    <playoutSets>
        <playoutSet id="1" definition="playoutset.xml"/>
    </playoutSets>
    <CAM type="cip" />
</testImplementation>
```

NOTE: If a test case requires a CAM to be inserted after the start of the test then the implementation XML shall specify a required CAM configuration using this syntax, and the test shall then request removal and eventual reinsertion of the CAM by calling the 'manualAction' JS-Function (or equivalent) once test execution has commenced. (If the CAM type is 'none' and CAM insertion is requested then the CAM configuration would be indeterminate.)

5.2.1.8 Companion Screen

Several test cases (mainly the tests of the Launcher Application) require a Companion Screen (CS) to be connected to the test network (i.e. on the same network as the DUT, and so that it uses the DNS server provided by the Test Harness). If a Companion Screen is required to run the test this will be indicated via the presence of a 'companionScreen' element in the test's implementation.xml file. If the element is absent then the harness may assume that no Companion Screen is required by the test.

A 'companionScreen' element is defined, with one mandatory 'type' attribute and an optional 'initiallyConnected' attribute.

The 'type' attribute indicates the type of the Companion Screen required to run the test. It shall have one of the following values:

- any – a Companion Screen compatible with the DUT is connected to the test network. It does not matter if the Companion Screen is implemented on iOS, Android, or some other OS,

- iOS - an iOS Companion Screen compatible with the DUT is connected to the test network, or

- Android - an Android Companion Screen compatible with the DUT is connected to the test network.

The "initiallyConnected" attribute indicates if the Companion Screen should be connected to the DUT at the start of the test. It shall have one of the following values:

- true (default) - the Companion Screen shall be connected to the DUT at the start of the test, or

- false - the Companion Screen shall not be connected to the DUT at the start of the test; instead the test will instruct the tester when to connect the Companion Screen.

An example of the XML syntax for the element would be:

```
<companionScreen type="iOS" initiallyConnected="true" />
```

5.2.1.8.1 Companion Screen Launcher Application

Support for launching a CS application from an HbbTV application as described in the HbbTV Specification 14.3 and 14.4is defined in that specification as an conditionally mandatory feature. If the manufacturer is going to support it, they must develop a launcher application that can run on the targeted companion screen platform.

There is no requirement on the manufacturer to provide or develop any companion application beyond the launcher application to execute any of the Companion Screen test cases.

The launcher application is developed by the manufacturer and is part of the system under test. It is not possible for it to be provided as part of the test harness or test environment because the interface between the launcher application running on the companion screen and the device is proprietary to the device manufacturer, as described in section 14.3 of the HbbTV Specification. Therefore it does not make sense to consider testing this feature of an HbbTV device in isolation from the companion application(s) developed with it.

Note that if this feature is supported, then the TV/STB manufacturer should make the launcher application available to all customers who have purchased the HbbTV TV or STB, so that they can use this application launching feature. E.g. by making it available in the Apple App Store / Google Play app store, and by documenting it in the TV/STB's user manual. It's not just for testing!

Those Companion Screen test cases that relate to the terminal launching an application on a companion screen have a CS_APP_LAUNCH optional feature precondition. Where they launch an application, they launch a specific widely-available application, or one usually pre-installed with the OS (e.g., Google Maps on Android devices). The test application running on the terminal is provided in the test case and discovers the CS launcher with HbbTVCSManager.launchCSApp() and invokes HbbTVCSManager.launchCSApp() to launch the application.

5.2.1.9 Checking media synchronization on a single device

Some of the test cases require precise measurement of the timing of video flashes and audio tones. This is used when the DUT is playing media that should be synchronized. This is the case where the media playback is on a single device, there is no inter-device synchronization involved.

The timing of these flashes and tones shall be measured in the physical domain i.e. as light and sound. However, if the DUT has an analogue audio output that is intended for use with normal headphones (where the headphones are expected to add no delay to the audio signal), then it is acceptable to use that output to detect the audio tones.

If the DUT does not have an integrated display, then the tester should select a suitable display, following any instructions provided by the DUT manufacturer. (E.g. if the DUT's instructions say that, for optimum performance, it should be used with a HDMI 2.0 compliant display, then the tester should ensure that they choose a HDMI 2.0 compliant display. Or if the DUT's instructions include a list of tested TV models then the tester should choose a TV from that list).

The Test Harness shall include sensors to detect and time light emissions from two different places on the display, and to detect and time audio emissions. These sensors shall be synchronised together and each sensor shall have an accuracy of +/-1ms.

NOTE: There are several ways the test harness might accomplish this, including:

- a dedicated device with light sensors and audio inputs
- a high-speed camera set to 1000Hz and pointed at the TV with a genlocked audio recorder, and then manual or automatic analysis of the recordings
- a person looking at a storage oscilloscope with light sensors and audio inputs connected.

There are 2 different requirements:

1) Confirm that a series of tones and flashes across a 15 second period are synchronised (within a tolerance specified by the test case, typically 10ms or 20ms). In this case, it is acceptable to check exact synchronisation (e.g. using an oscilloscope) for only some of those tones and flashes, including at least one near the start of the 15sec period and one near the end of that 15sec period. The rest of the tones and flashes can be checked by an unaided human. (Rationale: if the first and last flashes are synchronized, then the intermediate flashes are likely to either be synchronised or to be wrong by 0.5sec or more).

2) Confirm that the difference between the start times of 2 flashes is a given number of milliseconds, within a tolerance specified by the test case. This is used where the test case cannot precisely synchronize the flashes but it can precisely measure what the offset between the flashes should be.

5.2.1.10 Checking media synchronization when DUT is inter-device synchronisation master

NOTE 1: HbbTV [1] includes protocols for media synchronization between multiple devices [34]. An HbbTV terminal can be put into "master" mode, where it exposes 3 services:

- CSS-WC - Wall Clock. This is a simple UDP-based protocol that allows the Companion Screen to read a clock that ticks at "wall clock" rate. In so doing, the client (the Companion Screen) can measure and compensate for the network latency. It is a bit like NTP, but much simpler.
- CSS-TS - Timeline Synchronisation. This is a simple Websocket-based protocol that allows a Companion Screen to be told the mapping between the CSS-WC wall clock time and one of the timelines in the media that is currently playing on the HbbTV terminal. A Companion Screen can use this information to play out media that is synchronised to the media shown by the HbbTV terminal, e.g. an audio track in a different language, or a video at a different camera angle.
- CSS-CII - Content Identification and other Information. This is a simple Websocket-based protocol that allows a Companion Screen to be told the media that is currently playing on the HbbTV terminal, the timelines that are available, and the URLs for the CSS-TS and CSS-WC protocols.

NOTE 2: The Test Harness includes a partial implementation of the HbbTV inter-device media synchronisation slave client. The part that is needed is specified in the normative parts of this section and section 7.8.3.

The Test Harness shall include clients for the CSS-WC and CSS-TS services [34]. These clients shall be linked together and linked into the light and audio sensors described in the previous section, so that the DUT's emission of light and audio can be timed against the timeline being reported by the DUT.

The APIs for this are defined in section 7.8.3.

NOTE 3: There are several ways the test harness might accomplish this, including:

- Having a dedicated device with light sensors and audio inputs that synchronises (in some proprietary way) to a CSS-WC and CSS-TS client on a PC. (Note: BBC R&D have an example of this).
- Having a dedicated device with light sensors and audio inputs that also implements the CSS-WC client, and communicates with a CSS-TS client on a PC
- Having a dedicated device with light sensors and audio inputs that also implements the CSS-WC and CSS-TS clients
- Having software implementations of CSS-WC and CSS-TS on a PC, which synchronise with a high speed camera and genlocked audio recorder
- Having software implementations of CSS-WC and CSS-TS on a PC, which flashes the PC screen in a way that is synchronised to the timeline. That flashing is detected by an extra light sensor, which is connected to a storage oscilloscope. The light sensors and audio inputs from the DUT are connected to the same oscilloscope.

NOTE 4: Since the CSS-CII service is based on Websockets and is not timing-critical, it can be tested by having a test application connect to it via Websockets using the API defined in section 7.7. There is no need for explicit harness support for testing CSS-CII. The CSS-TS service is also based on Websockets, and is tested using a combination of Websocket-based testing and the Test Harness's client described above.

5.2.1.11 Checking media synchronization when DUT is inter-device synchronisation slave

NOTE 1: A HbbTV terminal that supports the +SYNC_SLAVE option can be put into “slave” mode, where it connects to another “master” terminal using the 3 protocols discussed in the previous section.

NOTE 2: The Test Harness includes a partial implementation of the HbbTV inter-device media synchronisation master server. The part that is needed is specified in the normative parts of this section and section 7.8.4.

The Test Harness shall include servers for the CSS-WC, CSS-TS and CSS-CII protocols [34], with configurability of those servers provided via the Test API function startFakeSyncMaster(). The reported timeline will just advance at a fixed rate.

The Test Harness will also provide a server accessible via a Websocket connection that can be used to retrieve the current timeline time, at the point the request is received from the DUT.

NOTE 3: This uses a Websocket so that the test can ensure the slow set-up of the TCP/IP connection is done in advance. It is expected that once the Websocket connection is set up, sending a Websocket message from the DUT to the Test Harness will be quick. The reply may be slower due to the behaviour of the DUT’s task scheduler, or because the DUT happens to be busy processing another JavaScript event at the time the reply is received, but the speed of the reply doesn’t affect the accuracy of the test.

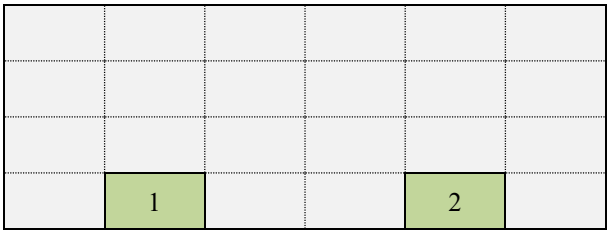
For the tests currently implemented, there is no reason for this to be linked to the light and audio sensors described previously.

The APIs for this are defined in section 7.8.4.

5.2.1.12 Light Sensor positioning for media synchronization tests

As described in section 5.2.1.9, the Test Harness will have sensor(s) that detect flashes of light from 2 different locations. The position of the two detection areas is defined in this section. These detection areas are referred to in this API specification as "light sensor 1" and "light sensor 2". Each of the two defined detection areas may be used for video, subtitles or UI graphics.

The positions of these detection areas are measured on the display that emits light, i.e. after all scaling has been applied. The positions are shown in the following diagram, which shows the whole TV screen (i.e. the entire panel excluding bezel):



(The dashed grid lines are spaced equally, to make it possible to read the position off this diagram).

In numbers, if (0, 0) is the top left of the display and (1, 1) the bottom right, then the co-ordinates of the two defined detection areas are:

- light sensor 1: (1/6, 3/4) to (1/3, 1)
- light sensor 2: (2/3, 3/4) to (5/6, 1)

Test cases using the light sensors must ensure that these detection areas are usually fully black, and a "flash" turns the entire detection area white, then black again. The detection area should be white for approximately 120ms. For measuring synchronization using the analyzeAvSync and analyzeAvNetSync APIs, the time of the "flash" is the mid-point of the period the frame is white. For measuring synchronization with the analyzeStartVideoGraphicsSync API, the time of the "flash" is the instant it changes from black to white.

The Test Harness may choose to monitor any point or area inside the detection area. It does not have to check the entire detection area.

Test cases may display anything they like outside of these detection areas. (E.g.: countdown timer; debug messages; etc).

The Test Harness shall ensure that the display outside the detection area, and ambient or external light sources, do not interfere with its sensors. (For example, by placing the sensors in the middle of the detection area and ensuring the sensitivity is set correctly, or shrouding them with black tape).

For health and safety reasons, test cases using this feature with a single light sensor should not have more than 3 flashes per second, and test cases using this feature with both light sensors at once should not have more than 1 flash per second. Note that some of the APIs defined here impose lower limits. (Rationale: it is desirable to not exceed the photosensitive epilepsy threshold of 3 flashes per second from [i.9]. It is predictable that test cases using two light sensors will sometimes fail due to the content not being synchronised, and in that case the tester will see the two flashes separately, hence the lower limit for such tests. In practise, these limits are not expected to cause any problems for test authors, and the sequence suggested in section 5.2.1.13 is well below these limits).

5.2.1.13 Media requirements for media synchronization tests

The video(s) and/or subtitles must contain a rectangular region that is normally filled with black pixels, but which flashes white at the relevant time(s) (see later). This region is called the “flash area”. This region should be sized and positioned so that, after the HbbTV terminal has applied any applicable scaling, (and, if applicable, the separate display has applied any scaling) it completely covers one of the two detection areas defined in sections 5.2.1.12.

When designing their media and test case, Test Case authors must make allowance for overscan. If the test case is positioning video on a 1280x720 display, then allowing for overscan can be done as follows. First, calculate the detection area positions if there is no overscan:

	Left	Top	Right	Bottom
Screen size	0	0	1280	720
Light Sensor 1	213	540	427	720
Light Sensor 2	853	540	1067	720

Then assume the largest overscan possible (20% horizontal overscan and 10% vertical overscan), and calculate the detection area positions in that case:

	Left	Top	Right	Bottom
Safe area	128	36	1152	684
Light Sensor 1	298	522	470	684
Light Sensor 2	810	522	982	684

Then take the min/max of those two tables to get a flash area which is guaranteed to cover the detection area regardless of the amount of overscan:

	Left	Top	Right	Bottom
Light Sensor 1	213	522	470	720
Light Sensor 2	810	522	1067	720

The audio must be silent, except for short -6dB Full Scale bursts of tone between 1 and 5 kHz. For measuring synchronization, the time of the tone burst is the mid-point of the period the tone burst is audible.

The flashes and tone bursts shall be synchronised.

The duration of each flash and tone should be short, but not so short that it is missed. It is recommended that, for 50Hz terminals, the flash duration is 120ms. This is chosen to be 3 frames of 25fps progressive video, or 3 full frames (6 fields) of interlaced video. It is 6 frames of 50fps progressive video. Choosing a constant duration, instead of a constant number of frames, allows tests to mix 50p and 25p video, and also means that the same audio track can be shared by test cases with different video framerates.

For some tests, there will be a single flash and a single, synchronised tone burst.

For other tests, there will be a repeating pattern of flashes and synchronised tone bursts. When choosing the time between consecutive flashes or consecutive tone bursts, the test case author must consider the analysis criteria documented for the analyzeAvSync() or analyzeAvNetSync() API as appropriate.

NOTE: One pattern that is suitable for these APIs is as follows:

For this pattern, the flashes and synchronised tone bursts should be 120ms in length and start at the following times, where time T is the start of the pattern (which may not be the start of the media):

Flash/tone start time	Time between flash/tones
T + 0 sec	
	1 sec
T + 1 sec	
	1 sec
T + 2 sec	
	2 sec
T + 4 sec	
	2 sec
T + 6 sec	
	3 sec
T + 9 sec	
	2 sec

The sequence then repeats with T being 11 seconds larger.

The irregular intervals are chosen so that medium-sized offsets between audio and video can be detected. Seeing 3 consecutive synchronised flashes/tones tells you that the audio and video are synchronised, and that check is guaranteed to detect synchronisation errors of less than 11 seconds. It will detect any synchronisation error that is not a multiple of 11 seconds long, which makes a “false pass” error very unlikely.

5.2.1.14 Internet-based TLS test server

A certain number of TLS test cases require access to low level details of the communication between DUT and remote server when HTTP protocol is used with TLS. The special server used to obtain low level details of the server-client communication is based on a mechanism similar to JSONP [i.10] and its response shall be JavaScript code that has to be evaluated later on.

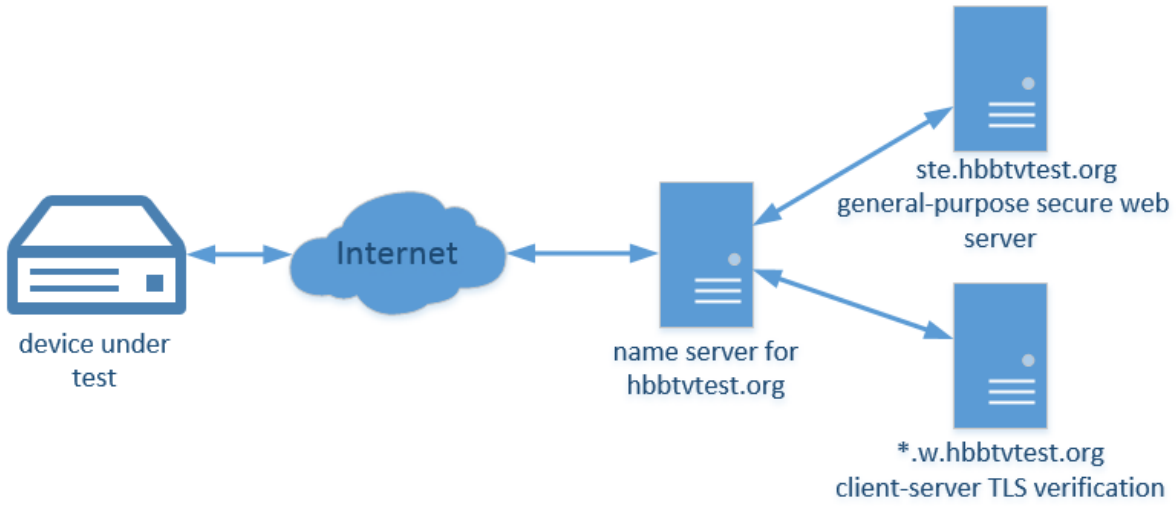
A separate server is used for test cases that require that the HbbTV application or some other resource be loaded from a https URI, hosted on a server with a valid certificate.

Because of this there are two differently configured virtual servers:

- Server configured for verifying TLS client-server communication
- General-purpose secure web server

Name server for hbbtvtest.org contains multiple NS records, that are sub-domain specific. Requests related to TLS client-server communication will target specific subdomains and the name server will return an IPv4 address of the virtual server configured for verifying TLS client-server communication. For requests targeting the subdomain ste.hbbtvtest.org (where ‘ste’ stands for standard test environment), name server shall return an

IPv4 address of the general-purpose secure web server that can serve static resources over HTTPS. This environment is illustrated on the image bellow.



5.2.1.14.1 Server configured for TLS client-server communication verification

This server runs dedicated web server application based on OpenSSL library that can parse a client's TLS handshake request and verify its content. Server application can also perform an invalid handshake or present an invalid certificate in order to validate the client's behaviour in these kind of scenarios. The domain used for testing purposes is hbbtvtest.org and it provides dedicated subdomains for each use case:

- *.w.hbbtvtest.org - used for verifying parts of the TLS handshake, presents a valid wild-card certificate
- a.hbbtvtest.org - used in for creating an exact match on certificate
- *.host-mismatch.hbbtvtest.org - used for creating a host mismatch
- *.expired.hbbtvtest.org - used for presenting an expired certificate
- *.sha1.hbbtvtest.org - used for presenting a SHA1 certificate
- *.1024.hbbtvtest.org - used for presenting a certificate that has an RSA key with only 1024 bits
- access via IP - used for testing Subject Alternative Name
- http2.hbbtvtest.org - server that supports HTTP/2 but not HTTP/1.1

URLs targeting wildcard subdomain on this server shall be in the following format:

https://LogId-TestId.w.hbbtvtest.org.

LogId part is used as a name of the file in which communication log is stored as a JSON (described in **Annex F)**. TestId corresponds to ID of the test case, and server uses it to set up a proper communication context (i.e. to determine which certificate to use or how to handle TLS handshake).

All subdomains used for TLS client-server verification resolve to the same IPv4 address.

Communication log contains low-level details of the communication between server and client, and depending if communication is established or rejected, it is delivered to client in one of two ways:

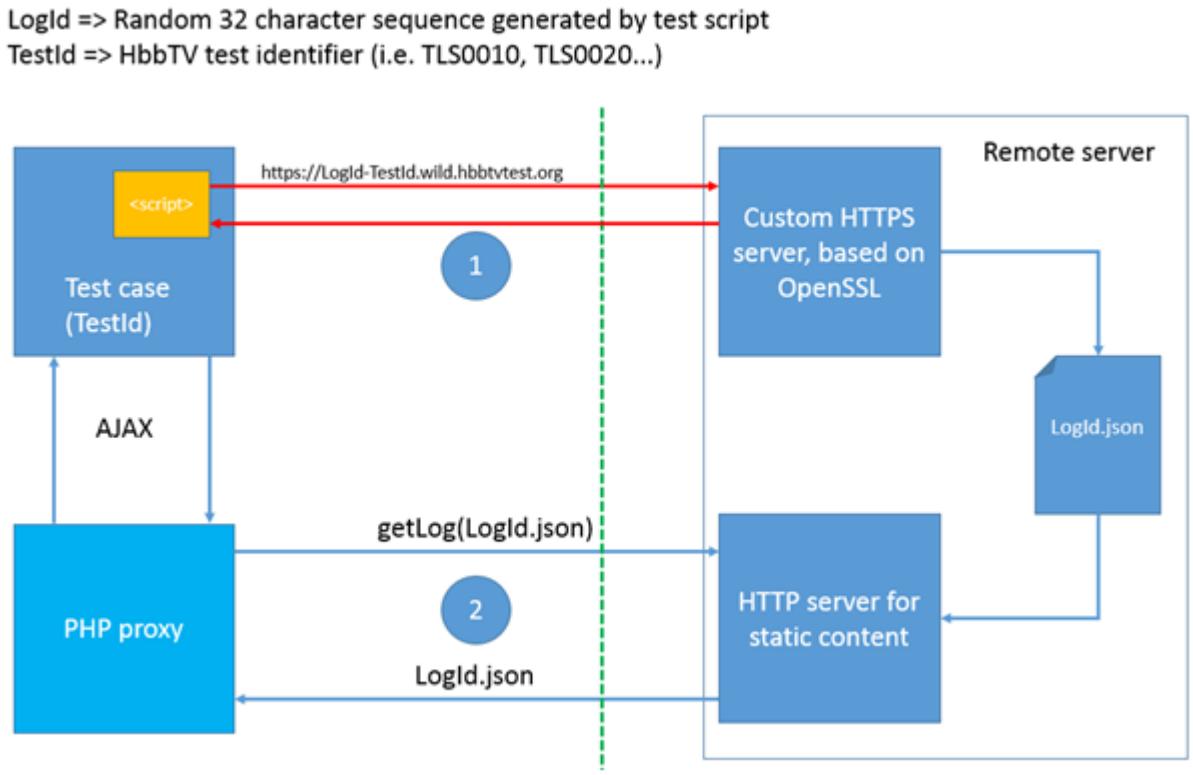
1. If communication is established, script delivers information to test page using mechanism similar to JSONP, where a 'src' attribute is dynamically set to an URL targeting the TLS test server via HTTPS. After the script is successfully loaded and evaluated, test script can use its information to determine the verdict.

2. If communication has failed (script was not loaded), test script uses PHP proxy to get contents of the communication log from remote server via HTTP, using known LogId (generated by test itself, a random 32 character sequence).

Communication logs older than certain time (default will be 60 seconds) are automatically deleted from server. Test case evaluates provided information and determines outcome of the test.

There are test cases which don't require detailed information about the client-server handshake, where test case outcome can be determined by evaluating if script tag was successfully loaded or not.

Block diagram of the system which illustrates both approaches is shown in image provided below.



5.2.1.14.2 General-purpose secure web server

For test cases where an application has to be loaded from an https:URI, or any other resource fetched via https:URI, this web server provides an environment similar to that of a regular Standard Test Equipment HTTP web server, described in section 5.2.1.1. The domain used for testing purposes is ste.hbbtvtest.org.

The web server shall accept incoming HTTP requests on port 80 and HTTPS requests on port 443.

The test suite includes directories matching the pattern:

```
TESTS/<test ID>/on_ste_hbbtvtest_org/v<version number>
```

In this pattern <test ID> is a test case ID and <version number> is an integer version number that starts at 1 for each test case. The web server shall be set up in a way that allows reaching files in those directories under the directory "/_TESTSUITE". For example, the "EME0010.html5" file of the test case org.hbbtv_EME0010 shall be reachable via the following URLs:

```
http://ste.hbbtvtest.org/_TESTSUITE/TESTS/org.hbbtv_EME0010/on_ste_hbbtvtest_org/v1/EME0010.html5
```

```
https://ste.hbbtvtest.org/_TESTSUITE/TESTS/org.hbbtv_EME0010/on_ste_hbbtvtest_org/v1/EME0010.html5
```

The first time a Test Case that uses this mechanism is included in the HbbTV Test Suite, the <version number> will be 1. When HbbTV has released a HbbTV Test Suite containing a file matching the pattern "TESTS/<test ID>/on_ste_hbbtvtest_org/v<version number>/", that file will be included unmodified in all future HbbTV Test Suite releases. (This means that the server will continue to work for people running older versions of the HbbTV Test Suite). If HbbTV needs to make a change to such a file, it will be copied to a new "v<version number>" directory using the next higher version number (i.e. 2 for the first time a change is needed, etc).

The server shall include a valid installation of the PHP Hypertext Pre-processor to allow dynamic generation of content. Only files with the extension '.php' shall be processed by the PHP Hypertext Pre-processor.

NOTE: HTML5 file on the secure server shall not include testsuite.js file due to unsecured mixed content. The js file may be blocked by the browser.

5.2.1.15 Network analysis tool

For some tests, it is necessary to analyze network activity at the low level in order to check whether expected network issues are happening and if DUT reacts to them as expected.

To make that kind of analysis feasible, test environment must have the Network analysis tool as a part of Standard Test Equipment.

Tests communicate with the Network analysis tool using JS-Function analyzeNetworkLog (see 7.9.1).

Features of the Network analysis tool:

- it has access to the network traffic between DUT and network interface (see Figure 1 in: 5 Test system)
- it is able to record network traffic into the human-readable log file on demand

Some of examples of the off-the-shelf network analysis tools are Wireshark [i.11] and Netcat [i.12].

5.2.2 Test Harness

This chapter further lists the components and requirements for the Test Harness parts of the HbbTV Test Environment.

The implementation of these components is a proprietary decision. For example, each of these components may or may not be physically installed on a single piece of hardware for each DUT, they may or may not be installed on commonly accessible machine, or they may be virtualized. They may also be already integrated into commercially available tools.

5.2.2.1 Test Harness Requirements

The Test Harness is a system which orchestrates the selection and execution of Test Cases on the DUT, and the gathering of the Test Results for the Test Report.

The Test Harness shall implement a mechanism for generation of broadcast transport streams specified by the HbbTV Playout Set as defined in section 7.4.3 Playout set definition . This mechanism may be offline (pre-generation of transport streams) or real-time (multiplexing on-the-fly). This is an implementation decision, and is outside the scope of this specification.

The Test Harness shall implement the Test API as defined in section 7 for communication with test applications.

The Test Harness shall store all reportStepResult, analyze[] (e.g. such as analyzeScreenExtended, analyzeScreenPixel etc.) and endTest calls received from the test application(s) during execution. The Test Harness shall use these calls to determine the result of the test case according to the requirements set out in section 6.4, and shall store all of this information in the result xml format defined in section 9.

5.2.2.1.1 OpenCaster Glue Code

The HbbTV Test Suite includes a python script known as the 'OpenCaster Glue Code' which can be used for pre-generation of the transport streams required to execute the test suite. OpenCaster is based on HbbTV specification v1.1.1 and there are some cases where it is not able to produce streams compatible with the requirements of this specification. In particular:

- Where streams are required to be synchronised with the HTTP server time
- Where streams require the application_recording descriptor to be present.

The OpenCaster Glue Code is a python script released under the Creative Commons license CC BY-SA. It's based on the free toolset called OpenCaster from the company Avalpa (see references below). It reads the information from the Test Case definition XML files of the Test Material and creates a transport stream which later can be played out by the Test Harness and the playout equipment used.

OpenCaster is a free and open source MPEG2 transport stream data generator and packet manipulator developed by Avalpa (www.avalpa.com). It can generate the different tables and convert the table section data to transport streams, filter out PIDs from already multiplexed streams and create object carousels from folders etc.

OpenCaster runs on common Linux© distributions and can be downloaded together with its manual from the Avalpa web page under the Technologies tab.

OpenCaster Glue Code files are located in the Test Suite delivery at TOOLS/OpenCasterGlueCode.

Setup instructions can be found in the TOOLS/OpenCasterGlueCode directory.

5.2.3 Base Test Stream

This section describes the structure of the Transport Stream used by all tests as a basis. For further information on implementation and operation of test cases see 7.5.1.

The base test stream shall be present in the test suite at the path:

`RES/BROADCAST/TS/generic-HbbTV-Teststream.ts.`

5.2.3.1 Elementary stream structure

The base stream total bitrate is 5,000,000 bps. This is the bitrate before any modification; the rate can be set to a specific value in the playout set XML.

Table 3 below shows the elementary stream allocations in the Transport Stream. Some PID allocations are referenced in the PAT/PMT only – there is no content with these PIDs contained in the stream as distributed.

Table 3: Base test stream PID allocations

PID	Contents
0	PAT
16	NIT
17	SDT
18	EIT (contains both actual present/following and actual schedule EIT tables)
20	TDT/TOT
100	PMT for service 10
101	Video
102	Audio
200	PMT for service 11
201	PMT for service 11 (DSM-CC signalled)
205	AIT for service 11 (see note)
300	PMT for service 12
305	AIT for service 12 (see note)
400	PMT for service 13
405	AIT for service 13 (see note)
500	PMT for service 14
505	AIT for service 14 (see note)
NOTE: PID values marked as AIT are populated in the distributed stream and are referenced in the included PAT/PMT. These shall be used as the insertion points of additional data required by the test implementation (e.g. the AIT for the test case).	

Any of the SI tables above which contain a version number shall use the version number of 1. Streams used by the test harness or by test cases shall not use the version number of 1 unless the SI table is an exact match to the SI table in the base stream. This will ensure that cached SI is not used and tests run as expected.

A test harness may play a stream in between test case runs to reset any cached SI from a test case. SI tables used by this stream which differ from the base stream as defined here shall use a version number of 0. Test cases shall therefore also avoid using a version number of 0 for any SI tables.

This Transport stream has original network ID 99, network ID 99, and transport stream ID 1.

5.2.3.2 Service configuration

The following chart shows the structure of the services in the Transport Stream.

PAT/SDT signalled services			
Service 10	Name	ATE Test10	
	PMT PID	100	
	AIT PID	None	
	Video PID	101	
	Audio PID	102	
	Triplet	63.1.a	
	EIT present	Name	ATE Test10 present
		Description	Present event for service ATE Test10
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test10 following
		Description	Following event for service ATE Test10
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 11	Name	ATE Test11	
	PMT PID	200	
	AIT PID	205	
	Video PID	101	
	Audio PID	102	
	Triplet	63.1.b	
	EIT present	Name	ATE Test11 present
		Description	Present event for service ATE Test11
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test11 following
		Description	Following event for service ATE Test11
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 12	Name	ATE Test12	
	PMT PID	300	
	AIT PID	305	
	Video PID	101	
	Audio PID	102	
	Triplet	63.1.c	
	EIT present	Name	ATE Test12 present
		Description	Present event for service ATE Test12
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test12 following
		Description	Following event for service ATE Test12
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	

Service 13	Name	ATE Test13	
	PMT PID	400	
	AIT PID	405	
	Video PID	101	
	Audio PID	102	
	Triplet	63.1.d	
	EIT present	Name	ATE Test13 present
		Description	Present event for service ATE Test13
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test13 following
		Description	Following event for service ATE Test13
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 14	Name	ATE Test14	
	PMT PID	500	
	AIT PID	505	
	Video PID	None (Radio)	
	Audio PID	102	
	Triplet	63.1.e	
	EIT present	Name	ATE Test14 present
		Description	Present event for service ATE Test14
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test14 following
		Description	Following event for service ATE Test14
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Non-signalled services			
Service 11	Name	ATE Test11	
	PMT PID	201	
	AIT PID	205	
	Video PID	101	
	Audio PID	102	
	DSM-CC PID	206	
	Component/Association Tag	200	
	Carousel id	1	
	EIT present	Name	ATE Test10 present
		Description	Present event for service ATE Test10
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test10 following
		Description	Following event for service ATE Test10
		Status	0x1 (following)
		Start time	0xd96c122000

		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	

5.2.4 Secondary Base Test Stream

This section describes the structure of the secondary Transport Stream used by all tests as a basis where two transport streams are required. For further information on implementation and operation of test cases see 7.5.1.

The secondary base test stream shall be present in the test suite at the path

RES/BROADCAST/TS/generic-HbbTV-Teststream_b.ts.

5.2.4.1 Elementary stream structure

The base stream total bitrate is 5,000,000 bps. This is the bitrate before any modification; the rate can be set to a specific value in the playout set XML.

Table 4 below shows the elementary stream allocations in the Transport Stream. Some PID allocations are referenced in the PAT/PMT only – there is no content with these PIDs contained in the stream as distributed.

Table 4: Secondary Base test stream PID allocations

PID	Contents
0	PAT
16	NIT
17	SDT
18	EIT (contains both actual present/following and actual schedule EIT tables)
20	TDT/TOT
100	PMT for service 15
101	Video
102	Audio
105	AIT for service 15(see note)
200	PMT for service 16
205	AIT for service 16 (see note)
300	PMT for service 17
305	AIT for service 17 (see note)
400	PMT for service 18
405	AIT for service 18 (see note)
500	PMT for service 19
505	AIT for service 19 (see note)
NOTE: PID values marked as AIT are populated in the distributed stream and are referenced in the included PAT/PMT. These shall be used as the insertion points of additional data required by the test implementation (e.g. the AIT for the test case).	

Any of the SI tables above which contain a version number shall use the version number of 1. Streams used by the test harness or by test cases shall not use the version number of 1 unless the SI table is an exact match to the SI table in the base stream. This will ensure that cached SI is not used and tests run as expected.

A test harness may play a stream in between test case runs to reset any cached SI from a test case. SI tables used by this stream which differ from the base stream as defined here shall use a version number of 0. Test cases shall therefore also avoid using a version number of 0 for any SI tables.

This Transport stream has original network ID 99, network ID 65281, and transport stream ID 2.

5.2.4.2 Service configuration

The following chart shows the structure of the services in the Transport Stream.

PAT/SDT signalled services			
Service 15	Name	ATE Test15	
	PMT PID	100	
	AIT PID	105	
	Video PID	101	
	Audio PID	102	
	Triplet	63.2.f	
	EIT present	Name	ATE Test15 present
		Description	Present event for service ATE Test15
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test15 following
		Description	Following event for service ATE Test15
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 16	Name	ATE Test16	
	PMT PID	200	
	AIT PID	205	
	Video PID	101	
	Audio PID	102	
	Triplet	63.2.10	
	EIT present	Name	ATE Test16 present
		Description	Present event for service ATE Test16
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test16 following
		Description	Following event for service ATE Test16
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 17	Name	ATE Test17	
	PMT PID	300	
	AIT PID	305	
	Video PID	101	
	Audio PID	102	
	Triplet	63.2.11	
	EIT present	Name	ATE Test17 present
		Description	Present event for service ATE Test17
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test17 following
		Description	Following event for service

			ATE Test17
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 18	Name	ATE Test18	
	PMT PID	400	
	AIT PID	405	
	Video PID	101	
	Audio PID	102	
	Triplet	63.2.12	
	EIT present	Name	ATE Test18 present
		Description	Present event for service ATE Test18
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test18 following
		Description	Following event for service ATE Test18
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	
Service 19	Name	ATE Test19	
	PMT PID	500	
	AIT PID	505	
	Video PID	None (Radio)	
	Audio PID	102	
	Triplet	63.2.13	
	EIT present	Name	ATE Test19 present
		Description	Present event for service ATE Test19
		Status	0x4 (running)
		Start time	0xd96c112000
		Duration	0x1000
		Language code	eng
	EIT following	Name	ATE Test19 following
		Description	Following event for service ATE Test19
		Status	0x1 (following)
		Start time	0xd96c122000
		Duration	0x1000
		Language code	eng
	EIT schedule	(Table present but empty)	

6 Test Case specification and creation process

6.1 Test Case creation process

The HbbTV Test Cases are based on the optional and mandatory requirements as defined in the HbbTV Technical Specification. Test Cases are proposed and managed by the HbbTV Test Group.

The process for defining, implementing and accepting HbbTV test cases consists of the steps as depicted in Figure 2.

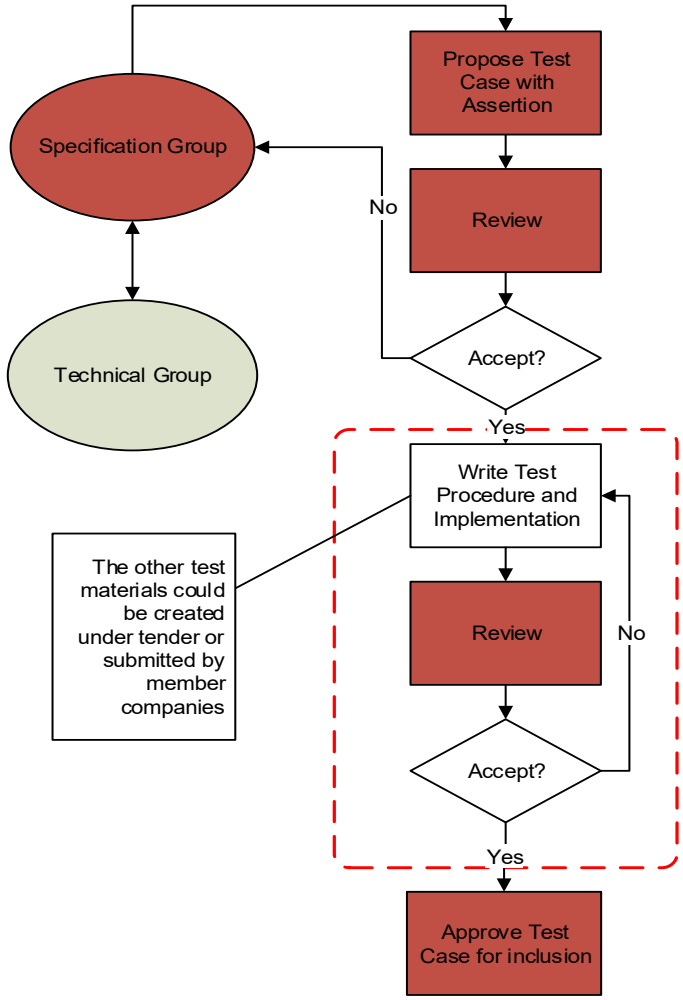


Figure 2: Test creation work flow

6.1.1 Details of the process for creating Test Cases

The Figure 2 only provides an abstract view of the work flow. There are some criteria for approving test cases. E.g. A test can be approved 4 months have elapsed since the test case most recently reviewed and no issues are raised since then. The detailed process steps for the acceptance of created HbbTV test material is described in the HbbTV Test Material Approval Procedure document which is part of the Test Suite package stored in the Documents folder.

6.2 Test Case section generation and handling

To enable the test process to be handled in a flexible and timely manner the Test Case part of the Test Specification is generated by applying following method:

- Each relevant specification item has been translated into one or more Test Cases.
- Each Test Case is described by a defined set of attributes as listed in section 6.3.
- The Test Case attributes shall be stored in the corresponding XML file which is validated by Schema SCHEMAS/testCase.xsd

NOTE: The use of XML for Test Case definition enables automated processing capabilities. I.e. to use scripting that can generate overviews of existing Test Cases, apply filtering, and allow for flexible generation of various output file formats.

6.3 Test Case Template

Each Test Case consists of a list of attributes, as described below.

6.3.1 General Attributes

The General Attributes uniquely identify the Test Case.

6.3.1.1 Test Case ID

The Test Case ID is a string that uniquely identifies the test case. It contains two parts, a “namespace” and a “Local ID”, separated by an underscore.

For official HbbTV tests, the Test Case IDs will usually be allocated by the testing group. In this case, the namespace shall be “org.hbbtv”. E.g. “org.hbbtv_0000123F”. The testing group must ensure those tests IDs are unique.

It is important that every test has a different Test Case ID. If another organization wants to generate Test Case IDs for its own tests, then it must not use the “org.hbbtv” namespace. Instead, it must take a domain name it controls, reverse it, and use that for the namespace part of the Test Case ID. In this case, the Local ID can be anything permitted by the schema. Organizations should have some internal procedure to allocate Local IDs so that they don’t generate duplicate Test Case IDs.

For example, a company that controls the “example.com” domain could use Test Case IDs like “com.example_FOO”, or “com.example_BAR_BAZ_9876-42”

(To be clear: The domain name used in Test Case IDs must be a real domain name, and must be registered on the Internet in the usual way, using the normal ICANN roots. There is no need for there to be a website there).

See 8.2 for further details on how the Test Case ID is used.

6.3.1.2 Test Case Version

The Test Case Version specifies a specific version of the Test Case and has the following format: <integer> version. See 8.2 for further details on how the Test Case version is used.

6.3.1.3 Origin Information

The Origin Information lists the organizations that own IPR in this Test Case. At least one contributor needs to be specified for each Test Case. However, more than one contributor can be listed. For each contributor, the following attributes exist:

6.3.1.3.1 Part

This is the part of the test case that this organization contributed to. It can be "assertion", "procedure" or "implementation". If the same organization contributes two or three of these parts, they should add a contributor tag for each part.

The “procedure” part is deprecated; everything that does not belong to the “assertion” part is now considered part of the “implementation”. New test material shall not use the “procedure” part. Older test material may still include it.

6.3.1.3.2 Company

The author’s company name (mandatory). If the author is an individual who is not associated with a company, this shall be the author's full name.

6.3.1.3.3 Contact Email address

A contact email address for the contributor (mandatory). This should be an address that will be active for many years. It will typically be a generic contact address, not a named individual. This may be used to contact the contributor with questions about licensing, as well as technical questions.

6.3.1.3.4 License information

The license that this contributor uses for the specified part of this test. Valid values are:

- "TMPA Commercial"
- "TMPA CC BY-NC-ND"
- “HbbTV Commercial”

Tests created by other groups may use different values here.

6.3.1.4 Title

A short title to identify this specific Test Case (mandatory).

6.3.1.5 Description

A longer description of what the test does if the title is not sufficient (optional). The format of the Test Case Description is a text field (no limit). Whitespace is not significant.

6.3.2 References

6.3.2.1 Test Applicability

The test specifies which specifications it applies to. Official HbbTV tests shall specify this element and shall use a name of "HBBTV" (case sensitive) and a version of "1.1.1", "1.2.1", "1.3.1" or "1.4.1". Tests that are valid for more than one version of HbbTV shall include tags for all applicable versions. Tests may include additional tags to indicate non-HbbTV specifications that are tested (e.g. OIPF). The master list of specification names is kept on the Wiki page “AppliesToSpecNames” [26]. For each spec element in the <appliesTo> tag, the test case shall:

- be conformant to that specification, and
- test some feature of that specification.

For example, a test which tests an OIPF feature which is also required for both HbbTV 1.1.1 and 1.2.1 would use the following appliesTo element:

```
<appliesTo>
    <spec name="HBBTV" version="1.1.1"/>
    <spec name="HBBTV" version="1.2.1"/>
    <spec name="HBBTV" version="1.5.1"/>
    <spec name="HBBTV" version="1.6.1"/>
```

```
    <spec name="OIPF" version="1.2"/>
</appliesTo>
```

It is not necessary to include a spec element for every potential regime that could reference HbbTV to use this test. For instance, a country-specific testing regime may require support for HbbTV and seek to use a particular test case - it is not required to include a spec element for that regime.

Every test case shall have an appliesTo element with at least one spec element. It must include a spec element for at least one HbbTV version. It may also include spec elements for other specifications as listed on the "AppliesToSpecNames" Wiki page [26].

6.3.2.2 Specification References

References to the different specification sections or versions. For each version of the HbbTV specification, there is a list of sections covered by this Test Case (top-level). Each top level entry includes references to one or more specification sections (HbbTV sections and optionally external specification sections, e.g. OIPF DAE). Each specification section has the following attributes.

6.3.2.3 Document Name

A short identifier for the document that contains the specification section (e.g. "HBBTV" for the HbbTV specification). While the schema allows any value, HbbTV tests must use the values defined in the wiki page [25]. If you need to reference a spec that isn't listed there, add it to that Wiki page.

6.3.2.4 Chapter

The chapter number within the specified document (a dot separated list of integers or characters without spaces, e.g. 9.3.1)

6.3.2.5 Specification Text

The specific text from the referenced specification section tested by this Test Case (optional). The format of the specification text is a text field (no limit). Whitespace is not significant.

6.3.2.6 Assertion Text

Describes what is tested in this test case (assertion). The format of the Assertion value format shall be a text field (no limit). Whitespace is not significant.

For HbbTV and OIPF tests, there shall be at most one assertion. (Other testing groups that reuse the HbbTV Test Case XML format may relax this limit).

6.3.2.7 Test Object

The description of the object under test (specification part or API part). Allowed characters include uppercase and lowercase ASCII letters, numbers, dot, underscore, dash, and forward slash.

The clause is optional. The master list of object names is kept on the Wiki page "TestObjectNames" [26].

Multiple test objects may be specified if appropriate.

If none of the above are suitable then the clause is omitted.

6.3.3 Preconditions

Lists preconditions on the DUT before this test can be run.

6.3.3.1 Required Terminal Options

The terminal options required on the DUT to run this test (if empty, this test is mandatory for all devices). It contains a combined list of option strings as defined in [1], section 10.2.4. The format of the Required Terminal Options value is a text field without spaces.

Available options are +DL for file download functionality, +DRM for DRM functionality, +PVR for PVR functionality, +SYNC_SLAVE for slave operation in inter-device synchronisation, +IPH for "IP delivery Host player mode" as defined in the DVB Extensions to CI Plus ETSI TS 103 205, +IPC for "IP delivery CICAM player mode" as defined in the DVB Extensions to CI Plus ETSI TS 103 205, +AFS for Support for the CICAM Auxiliary File System as defined in the DVB Extensions to CI Plus ETSI TS 103 205. Multiple requirements are concatenated to a single string without spaces in between. They must be listed in alphabetical order. Example: +DL+PVR

You can also use "-" prefixes to indicate the test should only run on devices that do not support the feature. For example, a test with required terminal options set to "-PVR" will not be run on PVRs, but will be run on terminals without PVR functionality.

6.3.3.2 Optional Features

Terminal features which are required on the DUT, in addition to required terminal options, to run this test. The master list of available feature strings is on the “OptionalFeatures” wiki page [42].

Some features specific to testing Operator Applications are defined in D.3.5.1 Operator Application Optional Features.

Multiple required features are concatenated to a single string without spaces in between. They must be listed in alphabetical order.

You can also use "-" prefixes to indicate the test should only run on boxes that do not support the feature. For example, a test with optional features set to "-EAC3" will not be run on terminals with E-AC-3 support, but will be run on terminals without E-AC-3 support.

You can also use the <or> tags to specify a requirement for logical combinations of features. For example, “this test requires either feature VK_PLAY_PAUSE, or both features VK_PLAY and VK_PAUSE” can be written as:

```
<preconditions>
  <or>
    <optionalFeatures>+VK_PLAY_PAUSE</optionalFeatures>
    <optionalFeatures>+VK_PLAY+VK_PAUSE</optionalFeatures>
  </or>
</preconditions>
```

You can also use the <and> tag within an <or> tag to combine requirements for Optional Features with requirements for Required Terminal Options within a logical combination of features. For example, “this test requires either optional feature OF_A, or both required terminal option TO_B and optional feature OF_C” can be written as:

```
<preconditions>
  <or>
    <optionalFeatures>+OF_A</optionalFeatures>
    <and>
      <requiredTerminalOptions>+TO_B</requiredTerminalOptions>
      <optionalFeatures>+OF_C</optionalFeatures>
    </and>
  </or>
</preconditions>
```

By default, a feature is required for all specification versions. The <optionalFeaturesVersionRange> tag can be used to indicate that a feature is required only for some specification versions (and is not optional for other specification versions).

```
<optionalFeaturesVersionRange start="1.3.1">+OF_A</optionalFeaturesVersionRange>
```
Indicates that OF_A is required for devices supporting version 1.3.1 and later

```
<optionalFeaturesVersionRange end="1.4.1">+OF_B</optionalFeaturesVersionRange>
```
Indicates that OF_B is required for devices supporting up to version 1.4.1, but not later versions

```
<optionalFeaturesVersionRange start="1.3.1" end="1.6.1">+OF_C</optionalFeaturesVersionRange>
```

Indicates that OF_C is required for devices supporting versions between 1.3.1 and 1.6.1 inclusive. The “start” and “end” attributes may be the same to indicate a single version.

6.3.3.3 Text Condition

A textual description of a precondition. Whitespace is not significant. There are two types of textual precondition, defined below.

6.3.3.3.1 Informative

This description is optional and considered informational for reviewers or implementers (it cannot be assumed that a test operator will see this pre-condition). The format of the textual precondition is a text field (no limit).

6.3.3.3.2 Procedural

This description requires the tester to take specific action (procedural) prior to the execution of the Test Case. If present the execution of this action shall be considered mandatory.

6.3.3.4 TestRun

A reference to zero or more Test Cases that must be passed successfully before this Test Case can run. The Test Cases are referenced by their Test Case ID (see above for format description).

6.3.4 Adaptive Tests

6.3.4.1 adaptsTo

There is an API, getDUTOptions, to allow the test case to query the Terminal Options and Optional Features that the tester claims the DUT supports and other settings, as configured in the test harness. This allows a Test Case to make fine-grained decisions about what to do based on what features the DUT supports and other settings. E.g. this API can be used if a large Test Case has a small bit that’s different dependent on a DUT feature.

There is an XML tag in the testcase XML file:

```
<adaptsTo>
    <terminalOptions>+DRM</terminalOptions>
    <optionalFeatures>+VK_PLAY+VK_PLAY_PAUSE</optionalFeatures>
    <setting>PARENTAL_REGION_DVB_SI</setting>
    <setting>PARENTAL_REGION_XML</setting>
</adaptsTo>
```

This lists the terminal options and optional features and settings that can affect the test case. For terminal options and optional features, only +OPTION can be specified, not –OPTION. For settings, the valid values are:

Identifier	Meaning
PARENTAL_REGION_DVB_SI	The configured 3-byte DVB region code as defined for parental_rating_descriptor. This is decoded from bytes to a string using the ISO/IEC 8859-1 encoding. This is used in broadcast AIT generation where required (see 7.4.4.3.1 AIT)
PARENTAL_REGION_XML	The configured region code in the format for the Region attribute of an OIPF ParentalRating XML tag. This should be an alpha-2 as defined in ISO 3166-1. It is expected that this refers to the same region as PARENTAL_REGION_DVB_SI, it's just represented differently. The test case can use this, with some PHP code, to generate XML AITs for broadcast-independent applications containing the

	correct ParentalRating tag Region attribute.
OPAPP_TLS_CLIENT_CA_TRUST_ANCHOR_CERT	The PEM-encoded Client CA trust anchor certificate. This shall either be a Client Root CA certificate or a Client Intermediate CA certificate, as defined in [37] sections 11.2.1.2.2 and 11.2.1.2.3.

If the test case does not adapt to any terminal options, then this <terminalOptions> tag must be omitted. If the test case does not adapt to any optional features, then this <optionalFeatures> tag must be omitted. The test case must include one of the <setting> tags if and only if it adapts to that setting. If the test case does not adapt to anything then the entire <adaptsTo> tag must be omitted.

The Test Harness shall have a mechanism where the tester can configure each setting listed in the table in the current section before running a test case.

6.3.4.2 Extended settings information

The test also indicates what extended settings it requires as defined in 7.2.13 Extended Settings, using this XML syntax:

```
<usesExtendedSettings>
   <setting key="vendorName"/>
   <setting key="numberOfTuners"/>
</usesExtendedSettings>
```

Each key must refer to a setting in one of the RES/META/ExtraInfoRequired/*.xml files. If it does not, then the test case is invalid.

The specified settings must be configured by the tester (or implicitly to their default values, if default values are specified and the tester has not explicitly configured them) before the test is run. If they are not configured, the harness may refuse to run the test case or immediately fail the test case, or it may run the test case anyway (in which case it will fail if getExtendedSetting() is called).

Test case authors must only list settings if the test case may actually call getExtendedSetting() for that setting and use the returned value.

NOTE: This XML makes it clear what tests have to be re-run if an extended setting is changed.

6.3.5 Testing

6.3.5.1 Test Procedure

Steps to perform in the test. The steps mentioned here should describe the major steps that can be found in the implementation code. The actual implementation may consist of more (and probably smaller) steps, but the general layout of the implementation should be described here. Each step has the following elements:

6.3.5.1.1 Index

The numerical sequence index of the test step (describing the order of steps to perform). This starts with index 1 for the first step and increments by 1 for each subsequent step. This attribute is optional.

6.3.5.1.2 Procedure

Contains the textual description of a single step. The format of the description is a text field (no limit). Whitespace is not significant.

6.3.5.1.3 Expected behaviour

Optionally describes the expected behaviour of the DUT when executing this test step.

6.3.5.2 Pass Criteria

Textual description of criteria required to pass the test. The format of the Pass Criteria value is a text field (no limit). Whitespace is not significant.

6.3.5.3 Media Files

Describes a media file used by the test. This is an optional element in the XML, and older tests don't use it. For new tests, it is recommended to use this element to define the media files.

If a test uses multiple media files, this element can be repeated to define each file.

6.3.5.3.1 Name

The path (preferred) or file name for the media file. The procedure can use this file name to refer to the media file.

This may be EITHER:

- a path relative to the directory containing the test case XML file, OR
- a file name only, in which case the location of the file is not specified (not recommended for new tests)

Where this attribute is a path, it shall use "/" as a directory separator. Paths must contain at least one "/" to distinguish them from file names. If the file is in the same directory as the test case XML file, the path shall start with "./", for example "./file.mp4". The use of "." or ".." is not allowed, except that the path may start with "./" or "../../RES/". If the path does not contain "/", it is treated as a file name.

In all cases, the attribute shall not contain "\" or any character that is forbidden in Windows paths.

6.3.5.3.2 Description

A human-readable description of the media file. The format of the description value is a text field (no limit). Whitespace is not significant.

6.3.5.4 Derived From

Indicates that this test is a derivative of another and what files differ.

6.3.5.4.1 Id

This attribute of the derivedFrom tag gives the id of the test case that this test case is derived from (the parent).

6.3.5.4.2 Difference

Definition of files that differ from the parent. There must be at least one difference tag within a derivedFrom tag. The filename attribute gives the name of a file in this test case that is different from the parent, relative to the directory containing the XML file for this test case. The difference element can optionally contain an explanation of what differs from the parent and why – particularly important for binary files where this may not be readily apparent or easy to understand with difference tools. If the file that is different from the parent is renamed, the original name should be stated in the difference element.

6.3.5.5 Has Derived

Empty tag that indicates that this test has derivative tests. To find the derivative tests, look for tests with a derivedFrom tag specifying the id of this test.

6.3.5.6 History

Contains a history of additions and modifications to this test.

6.3.5.4.1 Date

The date of change/update/proposal. The format of the date is YYYY-MM-DD (e.g. 2010-06-10).

6.3.5.4.2 Part

The part of the test that was or is to be updated. Contains one of the following values:

- assertion: for the test metadata (including the assertion)
- procedure: for the test procedure (test steps),
- implementation: for the actual implementation of the test (not included in this document)

The “procedure” part is deprecated; everything that does not belong to the “assertion” part is now considered part of the “implementation”. New test material shall not use the “procedure” part. Older test material may still include it.

6.3.5.4.3 Type

The type of update. Contains one of the following values:

- proposed: Used to indicate the intention to provide the material for the specified part
- submitted: For the initial submission or an update that has to be reviewed
- review_proposed: To indicate the intention to review the specified part
- reviewed: If the update was reviewed and accepted by someone
- accepted: If the update was accepted
- rejected: If the update was rejected and probably should be modified and resubmitted.
- edited: The update provides only editorial changes and has no impact on the test logic. Any changes of this type do not require review.

6.3.6 Others

6.3.6.1 Remarks

Remarks and comments to this test (optional). The format of the remarks is a text field (no limit).

6.4 Test Case Result

6.4.1 Overview (informative)

This document describes the technical process for verification of HbbTV Devices, and as such the choice of pass or fail given by the criteria in 6.4.2 are those that shall be used when generating a conformance report.

A test harness may generate and store other data about tests and use this to present alternative reports to operators. Such information may include indications that although a test would currently be considered as failed by the criteria in 6.4.2, the test may be considered as ‘incomplete’, or passed after further events have taken place (e.g. further test steps or a call to the endTest API method). Such indications and reporting are outside the scope of this specification.

6.4.2 Pass criteria

The result of the test shall be PASS only if all of the following criteria are met:

1) All normative test preconditions were satisfied
2) All step results stored by the test harness have the value ‘true’ for the ‘result’ parameter
3) All calls to analyze API method have been evaluated to give a result and that result is ‘true’

4) All calls to the test API methods that interact with the test environment (e.g. sendKeyCode, changePlayoutSet) succeeded

5) The ‘endTest’ API method was called

If, at a given time, any of the above criteria are not met the result at that time shall be FAIL. The result may change to PASS if these criteria are met at a later time (e.g. an analyze API method is evaluated, or a call to the endTest API method is made.) As step results and analysis results may not be changed, if at a given time the result is PASS then at a later time the result shall not change to FAIL.

7 Test API and Playout Set definition

7.1 Introduction

This section describes the following interfaces:

- A JavaScript API that defines the interface between the Test Harness and DUT.
- A set of XML files that define how the Test Harness should interpret a test case. This allows definition and control of DVB playout required to initiate a test.

By defining these interfaces is it possible for a test case contributor to author test cases that can be run on any HbbTV compatible Test Harness. Similarly, the interface definition allows multiple Test Harnesses to be implemented, with different levels of automation, but still all compatible with test cases that adhere to the interface.

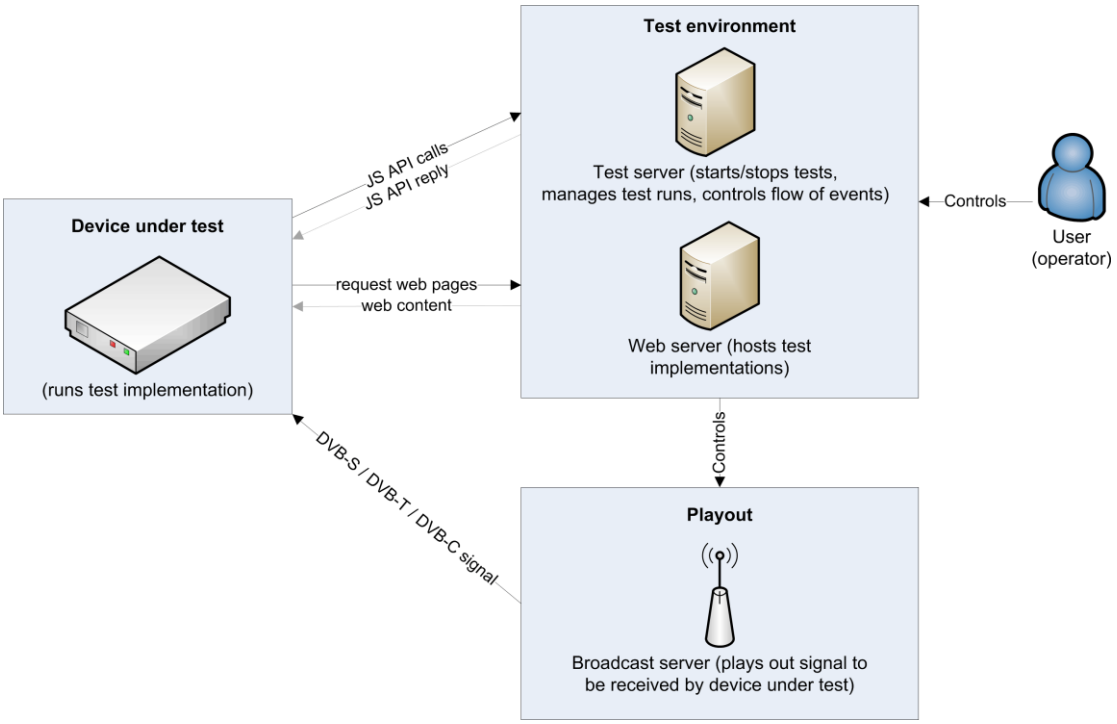


Figure 3: Overview of JS API communication between DUT and Test Harness

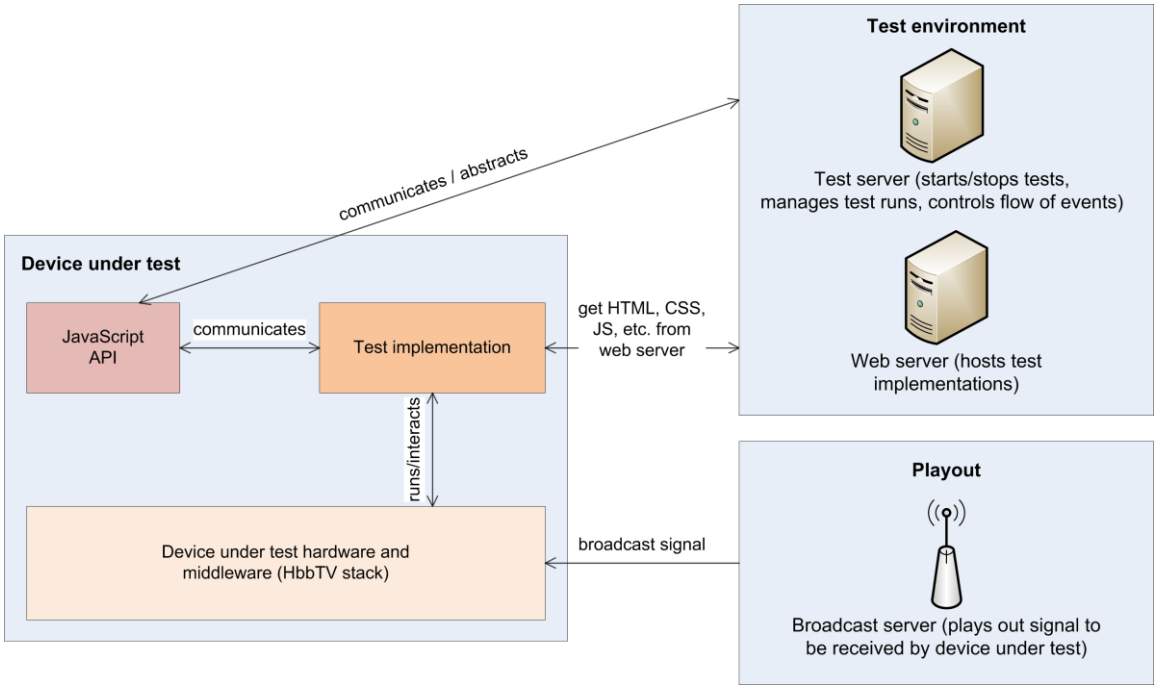


Figure 4: Detailed overview on JS-API abstraction of Test Harness communication

The layout of the APIs described in this document is designed in a way that allows for a high percentage of automation.

There is no necessity for an HbbTV Test Environment to be fully automated. HbbTV JS APIs are therefore designed in such a way that any required test may be operated manually. The implementer of an HbbTV Test Harness may choose for interaction by an operator with the HbbTV Test Environment in a manual way.

The HbbTV JS APIs allow for multiple implementations from different implementers and are designed in such a way that it allows a potential test implementer to implement compatible Test Cases and or Test Harnesses. This allows for combining test cases created by one or more implementers of test cases to create a complete HbbTV (automated) Test Environment.

The JavaScript APIs are divided into four parts:

1) APIs communicating with the Test Environment (see section 7. 2). It informs the Test Environment about the current test's status.

2) APIs communicating with the Device under Test (see section 7.3). This part of the API communicates with the DUT (e.g. send key codes, make screenshots). This can be either be implemented directly by the DUT manufacturer (send commands directly via Ethernet to the DUT), or it can be implemented by someone else (e.g. send commands to an IR sender or a frame grabber).

3) APIs communicating with the Playout Environment (see section 7.4). This part of the API communicates with the Playout Environment (which generates and transmits the DVB-S/-C/-T signal to the DUT). For example, the Playout Environment is responsible for sending the correct AIT to the DUT, such that the test is started on the DUT.

4) APIs for testing specific further areas of functionality (DIAL, Websockets, Media Synchronization, Network Testing, etc.). See sections 7.6+

7.1.1 JavaScript strings

Where a String is passed to any JS functions in the HbbTV test API, the following rules apply:

1) It must contain valid UTF-16

2) It must only contain Unicode code points that are allowed by the XML 1.0 specification (http://www.w3.org/TR/2006/REC-xml-20060816/#charsets, section 2.2 Characters).

3) It must not contain the code point U+000D.

I.e. the JavaScript string can only contain the Unicode code points U+0009, U+000A, U+00020-U+D7FF, U+E000-U+FFFD, and U+10000-U+10FFFF.

If a test breaks the rules in the previous paragraph, the test harness shall fail the test.

The test harness must handle any XML escaping necessary when writing characters such as "<" in the test report XML.

7.2 APIs communicating with the Test Environment

7.2.1 Starting the test

The DUT must be in a pre-defined state before a test is started. The pre-defined state is defined in 7.2.2. There are two ways to execute a test:

- The test is started automatically by the DUT as soon as the AIT is parsed and the test application (which is signalled as “AUTOSTART”) is detected. This will open the entry URL of the initial test page in the browser on the DUT.
- The test is started by the test harness, executing an application using its resident ECMAScript environment. When the test is started is up to the test harness implementation. This mechanism is used only when the test cannot be practically implemented on the DUT.

NOTE: All JS functions defined in this document are defined within the HbbTVTestAPI prototype. This means that e.g. for reportStepResult, you would call “testapi.reportStepResult(...);”

7.2.1.1 Tests started automatically by the DUT

The initial test page includes the JavaScript file “../RES/testsuite.js” which contains the implementation of the JavaScript classes/functions defined in this document. The testing API described in this document is then initialized. To initialize the testing API and to set up the connection to the test harness, the following steps need to be performed by the test application (usually the initial start page referenced in the AIT).

- Include the JavaScript file “testsuite.js” from “RES” directory (either by relative reference “../../RES/testsuite.js” or by absolute reference “http://hbbtv1.test/_TESTSUITE/RES/testsuite.js”), e.g.:

```
<script type="text/javascript" src="../../RES/testsuite.js"></script>
```

- Initialize the test suite by creating a new instance of the HbbTVTestAPI class and calling init() on it. The init() function needs to be called by the initial test page to initialize the connection between the DUT and the server. e.g.:

```
<script type="text/javascript">
var testapi;
window.onload = function() {
     testapi = new HbbTVTestAPI();
     testapi.init();
};
</script>
```

7.2.1.2 Tests started by the harness

The test application includes the JavaScript file which contains the implementation of the JavaScript classes/functions defined in this document. The testing API described in this document is then initialized by creating a new instance of the HbbTVTestAPI class and calling init() on it.

7.2.1.3 Multiple clients

The test harness shall support having up to three “clients” all running at the same time and all using the HbbTVTestAPI, which can include:

- harness-based test code (as already defined in 5.2.1.6 ECMAScript Environment)
- a “Regular” HbbTV application running on the device under test;
- an operator application (of either type) running on the device under test.

The operator application shall only be supported if Annex D is supported. If it is not supported, the test harness shall support having up to two “clients” all running at the same time, harness-based test code and a regular application.

If more than one client uses HbbTVTestAPI at the same time then:

- The openWebsocket() API must only be used by one client in a test case.
- The following functions are classified as “multi-client-safe” and can be used by all clients at any time without worrying about what other clients are doing:
    - reportMessage, reportStepResult, waitForCommunicationCompleted, multiClientConnect, MultiClientComms.send, getPlayoutInformation, getPlayoutStartTime
    - HbbTVTestAPI() constructor and init() method, although each client may only create one HbbTVTestAPI instance.
    - endTestApp, although each client may only call it once
- The testing JS APIs not explicitly listed in the previous 2 bullet points are classified as “primary-client-only”. The test case author shall choose at most one client at a time as the “primary” client. There is no special API to identify the “primary” client, it is just a concept (and should be documented in comments!). Only the “primary” client may make calls to any of the primary-client-only APIs. The test case may change which client is the “primary” client as often as it likes, but it must ensure that no primary-client-only API calls are in progress at that time. The multi-client messaging APIs defined in 7.2.12 JS-Functions for multiple-client communication can be used to co-ordinate a change of primary client.
- endTest() has an extra restriction, as well as being primary-client-only: The test case must ensure that any other clients that made reportMessage / reportStepResult calls have subsequently called waitForCommunicationCompleted and got the callback from that before endTest() is called, else messages will be lost. The client that is calling endTest does **not** need to call waitForCommunicationCompleted, it is implicit in the endTest() call.
- Instead of calling endTest() to end the test, consider using endTestApp() instead. It is designed to be simpler in multiple-client scenarios.
- If there are simultaneous calls to changePlayoutset and either getPlayoutInformation or getPlayoutStartTime, then it is undefined whether getPlayoutInformation and getPlayoutStartTime return information about the new or old playoutsets. This is similar to the situation where the playoutset changes due to a timeout set in implementation.xml while those APIs are being called. It is recommended to design test cases to avoid these situations.

7.2.2 Pre-defined state

Before starting a test, the DUT must be in the state described in this chapter. The method for setting the DUT to the pre-defined state is specific to the testing environment of the Test Harness and the DUT, and is not described in this document. Possible solutions for setting the DUT to the pre-defined state are:

- using a proprietary API provided by the DUT manufacturer
- using an external solution: resetting the power of the DUT and tuning to a pre-defined channel via the remote control (or automated IR sender)

The pre-defined state of the DUT is as follows:

- services scanned and stored in the service list of the DUT , as defined in section 7.2.2.1.
- tuned to transponder 1, service “ATE test11”

- no application is running, so the AUTOSTART application signalled on that service will be started automatically.

Additional requirements are also specified in section 7.4.4 of this document.

7.2.2.1 Initial Service List

The service list on the DUT at the start of a test case shall be configured as follows:

1. If the test case implementation.xml file has an <installPlayoutSet> tag, then:
    a. If that tag has a skipInstallationBeforeThisTest="true" attribute, then there are no requirements on the initial service list – it may contain anything.

        NOTE: This is typically used by harness-based test cases that are going to do a service scan early in the test case.

    b. Else the service list on the DUT must be the one that would be created by playing out the XML playoutset file specified by the definition="" attribute on that tag, and doing a full rescan on the DUT.
2. Else if any playoutset in the test case uses two modulators, then the initial service list shall be the one that would be created by playing out the two standard base streams on different modulator channels, and doing a full rescan on the DUT. It is acceptable for the tester or test harness to use streams that have equivalent SI to the standard base streams, instead of the actual base streams. The standard base streams are:
    - modulator channel 1: the file RES/BROADCAST/TS/generic-HbbTV-Teststream.ts or equivalent. This has network id=99, original network id=99, transport stream id=1, and contains the following services:
        - service name "ATE test10", TV service, id=10
        - service name "ATE test11", TV service, id=11
        - service name "ATE test12", TV service, id=12
        - service name "ATE test13", TV service, id=13
        - service name "ATE test14", Radio service, id=14
    - modulator channel 2: the file RES/BROADCAST/TS/generic-HbbTV-Teststream_b.ts or equivalent. This has network id=65281, original network id=99, transport stream id=2, and contains the following services:
        - service name "ATE test15", TV service, id=15
        - service name "ATE test16", TV service, id=16
        - service name "ATE test17", TV service, id=17
        - service name "ATE test18", TV service, id=18
        - service name "ATE test19", Radio service, id=19
3. Else the initial service list shall be as described in the previous option, except that playing out the "modulator channel 2" stream during the rescan is optional.

7.2.2.2 Changing the Service List during a test case

Test cases must not change the service list from its initial value unless they indicate that by setting the forceInstallationAfterThisTest="true" attribute on the <installPlayoutSet> tag.

NOTE: People sometimes **incorrectly** think that "*if this test case puts the channel list back how it was, then it doesn't need forceInstallationAfterThisTest="true"*". **That is <u>WRONG</u>**. In the case where the test fails, it is **impossible** for the test case to "put the channel list back how it was". So such a test case needs forceInstallationAfterThisTest="true".

If a test case sets skipInstallationBeforeThisTest="true" then the test case must also set forceInstallationAfterThisTest="true"

7.2.2.3 The Install Playoutset XML tag

The "Install Playoutset" may be specified using this optional tag in the Implementation XML file:

```
<installPlayoutSet definition="playoutset3.xml"/>
```

NOTE: Most test cases will not need to use this tag. Omitting it makes the test harness use a standard install playoutset as defined in 7.2.2.1.

The value of the mandatory "definition" attribute is a relative path to a playoutset XML file. The path is relative to the directory containing the Test Case XML file, and uses "/" as a directory separator. The path must be within the Test Suite, but it can be within the test case directory (or a subdirectory) or a shared file in RES/.

When moving from one test case to another, the test harness needs to know whether the new "Install Playoutset" is the same or different, to decide whether or not the DUT needs rescanning. This decision may be based purely on whether they have the same (absolute) path. So if two Test Cases require the same DVB environment then they should both refer to the same "Install Playoutset" XML in the RES/ directory.

The playoutset XML specified here must specify a playoutset containing no applications. (It may contain AITs, but those AITs must either not signal any applications, or signal applications with the KILL code).

There are two other, optional, attributes:

```
<installPlayoutSet definition="playoutset3.xml"
      skipInstallationBeforeThisTest="true"
      forceInstallationAfterThisTest="true"/>
```

The meaning of these attributes is documented in section 7.2.2.1. If not specified, these attributes default to false.

NOTE: Coding style guideline: Don't specify these attributes with the value false, just omit the attributes(s) completely and let them default to false.

NOTE: When a test is not running, the test harness may play out a stream that has the same identification, services, and SI/PSI as the standard HbbTV testing stream. This allows the user to do a channel scan before running the tests, and ensures the DUT doesn't go into "no signal" mode when the test ends. This XML tag allows each test case to specify an "Install Playoutset", which is the stream that the tester should use to do the channel scan, instead of the standard stream. When starting a new test, if it specifies the same "Install Playoutset" as the previously-run test, then the test may just run without the user having to rescan the DUT. If the new test specifies a different "Install Playoutset" from the previously-run test, then the test harness may play out the new "Install Playoutset" and prompt the user to do a rescan on the DUT before running the test case.

7.2.3 Callbacks

All API calls defined in this document are asynchronous, as the API may have to interact with the test harness. To know when the API call actually completes, the caller needs to provide a callback function. This function is called as soon as the API call completes (either locally on the DUT or on the server). The first argument of the callback function is always the callback object passed to the API function. This allows the callback function to determine which API call was processed (if the same callback function is used for multiple API calls). The callback function might be called with additional arguments. These additional arguments are defined in the respective API function definition.

The implementation of a function can either be synchronous (e.g., via OIPF debug function, chapter 7.15.5 of OIPF DAE) or asynchronous (e.g. via XMLHttpRequest to the server). The callback function shall be in both cases asynchronous.

Specifying callback functions and callback objects is not required. The callback arguments may be null if the caller is not interested in the callback and the callbackObject may be null if no state is required.

7.2.4 JS-Function getPlayoutInformation()

This function retrieves information on the current playout.

```
void HbbTVTestAPI.getPlayoutInformation(
        callback : function,
        callbackObject : object);
```

The parameters prefixed ‘signalLevel’ below may be undefined if the harness does not implement Operator Application testing as described in Appendix D. Test cases that are not testing Operator Applications shall not reference these parameters.

ARGS: **callback/callbackObject:** a callback function to invoke when the information is available (also see chapter 7.2.3 Callbacks). The callback function will be called with the following parameters: callback(callbackObject, playoutInformationObject) where the playoutInformationObject is a JavaScript object which has a ‘length’ property that returns the integer number of multiplexes in the current playout set. It can be indexed using the integers 0 to `length-1` to obtain JavaScript objects giving information about each multiplex in the current playout set. The order shall match the order that the multiplexes were defined in the playout set XML file. For each multiplex object, the following properties shall be available:

transponderDeliveryType: the idType (integer) for the DVB delivery type (DVB-S(2), DVB-T(2), DVB-C(2)), as specified in OIPF DAE, chapter 7.13.11.1 to be used for createChannelObject() function calls.

transponderDsd: the delivery system descriptor (tuning parameter) as specified in OIPF DAE, chapter 7.13.1.3 to be used for createChannelObject() function calls, with the corrections from section A.2.4.4 of the HbbTV specification [1] with errata 1 applied

eitOffset: a non-negative integer that is the ‘EIT Offset’ for this multiplex as defined in section 7.4.4.7. If the <adjustEit/> tag was not specified for this multiplex then this shall be zero.

signalLevelCurrent: the current output signal level in dBm, as a floating-point number. Note that this will normally be negative. Note that the actual signal level received by the terminal will usually be lower due to losses in the cabling connecting the harness to the terminal, especially if a signal combiner is in use to support multiple multiplexes. Also note that there may be some error in this value, modulator manufacturers typically quote an accuracy of +/-3dBm. But it is expected that relative changes in this value will result in similar relative changes of the signal level at the input to the terminal.

NOTE: We do not require this value to be accurate and calibrated because that would require everyone running the tests to have calibrated RF analysis hardware to calibrate the signal level. For the purposes it is currently used for, this is good enough.

signalLevelDefault: the harness’s default output signal level in dBm, as a floating-point number. This is the signal level used if the test case does not explicitly change the signal level.

signalLevelMax: the maximum signal level supported by the harness, in dBm, as a floating-point number. E.g. for one example modulator this is 0dBm

signalLevelMin: the minimum signal level supported by the harness, in dBm, as a floating-point number. For harnesses that support OpApps, this must be at least 9dBm less than signalLevelMax. E.g. for one example modulator this is -60dBm

signalLevelStep: the granularity that the signal level can be adjusted, in dBm, as a nonnegative floating-point number. Note that it is guaranteed that both signalLevelMax and signalLevelMin are achievable, so this cannot exceed (signalLevelMax – signalLevelMin). May not be exact due to floating-point rounding. This may be low (e.g. 0.5) if the modulator has a built-in output power control, but may be much larger (e.g. 9) if output power control is being done by manually connecting an attenuator.

NOTE: The supported signal levels are likely to be different for different modulation types (e.g. satellite vs terrestrial) and may be different for different multiplexes if the harness is using two modulators that are different models.

If ‘length’ is 1, then the properties above shall also be available on the playoutInformationObject directly. (I.e. if `length` is 1, then playoutInformationObject.transponderDeliveryType shall be the same as playoutInformationObject[0].transponderDeliveryType, but if ‘length’ is not 1 then playoutInformationObject.transponderDeliveryType should be undefined).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

If this method call fails, an empty object with no key/value pairs is returned.

7.2.5 JS-Function endTest()

This function indicates that the test case has completed.

```
void HbbTVTestAPI.endTest();
```

ARGS: None

This will end the test. No further calls to any API functions after this call shall have any effect on the test result.

The result of the test case is "PASSED" only in the following case:

- there were no calls to reportStepResult with result being *false* during the complete run of the test, and
- the analysis of all analyze function calls succeeded (or there were no analyze calls at all). There may not be any network connection at the time of the function call (or the Test Harness cannot be reached for other reasons), so this function may be implemented asynchronously. Therefore a test implementer must make sure that the network connection is available at the end of the test. (see reportStepResult documentation).

NOTE: This function could be internally implemented by calling reportStepResult with a stepId below 0, but this is not a requirement.

7.2.6 JS-Function reportStepResult()

This function reports a step result (succeeded step or failed step) back to the Test Harness. The Test Harness shall report the stepId, the result, and the comment in the test report. Test implementers should use this function to report the result of at least each major test step within the test. If no actual test is performed and only an informational message should be delivered, test implementers shall use the reportMessage function.

```
void HbbTVTestAPI.reportStepResult(
        stepId : integer,
        result : boolean,
        comment : String);
```

ARGS: **stepId:** the step number that has been performed. Shall be called with an integer value >= 0. stepId = 0 indicates that the test application has just started. stepId values shall be unique throughout the execution of each test case. They shall not be repeated. If a Test Harness receives a repeated value of stepId for a test case then the test shall fail.

There is no need for the stepId in the implementation to match the step values in the test specification procedure.

result: *true* if the step has completed successfully, *false* if the step has failed. To send a failure, no endTest() call is required. In this case, the stepId references the failing step. If the result is false, the server may not stop the application immediately (this may take a few seconds or even minutes, depending on the server). Therefore the testing application should ensure that no more reportStepResult calls are executed after this. Furthermore, the Test Harness shall ignore any reportStepResult calls received after a call with *false* result while executing a test.

comment: a comment from the test developer describing what the step actually does (e.g. a reference to the test procedure)

NOTE 1: The step number is usually in ascending order, but this is not a requirement. A test may skip test numbers and/or not report steps in ascending order. This is especially the case if multiple steps are executed in parallel.

NOTE 2: The test source code may have multiple calls to reportStepResult with a particular stepId value, as long as only one of these calls is executed when the test is run.

7.2.6.1 Reporting results when no network connection is available

There may not be any network connection at the time of the function call (or the Test Harness cannot be reached for other reasons), so this function may be implemented asynchronously. In any case, an implementation must make sure that all step results are reported back to the server as soon as network connection is up again in the order they were made. So there might be the following implementations:

EXAMPLE 1: synchronous communication to the server via a non-network communication path (e.g. a serial line connection)

EXAMPLE 2: asynchronous communication with the server (e.g. XmlHttpRequest via network) where all calls are stored in a FIFO queue, that is, one by one, reported to the server. If communication is not possible (e.g. network not available), the JS API implementation will continually try to report the step results in the queue in the background. This is why a test implementer must make sure that the network connection is available at the end of the test.

The following diagram shows the communication (and queuing) with the server in the case of an asynchronous communication with the server:

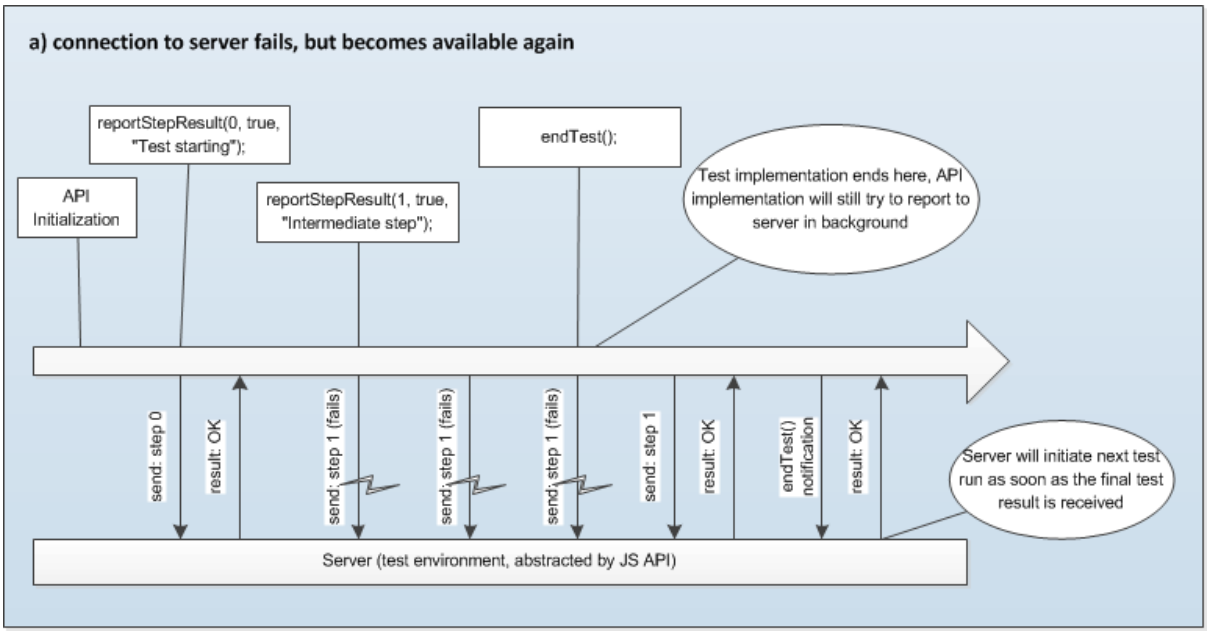


Figure 5: Communication between DUT and Test Harness when network is temporarily unavailable

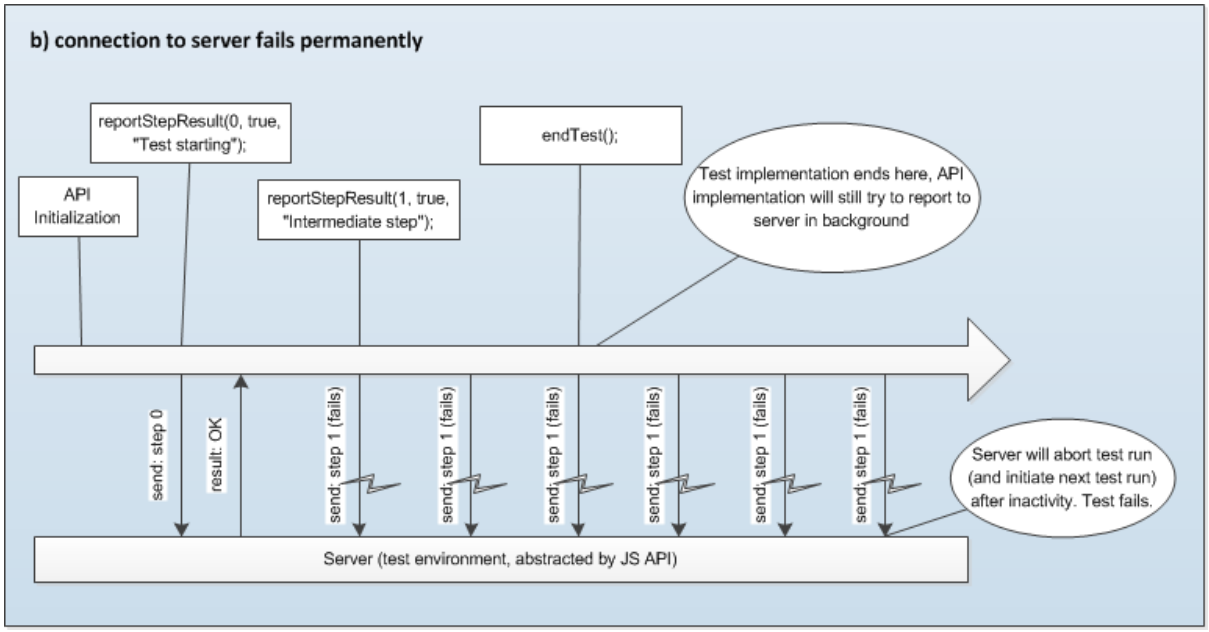


Figure 6: Communication between DUT and Test Harness when network is permanently unavailable

7.2.7 JS-Function reportMessage()

This function reports an informational message to the Test Harness. These messages usually serve as a hint to the test operator (e.g. “waiting 30 seconds”) or give more detailed information when debugging this test. These messages are only informational and do not form part of the test report. It is not specified how the Test Harness should handle these messages: They may be discarded, displayed, logged, etc.

```
void HbbTVTestAPI.reportMessage(
        comment : String);
```

ARGS: **comment:** the informational message as String.

NOTE: There may not be any network connection at the time of the function call (or the Test Harness cannot be reached for other reasons), so this function might be implemented asynchronously.

7.2.8 JS-Function waitForCommunicationCompleted()

As the function calls of the API defined in this document are asynchronous, a test implementation might need to make sure that all calls were successfully transmitted to the server and that the queue containing the step results to be reported to the server is now empty. This check should be performed before the network connection is switched off or before the test application destroys itself.

```
void HbbTVTestAPI.waitForCommunicationCompleted(
        callback : function,
        callbackObject : object);
```

ARGS: **callback/callbackObject:** a callback function to invoke when server communication queue is now empty (also see chapter 7.2.3).

NOTE: An implementation of this API which is based on manual interaction may immediately call the callback function when this function is called, as it does not have any communication queue. If a queue is present but empty, the callback function may also be called immediately (but may also be called asynchronously).

7.2.9 JS-Function manualAction()

This function instructs the test operator to carry out an action. This function should only be used if the requested action cannot be achieved using other API methods (e.g. sendKeyCode).

```
void HbbTVTestAPI.manualAction(
        check: String,
        callback : function or null,
        callbackObject: object);
```

ARGS: **check:** a textual description of the action required which will be presented to the test operator

callback/callbackObject: a callback function to invoke when the action has been completed by the test operator (also see chapter 7.2.3 Callbacks).

7.2.10 JS-Function getDUTOptions ()

This function allows the test case to query the Terminal Options and Optional Features that the tester claims the DUT supports and other settings, as configured in the test harness. This allows a Test Case to make fine-grained decisions about what to do based on what features the DUT supports and other settings. E.g. this API can be used if a large Test Case has a small bit that's different dependent on a DUT feature.

NOTE: where possible, this API should not be used. Instead, use the preconditions mechanism to arrange for a whole Test Case to be run or not based on the DUT's features.

To use this API, the test case must specify what options and settings it adapts to as described in 6.3.4.1 adaptsTo.

The API is a method:

```
void HbbTVTestAPI.getDUTOptions(
        callback : function,
        callbackObject);
```

This arranges for the callback to be called like this:

```
void callback(
        callbackObject,
        optionsData: OptionsData);
```

Where the optionsData object has the following methods defined:

```
bool optionsData.hasTerminalOption(
        optionName : string);

bool optionsData.hasFeature(
        optionName : string);

string optionsData.getSetting(
        settingName : string);
```

For the first two of these methods the parameter should be a single option name of the relevant type. Only +OPTION can be specified, not –OPTION.

NOTE: If you want –OPTION, call the function with +OPTION and negate the result in Javascript.

Both these methods return immediately, and have 3 possible results:

- If the option is not a single option name or was not listed in the relevant sub-tag of the <adaptsTo> tag, this is a test case bug, and the harness will cause the test case to fail automatically. For hasTerminalOption() the relevant sub-tag is <terminalOptions>. For hasFeature() the relevant sub-tag is <optionalFeatures>.
- If the option is supported, return true
- If the option is not supported, return false

The optionsData.getSetting() method returns immediately, and has two possible results:

- If the value of settingName is not a single value listed in the table in section 6.3.4.1 adaptsTo, or if the value was not listed in a <setting> sub-tag of the <adaptsTo> tag, this is a test case bug, and the harness will cause the test case to fail automatically.
- Otherwise, return the value corresponding to the setting as configured in the harness.

7.2.11 JS-Function endTestApp()

This function can be used to simplify ending the test in multiple-client test cases as described in 7.2.1.3 Multiple clients. The alternative is to use the APIs defined in the next section to coordinate having only one of the test applications call endTest().

```
void HbbTVTestAPI.endTestApp(
        numberOfApps : int);
```

Reports to the harness that, as far as the currently running application is concerned, the test case is complete and can pass, but the harness should wait for calls to this function from other applications before actually marking the test as passed. For the purposes of this function, each "application" could be a harness application, regular HbbTV app, OpApp, or Launcher OpApp page (the latter two only if Appendix D is implemented in the Test Harness).

numberOfApps indicates the total number of applications that should call this API. Must be an integer no less than 2. When that number of calls have been made to this API, the test ends (possibly passing) as if endTest() was called.

After calling this, the currently running JS execution environment (i.e. web page or harness-based test code) should not make any more HbbTVTestAPI calls, and should not make any calls to MultiClientComms.send. Other JS execution environments may continue to use those APIs.

It is a test case bug if a test case calls endTest() after endTestApp().

It is a test case bug if a test case calls endTestApp() with different numberOfApps.

7.2.12 JS-Functions for multiple-client communication

Some APIs are defined to allow multiple clients (e.g. harness-based test code and a regular HbbTV application running on the DUT, or an OpApp running on the DUT if Annex D is implemented) to co-ordinate amongst themselves.

```
void HbbTVTestAPI.multiClientConnect(
        myClientId : string,
        onConnect : function or null,
        onMessage : function or null,
        onFail : function,
        callbackObject : object,
        onClientJoinOrLeave: function or null or undefined);
```

Connects to the multi-client messaging system.

NOTE: The harness is likely to implement the multi-client messaging system via a websocket connection from each client to a server that is part of the harness. This call opens that websocket.

myClientId identifies this client. Recommended values are "OpApp" for OpApps, "BroadcastApp" for regular HbbTV apps running on the DUT, or "HarnessBased" for harness-based test code, but any string matching the Javascript regex /^[-a-zA-Z0-9_]{1,25}$/ is legal. Within a test case it should be unique.

If onConnect is non-null it is called once this client is listening:

```
void onConnect(
        callbackObject : object,
        comms : MultiClientComms,
        listOfOtherClients: string[]);
```

ARGS: **comms:** can be used to call comms.send(), see below.

listOfOtherClients: a list of clients that are currently connected to the multi-client messaging system, identified by the client ID values they passed as the myClientId argument to the multiClientConnect function.

If onMessage is non-null it is called whenever a message is received:

```
void onMessage(
        callbackObject : object,
```

```
        message,
        sourceClientId : string);
```

ARGS: **message:** The message that was sent. May be any JSON-compatible value (i.e. anything that JSON.parse() returns).

sourceClientId: the client ID that the sender of the message passed to multiClientConnect()

If onFail is non-null it is called if the connection cannot be established (in which case onConnect is never called) or (after a call to onConnect) if the connection is lost. This should not normally happen, unless the test case disables the terminal's network connection. Call is:

```
void onFail(
        callbackObject : object,
        reason : string);
```

ARGS: **reason:** A human-readable string describing the error. E.g. the test case may choose to fail the test and use this string as part of the failure reason.

Each client (i.e. each Javascript execution context) must not try to open more than one connection to the multi-client messaging system at any time. Once multiClientConnect() has been called it must not be called again by that client unless or until the onFail callback has been called. If this rule is broken then the harness should fail the test case automatically with an error.

```
void MultiClientComms.send(
        toClientId : string,
        message);
```

Sends a message to another client. Note that this function is on the object returned via the onConnect callback, so you must first call multiClientConnect() and wait for the onConnect() callback.

ARGS: **toClientId:** identifies the client that should receive the message. Same legal values as the myClientId argument to multiClientConnect()

message: may be any JSON-compatible value (i.e. anything that JSON.stringify() accepts).

The message is sent to every client with a client ID that matches toClientId. If there are multiple clients connected using the same client ID, then it gets sent to all of them. If there are no clients with that client ID then that is a test case bug and the harness should fail the test case automatically with an error. You can send messages to a client that has onMessage set to null, even though the client ignores the message.

onClientJoinOrLeave: A callback function to invoke when another client joins the multi-client messaging system (by calling multiClientConnect()) or disconnects from the multi-client messaging system (e.g. due to the application that called multiClientConnect() being terminated, or the network connection going down). Prototype:

```
void onClientJoinOrLeave(
        callbackObject : object,
        clientId : string,
        join : boolean);
```

ARGS: **clientId:** the identity of the client that connected or disconnected. This is the string that client passed as the myClientId argument to the multiClientConnect function.

join: true if the client joined the network (i.e. connected to the server), false if the client left the network (i.e. disconnected from the server).

7.2.13 Extended Settings

The test suite can have files RES/META/ExtraInfoRequired/*.xml. This has a list of extra settings that need to be set by the tester before running the test suite, in XML format. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<eir:extraInformationRequired xmlns:eir="http://www.hbbtv.org/2018/ExtraInfoRequired">
  <setting key="speciesOfET">
     <string minlength="1"/>
```

```
     <title>
        Species of Extra Terrestrial
     </title>
     <description>
        The species of entity that will be using this terminal.
     </description>
  </setting>
  <setting key="numberOfHeads">
     <number min="0" step="1"/>
     <default>1</default>
     <title>
        Number of heads
     </title>
     <description>
        Number of heads that the user of this terminal has.  Typically 1, although 2 for
Zaphod Beeblebrox and 3 for Cerberus.
     </description>
  </setting>
</eir:extraInformationRequired>
```

The harness reads all these files and merges all the data together to determine what settings to show the user. The user then configures the settings in the harness.

NOTE: Storing this metadata in files in the test suite allows the list of bilateral agreement settings to be changed without having to update this specification and release a new test harness.

NOTE: Having this per-testsuite rather than per-test allows changes to the descriptions etc in a single central place, and avoids worrying about "what if one test has a description that's different from the others".

NOTE: the filename, schema and JS API carefully does not have "bilateral agreement" in the name because we expect this to be more generally useful for extra settings that other test suites require.

NOTE: The file should be named after the test package that uses the settings.

The key is used to uniquely identify the setting, and must be unique (across ALL these XML files). The title and description are mandatory. The title is used to identify the setting to the user, so it should be unique, and should be kept short – generally 1-4 words. The description is used to explain the setting to the user, so can be long as necessary (although good UI design is to keep things as short as possible, but no shorter).

Settings can be a string or a number. For numbers, the "min", "max", and "step" attributes can be set – these have the same values and meanings as in the HTML5 number input tag. For strings, the "minlength", "maxlength" and "pattern" attributes can be set, and have the same values and meanings as in the HTML5 text input tag.

A default value can optionally be specified. If specified, the default must validate as a valid value.

A string with no "minlength" must have a default specified.

NOTE: This is because the harness will typically use an empty string as "not specified" internally. But if there is no minimum length then an empty string is a valid value.

7.2.14 JS-Function getExtendedSetting

Within test case, value of each extended setting item as described in the previous section can be fetched using following test API function:

```
void HbbTVTestAPI.getExtendedSetting(
        key : String,
        callback : function,
        callbackObject : Object);
```

ARGS: **key:** Key of the extended setting item to be obtained.

callback: A callback function to invoke when the call completes, providing callbackObject as first parameter, and string which represents value of that item obtained from user by test Harness as the second parameter.

callbackObject: Optional object to be passed back to the invoker as the first parameter of the callback function.

If the requested key is not listed in ExtraInfoRequired.xml, or is not listed in <usesExtendedSettings> in the testcase XML for the running test, or if the setting is not set and does not have a default value, then the test harness shall fail the entire test automatically.

7.2.15 JS-Function getHarnessHttpsServerUri ()

```
string HbbTVTestAPI.getHarnessHttpsServerUri();
```

If the test case implementation.xml file includes the <httpsServerConfig/> XML tag, then this immediately returns the URI to the TLS server used to serve OpApps. This shall be of the form "https://<domainname>/" or "https://<domainname>:<port>/" – note that the HTTPS protocol and trailing slash are always included.

The return value of this function is a constant, its value shall not be changed by the test harness across the scope of a single test run.

This function returns immediately, without waiting for a network request. It is even available if the terminal does not have network access and has never had network access (and has presumably loaded the application from DSMCC).

NOTE: Harness authors, please note: The above rules mean that the return value must be part of testsuite.js, either hardcoded or substituted in.

If the test case does not specify the <httpsServerConfig/> XML tag, then it must not call this function.

7.3 APIs Interacting with the Device under Test

7.3.1 JS-Function initiatePowerCycle()

This function initiates a power cycle of the DUT. It will first wait a user-defined amount of time, then switch off the DUT, then wait a short period of time, then switch it back on again. The Test Harness shall ensure that the DUT finally returns to the pre-defined state. The period for which the DUT is switched off is not defined, but should be chosen to ensure a clean power cycle of the DUT.

```
void HbbTVTestAPI.initiatePowerCycle(
        delaySeconds : int,
        type : String,
        callback : function or null,
        callbackObject : object);
```

ARGS: **delaySeconds:** the number of seconds after which the device is switched off. These seconds start counting as soon as the callback function is called, if it is set to 0, the reset may occur while the application is still handling the callback function. If delaySeconds < 0 then the test shall fail.

type: one of the following strings:

- "STANDBY": the device will go to standby mode – if the device does not support standby mode, POWERCYCLE will apply.
- "POWERCYCLE": the device will be physically disconnected from power supply – if the device has a built-in power supply (e.g. battery), the power cycle is defined by the device.

If any other string is received the test shall fail.

callback/callbackObject: a callback function to invoke when the power cycle request was received (also see chapter 7.2.3 Callbacks). The power cycle delay shall start when the callback has completed.

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

Calling this function will cause the current application to be stopped. After reboot, the device needs to be transferred to the pre-defined state (see chapter 7.2.2), so the AUTOSTART application signalled on the initial service will be started.

7.3.2 JS-Function sendKeyCode()

This function requests a key code to be sent to the DUT. This can be implemented by directly sending IR codes to the DUT. Alternatively, this can be implemented by a manufacturer specific API.

```
void HbbTVTestAPI.sendKeyCode(
        keyCode : String,
        durationSeconds : int,
        callback : function or null,
        callbackObject : object);
```

ARGS: **keyCode:** a string describing the key to send: any VK_* code in the table in OIPF 2.3 volume 5a Web Standards TV Profile section 6.2, except for VK_UNDEFINED (because that's not actually a key). It will also be legal to use this API to send the other VK_* codes defined in table 17 in the OpApps specification. If any other keyCode string is requested then the test shall fail.

durationSeconds: the number of seconds for which the key is held down continuously, or a value of 0 (zero) for a single key press event. If durationSeconds < 0 then the test shall fail.

callback/callbackObject: a callback function to invoke when the key code was actually sent to the receiver (also see chapter 7.2.3 Callbacks).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.3.3 JS-Function analyzeScreenPixel()

This function analyzes the current screen and checks if a specified pixel on that screenshot matches a given reference colour.

```
void HbbTVTestAPI.analyzeScreenPixel(
        stepId : integer,
        comment : String,
        posx : integer,
        posy : integer,
        referenceColor : String,
        callback : function or null,
        callbackObject : object);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

posx: the horizontal position of the pixel to analyze within safe border (128-1152)

posy: the vertical position of the pixel to analyze within safe border (36-683)

referenceColor: the reference colour that the pixel should match. The String must start with a hash (#) followed by a 2-digit case insensitive hexadecimal representation of the red, green, and blue colour (e.g. #ff0000 for red colour).

callback/callbackObject: a callback function to invoke when the call completes (also see chapter 7.2.3 Callbacks). The analysis may not have taken place at the time the callback is invoked. The analysis result is not passed back to the application. A failed analysis must cause the complete test

to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screen shot.

The method for pixel colour matching is not defined. The following tolerances are defined for analysis:

- The area of pixels analyzed may be up to +/- 10 pixels in each direction of the specified pixel position
- The pixel colour value analyzed may be up to +/- 80/255 of the specified colour value for each colour component

EXAMPLE: This function may be implemented either for manual interaction or for automated processing. During analysis, the implementation may modify the HTML DOM (e.g. add a black layer and a cross hair on top of the screen and removing it after analysis is finished). This might trigger an Application.show() call. If only manual processing is desired, an API application could call analyzeScreenExtended("Is colour of Pixel at Pos. "+posx+"/"+posy+" similar to reference colour "+referenceColor+"?", callback, callbackObject).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.3.4 JS-Function analyzeScreenExtended()

This function analyzes the current screen and performs an extended check on the currently displayed screen content. As this check is described by a string, the check most probably has to be done by a human being, although a test environment may provide an automated implementation of this check. This call should be avoided by test case implementers, if possible.

```
void HbbTVTestAPI.analyzeScreenExtended(
        stepId : integer,
        comment : String,
        check : String,
        callback : function or null,
        callbackObject : object,
        image_relative_url : String or null or undefined);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

check: a textual description detailing which test to perform. This is the only criteria that shall be used for the assessment of this analysis call.

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screen shot.

image_relative_url: specifies a path, relative to the testcase XML file, to a reference image file for the tester to refer to when performing the analysis. This is optional, if it is not specified or undefined or null then there is no image to display. If specified, the reference image file shall have one of the file extensions from the table below.

The same pixel and colour tolerances apply as specified in the analyzeScreenPixel function.

EXAMPLE: This function may be implemented by first taking the screenshot and then performing an offline analysis at a later point in time.

NOTE: Even when an image is specified, the tester is still required to follow the instructions about how to compare the images, it is NOT just a straight image comparison. E.g. “do the subtitles look like the image shown here” when the terminal is showing moving video behind the subtitles. As a result, this is NOT suitable for full automation, but it can handle trickier comparisons than analyzeScreenAgainstReference()

The supported reference image formats are:

Format	File extension	MIME type
PNG image	.png	image/png
JPEG image	.jpg	image/jpeg

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.3.5 JS-Function analyzeAudioFrequency()

This function analyzes the current audio and performs a frequency check on that data.

```
void HbbTVTestAPI.analyzeAudioFrequency(
        stepId : integer,
        comment : String,
        channelId : integer,
        referenceFrequency : int,
        callback : function or null,
        callbackObject : object);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

channelId: the channel in the referenced audio that is being analyzed. The following channel IDs are supported:

- 1: left channel
- 2: right channel
- 3: centre channel
- 4: rear left channel
- 5: rear right channel

If the channelId is not implemented (channelId < 1 or channelId > the number of audio channels in the sample) then the test shall fail.

referenceFrequency: the reference frequency in Hz. Allowed values are 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000. Other values shall cause the test to fail.

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screen shot.

Frequencies should be detected that have a minimum signal level between -50dB and 0dB, assuming no attenuation by the DUT. The tolerance on frequency detected shall be +/- 10%.

The method of audio frequency matching is not defined.

EXAMPLE 1: If the implementation uses a standard one third octave analyzer, ten different one-third octave band frequencies fall into the above defined range (500 Hz, 630 Hz, 800 Hz, 1000 Hz, 1250 Hz, 1600 Hz, 2000 Hz, 2500 Hz, 3150 Hz, 4000 Hz), which should be reasonable distinguishable.

EXAMPLE 2: This function may be implemented either for manual interaction or for automated processing. If only manual processing is desired, an API application could call analyzeAudioExtended("Do you hear a tone with a frequency of about "+referenceFrequency+" Hz on channel "+channelId+"?", callback, callbackObject).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: This function may be implemented by first making an audio capture and performing the analysis at a later point in time.

7.3.6 JS-Function analyzeAudioExtended()

This function analyzes the current audio and performs an extended check on that data (as this check is described by a String, the check most probably has to be done by a human being, although a test environment may provide an automated implementation of this check). This call should be avoided by test case implementers, if possible.

```
void HbbTVTestAPI.analyzeAudioExtended(
        stepId : integer,
        comment : String,
        check : String,
        callback : function or null,
        callbackObject : object,
        duration : integer or undefined,
        delay : integer or undefined,
        audio_relative_url: String or null or undefined,
        audio_mime_type : String or null or undefined);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

check: a textual description detailing which test to perform on the audio data. This is the only criterion that shall be used for the assessment of this analysis call. An empty or null string shall cause the test to fail.

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screen shot.

duration: duration of recording in seconds. The argument is optional; if it is not present then the default is 10 seconds.

delay: delay before recording in seconds. The argument is optional; if it is not present then the default is 0 seconds.

audio_relative_url: specifies a path, relative to the testcase XML file, to a reference audio file for the tester to refer to when performing the analysis. This is optional, if it is not specified or undefined or null then there is no audio file to refer to.

audio_mime_type: specifies the MIME type of the audio file. This is optional, if a string is specified for audio_relative_url then audio_mime_type must be a string specifying the MIME type, otherwise this parameter must be either not specified or undefined or null.

This function analyzes (or records for later analysis) the audio over the following period:

- Starting between (delay + 0) and (delay + 2) seconds after the API is called (although this may be delayed if other Test API calls are in progress when this API is called).
- Ending (duration) seconds after the analysis (or recording) started.

The described check shall not require audio data from outside the specified period. This allows the audio to be recorded and then processed later on.

The callback may be called much later than the end of the analysis period, e.g. if the harness is waiting for a user to enter the result, or if the harness is doing non-real-time audio compression or some automated analysis.

The method of audio analysis is not defined.

Test implementers should consider the audio content to be analyzed, assuming there will be a human tester.

E.g., if implementation uses a microphone for audio capture, provided results might not be very exact. Inaccuracy of audio capture may be up to +/- 1000 Hz. Only frequencies should be detected that have a minimum loudness of at least 50% of the maximum allowed loudness.

NOTE: This function may be implemented by first making an audio capture and performing the analysis at a later point in time.

The supported reference audio formats are:

Format	**File extension**	**MIME type**
MP3 audio	.mp3	audio/mpeg
MP4 container containing only AAC audio (no video)	.m4a or .mp4a or .aac	audio/mp4, optionally with the codecs parameter (RFC4281)

NOTE: These formats have been chosen to be compatible with the <audio> tag in all major desktop web browsers. The file extensions are chosen to match the list in section 5.2.1.1Web Server

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.3.7 JS-Function analyzeVideoExtended()

This function analyzes the current video and performs an extended check on that data. As this check is described by a string, the check most probably has to be done by a human being, although a test environment may provide an automated implementation of this check. This call should be avoided by test case implementers, if possible.

```
void HbbTVTestAPI.analyzeVideoExtended(
        stepId : integer,
        comment : String,
        check : String,
        callback : function or null,
        callbackObject : object,
        duration : integer or undefined,
        delay : integer or undefined,
        image_or_video_relative_url : String or null or undefined,
        image_or_video_mime_type : String or null or undefined);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

check: a textual description detailing which test to perform on the video data. This is the only criteria that shall be used for the assessment of this analysis call. An empty or null string shall cause the test to fail.

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed

analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording.

duration: duration of recording in seconds. The argument is optional; if it is not present then the default is 10 seconds.

delay: delay before recording in seconds. The argument is optional; if it is not present then the default is 0 seconds.

image_or_video_relative_url: specifies a path, relative to the testcase XML file, to a reference image or video file for the tester to refer to when performing the analysis. This is optional, if it is not specified or undefined or null then there is no audio file to refer to.

image_or_video_mime_type: specifies the MIME type of the image or video file. This is optional, if a string is specified for image_or_video_relative_url then image_or_video_mime_type must be a string specifying the MIME type, otherwise this parameter must be either not specified or undefined or null.

This function analyzes (or records for later analysis) the video over the following period:

- Starting between (delay + 0) and (delay + 2) seconds after the API is called (although this may be delayed if other Test API calls are in progress when this API is called).
- Ending (duration) seconds after the analysis (or recording) started.

The described check shall not require video data from outside the specified period. This allows the video to be recorded and then processed later on.

The callback may be called much later than the end of the analysis period, e.g. if the harness is waiting for a user to enter the result, or if the harness is doing non-real-time video compression or some automated analysis.

The supported reference image and video formats are:

Format	**File extension**	**MIME type**
PNG image	.png	image/png
JPEG image	.jpg	image/jpeg
MP4 container containing H.264 video only (no audio)	.mp4	video/mp4, optionally with the codecs parameter (RFC4281)
MP4 container containing H.264 video and MP3 audio		
MP4 container containing H.264 video and AAC audio		

NOTE: These formats have been chosen to be compatible with the <img> and <video> tags in all major desktop web browsers. The file extensions are chosen to match the list in section 5.2.1.1 Web Server.

If specifying a reference video, it may optionally have sound, which should be ignored for the analysis. (This allows reusing existing video assets).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: This function may be implemented by first making a video capture and performing the analysis at a later point in time.

7.3.8 JS-Function analyzeManual()

This function instructs the test operator to carry out the analysis described in the check parameter, and record the result. This function should only be used if the analysis cannot be achieved using other API methods.

```
void HbbTVTestAPI.analyzeManual(
        stepId : integer,
        comment : String,
        check: String,
        callback : function or null,
        callbackObject: object);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step

check: a textual description of the analysis that must be done which will be presented to the test operator

callback/callbackObject: a callback function to invoke when the power cycle request was received (also see chapter 7.2.3 Callbacks).

7.3.9 JS-Function selectServiceByRemoteControl()

This function requests to select a service by a sequence of (mainly numeric) key codes that is sent to the DUT. This shall be implemented by directly sending IR codes (or equivalent) to the DUT (automated or by a request to the tester to do so manually, similar to reportMessage). The service is identified by its name. The service selection shall switch directly from the current to the new service.

```
void HbbTVTestAPI.selectServiceByRemoteControl(
        serviceName : String,
        callback : function or null,
        callbackObject : object);
```

ARGS: **serviceName:** the name of the service to select, e.g. "ATE Test11". The name shall be the service name as defined in the service descriptor of the target service. If the service is not signalled in the current broadcast then the function shall fail.

callback/callbackObject: a callback function to invoke when the key codes were actually sent to the receiver (also see chapter 7.2.3 Callbacks).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

This function should not be used to select a radio service when a TV service is selected or vice-versa, as this could require a change of service lists and could involve intermediate service selections (e.g. to the last active service of the new list).

Before calling this API, the test case should ensure no HbbTV applications are requesting the NUMERIC KeySet.

NOTE: Although the naming of this function indicates that the service change must be performed by emulating a remote control (i.e. sending IR commands to an IR receiver), there is no specific requirement to do this. If alternative methods of service selection are available then these are valid.

7.3.10 JS-Function sendPointerCode()

This function requests a pointer device to be moved to a location on the screen of the DUT and optionally a pointer code to be sent to the DUT. This can be implemented by manually moving a pointer device and sending codes to the DUT. Alternatively, this can be implemented by a manufacturer specific API.

```
void HbbTVTestAPI.sendPointerCode(
        posx : integer,
        posy : integer,
        pointerCode : String,
```

```
        callback : function or null,
        callbackObject : object);
```

ARGS: **posx:** the horizontal position of the pixel to move to (0-1279)

posy: the vertical position of the pixel to move to (0-719)

pointerCode: a string describing the code to send once the pointer is in position: “P_CLICK”, “P_DBLCLICK”, “P_DOWN”, “P_UP”, or “NONE” if no code is to be sent. If any other pointerCode string is requested then the test shall fail. For touchpad-type devices, code “P_CLICK” shall be sent by the harness if pointerCode "NONE" is specified, to better simulate real user interaction with a touchpad where pointing and clicking are both represented by a single tap.

callback/callbackObject: a callback function to invoke when the pointer has been moved and any requested pointer code was sent to the DUT (also see chapter 7.2.3 Callbacks).

The tolerance on posx and posy is +/- 10 pixels.

7.3.11 JS-Function moveWheel()

This function requests the wheel device to be moved relative to the current position. This can be implemented by manually moving a wheel device. Alternatively, this can be implemented by a manufacturer specific API.

```
void HbbTVTestAPI.moveWheel(
        deltax : integer,
        deltay : integer,
        deltaz : integer,
        deltaMode : String,
        callback : function or null,
        callbackObject : object);
```

ARGS: **deltax:** the relative horizontal position to move by

deltay: the relative vertical position to move by

deltaz: the relative depth position to move by

deltaMode: the measurement mode, one of: “WHEEL_DELTA_PIXEL”, “WHEEL_DELTA_LINE”, “WHEEL_DELTA_PAGE”

callback/callbackObject: a callback function to invoke when the key code was actually sent to the receiver (also see chapter 7.2.3 Callbacks).

7.3.12 JS-Function analyzeScreenAgainstReference()

This function allows comparison of an area of the DUT screen against a reference image. The comment gives additional information on the content of the expected area which may be used for the comparison along with the reference image. Additionally, the reference could be used to perform machine-based comparison of captured DUT output in the specified area of the image.

```
void HbbTVTestAPI.analyzeScreenAgainstReference(
        stepId : integer,
        comment : String,
        referenceImage : String,
        areaOfInterest : Object,
        callback : function or null,
        callbackObject : Object);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment describing to the tester content which is expected to appear in the specified area. The area of interest is specified as a separate parameter.

referenceImage: path to a reference image relative to test folder. The reference image should be a complete capture of the screen, although the function will consider only the area of interest. The image shall be in PNG format and the filename shall end in '.png'

areaOfInterest: JS object which contains pixel coordinates of the screen region that should be compared (top, left, bottom, right). Pixel coordinates are zero-based.

callback/callbackObject: a callback function to invoke when the call completes (also see chapter 7.2.3 Callbacks). The comparison may not have taken place at the time the callback is invoked. The comparison result is not passed back to the application. A failed comparison must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screenshot.

7.3.13 JS-Function analyzeTextContent()

This function analyzes text in the specified area (parameter areaOfInterest) searching for specified text (parameter expectedText). In case of manual execution, tester is instructed which text to search if specified area. Additionally, reference image is presented to the tester to further clarify appearance of the text. In case of test automation, OCR algorithm may be used to extract and compare text content from captured DUT output.

```
void HbbTVTestAPI.analyzeTextContent(
        stepId : integer,
        comment : String,
        expectedText : String,
        areaOfInterest : Object,
        referenceImage : String,
        callback : function or null,
        callbackObject : Object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing to the tester (if test is performed manually) text which is expected to appear in specified area. Area of interest is specified as a separate parameter.

expectedText: text content expected to be found in given region. If text content is found, test passes.

areaOfInterest: JS object which contains pixel coordinates of the screen region in which text should appear (top, left, bottom, right). Pixel coordinates are zero-based.

referenceImage: path to the image relative to test folder. Reference image should be a complete capture of the screen, although function will consider only area of interest. The image shall be in PNG format and the filename shall end in '.png'. This image may be presented to the tester by the test harness in order to provide further clarification of expected text appearance on the screen.

callback/callbackObject: a callback function to invoke when the call completes (also see chapter 7.2.3 Callbacks). The analysis may not have taken place at the time the callback is invoked. The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken screenshot.

7.3.14 JS-Function analyzeAudioVideoExtended ()

This function analyses the current audio and video at the same time, and performs an extended check on the data (existing APIs analyzeAudioExtended and analyzeVideoExtended performed such analysis separately). As this check is described by a strin, the check most probably has to be done by a human being, although a test environment may provide an automated implementation of this check. This call should be avoided by test case implementers, if possible.

```
void HbbTVTestAPI.analyzeAudioVideoExtended(
        stepId : integer,
```

```
comment : String,
check : String,
callback : function or null,
callbackObject : object,
duration : integer or undefined,
delay : integer or undefined,
image_or_video_relative_url : String or null or undefined,
image_or_video_mime_type : String or null or undefined,
audio_relative_url : String or null or undefined,
audio_mime_type : String or null or undefined);
```

ARGS: **stepId:** the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

check: a textual description detailing which test to perform on the video data. This is the only criteria that shall be used for the assessment of this analysis call. An empty or null string shall cause the test to fail.

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording.

duration: duration of recording in seconds. The argument is optional; if it is not present then the default is 10 seconds.

delay: delay before recording in seconds. The argument is optional; if it is not present then the default is 0 seconds.

image_or_video_relative_url: specifies a path, relative to the testcase XML file, to a reference image or video file for the tester to refer to when performing the analysis. This is optional, if it is not specified or undefined or null then there is no audio file to refer to. If image_or_video_relative_url specifies a video that has sound, then both the audio_... parameters shall not be used and shall be passed as null.

image_or_video_mime_type: specifies the MIME type of the image or video file. This is optional, if a string is specified for image_or_video_relative_url then image_or_video_mime_type must be a string specifying the MIME type, otherwise this parameter must be either not specified or undefined or null.

audio_relative_url: specifies a path, relative to the testcase XML file, to a reference audio file for the tester to refer to when performing the analysis. This is optional, if it is not specified or undefined or null then there is no audio file to refer to.

audio_mime_type: specifies the MIME type of the audio file. This is optional, if a string is specified for audio_relative_url then audio_mime_type must be a string specifying the MIME type, otherwise this parameter must be either not specified or undefined or null.

This function analyzes (or records for later analysis) the video over the following period:

- Starting between (delay + 0) and (delay + 2) seconds after the API is called (although this may be delayed if other Test API calls are in progress when this API is called).
- Ending (duration) seconds after the analysis (or recording) started.

The described check shall not require video data from outside the specified period. This allows the video to be recorded and then processed later on.

The callback may be called much later than the end of the analysis period, e.g. if the harness is waiting for a user to enter the result, or if the harness is doing non-real-time video compression or some automated analysis.

Test implementers should consider the audio and video content to be analyzed, assuming there will be a human tester.

E.g., if implementation uses a microphone for audio capture, provided results might not be very exact. Inaccuracy of audio capture may be up to +/- 1000 Hz. Only frequencies should be detected that have a minimum loudness of at least 50% of the maximum allowed loudness.

For the supported reference audio formats, see 7.3.6 JS-Function analyzeAudioExtended()

For the supported reference image and video formats, see 7.3.7 JS-Function analyzeVideoExtended()

The reference file(s) can be:

- No reference files at all (all null or undefined), or
- video with sound, or
- video without sound plus a separate audio file, or
- image plus a separate audio file

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: This function may be implemented by first making a video capture and performing the analysis at a later point in time.

7.4 APIs communicating with the Playout Environment

7.4.1 Playout definition

Each test must have at least one definition of a playout set. A playout set comprises the following information:

- Transport streams to play out (independent of delivery type: e.g. DVB-S, DVB-C, DVB-T)
- Definition for AIT tables (optional, if not defined, they must be inside transport streams)
- Definition of DSM-CC carousels
- Other settings (e.g. network connection down)

The playout set definition with id "1" is the initial playout set and shall be active when the test starts. The referenced AIT should start the actual test application. Some tests may require switching between multiple playout sets. The switching is either done after a specified amount of time (timeout attribute in a playout set specifies after how many seconds to automatically switch to the next playout set) or after an API call from the test (see changePlayoutSet function below).

Each test must have a XML definition file called "implementation.xml" residing in the test directory defining all the requirements of the test:

```
<testimplementation id="<testcaseid>">
    <playoutsets>
        <playoutset id="<number>" definition="<rel_filename>"
         [timeout="<seconds>"] />+
    </playoutsets>
</testimplementation>
```

The XML file "implementation.xml" must validate against the "testImplementation.xsd" XML schema as defined in /SCHEMAS/testImplementation.xsd of the Test Suite.

When the timeout occurs (playout is played out for the specified time in seconds and no changePlayoutSet() call was made), the next playout set to be played out is determined as follows: the id of the current playout set is incremented by one, and the playout set with that new id is played out. If no such playout set exists, the playout set with id 1 is played out.

When changing playout sets (especially the ones containing audio/video), no seamless switching is required: continuity counter errors, PCR continuity problems, incomplete sections and/or tables may occur shortly after

time of switching. However, the overall signal should never be interrupted, and the version numbers of changed tables need to be modified to indicate the change to the DUT. As audio/video might show some artefacts, test case implementers should not perform any broadcast audio/video checks up to 5 seconds after a switch occurs.

7.4.1.1 Minimum timeout

A test harness may optionally have a watchdog timer, which automatically fails or aborts the test if there is no test case activity (defined below) for a certain period of time. This is intended to detect test cases that have failed in unexpected ways, especially where the terminal has crashed or hung.

For the purposes of this section, a reportXxx or analyzeXxx API call counts as test case activity that resets the watchdog timer.

The following states temporarily disable the watchdog timer:

- the DUT's network connection being disabled due to the current playoutset specifying <networkConnection available="NO"/>.
- the DUT rebooting, when triggered via initiatePowerCycle(). (The test harness may use a separate timer to fail the test if the reboot doesn't work).
- analyzeXxx API calls that take a significant amount of time. E.g. video capture, manual analysis steps.
- other API calls that require a human to do a manual action. E.g. sendKeyCode if the tester is required to manually press a key on the remote and indicate they have done that.

When the above states no longer apply, the watchdog timer is automatically reset and re-enabled. Ending those states counts as test case activity that resets the watchdog timer.

By default, the watchdog timer shall have a timeout of at least 2 minutes, although harnesses may use a higher value and may make this configurable.

A tag is defined that configures the harness with a minimum timeout that shall be respected. If this tag is used, the watchdog timer's timeout shall be at least as long as the value specified.

```
<minimumTimeout duration="PT5M"/>
```

The required "duration" attribute is a duration in ISO 8601 format. Only seconds, minutes, hours and days are supported. The seconds field may specify up to 3 digits after the decimal point for millisecond precision. The harness may round this value up to whatever granularity the harness supports (e.g. whole seconds or whole minutes). This value must be positive and nonzero.

Examples include:

- "PT5S" = 5 seconds
- "PT5.234S" = 5.234 seconds
- "PT5M" = 5 minutes
- "PT5M6S" = 5 minutes and 6 seconds
- "PT4H" = 4 hours
- "PT4H5M5.234S" = 4 hours 5 minutes and 5.234 seconds
- "P1D" = 1 day
- "P1DT5H5M5.234S" = 1 day and 5 hours 5 minutes and 5.234 seconds

For the purpose of this tag, a "day" always means 24 hours, even if the clocks go forward or back due to DST changes while the test is running.

Test cases that need to run for more than 2 minutes between reportXxx or analyzeXxx API calls must use this tag, **unless** the only time the test case does that is when it triggers one of the states listed above that temporarily disables the watchdog.

Note that this is a **minimum** timeout, so a harness may choose to use a longer timeout or no timeout, and the harness may choose to reset the watchdog timer upon other activity not listed here. Test cases that actually depend on a timeout should use a Javascript timer, they should not depend on the watchdog.

7.4.2 Relative file names

File names are always relative to the XML file containing them. When the test is executed, the complete test directory is available on the web server in a directory called “TESTS”. On the same level, there is a “RES” directory including the general resource files from the test suite. In addition, the “RES” directory will also contain a file called “testsuite.js” containing the implementation of the test suite API defined in this document.

7.4.3 Playout set definition

A playout set as defined in the test definition must be an XML file. This file defines all the transport stream files, AITs, DSM-CCs that need to be multiplexed to the final test signal played out. The transport streams are included as MPTS files referenced by relative file names.

```
<playoutsetdefinition>
    <transportstream
     file="<rel_filename>" file="<bitspersec>">+
        <pid src="<pid0-8190>" dst="<pid0-8190>"
             description="<text>" />*
        <!-- a PID in the transport stream will only be played
             out if it is listed here. src is the PID within
             the transport stream file. dst is the PID played
             out. Note: the test itself does not include any
             NIT play out. A modulation-type specific NIT is
             inserted later on by the playout server. -->
    </transportstream>
    <generatedData>
        <nitPid bitrate="<bps>">
            <nit src="<rel_filename>"/>
            <nitOther src="<rel_filename>"/>
        </nitPid>?
        <sdtAndBatPid bitrate="<bps>">
            <sdt src="<rel_filename>"/>?
            <bat src="<rel_filename>" enabled="auto"?/>*
        </sdtAndBatPid>?
        <ait pid="<pid0-8190>" src="<rel_filename>"
             bitrate="<bps>" version="<0-7>" />+
        <dsmcc pid="<pid0-8190>" association_tag="<0-255>"
               source_folder="<dir>" bitrate="<bps>"
               version="<0-255>" carousel_id="<0-255>">*
            <directory src="<rel_dirname>" dst="<name>" />*
            <file src="<rel_filename>" dst="<name>" />*
            <streamEvent src="<rel_filename>" dst="<name>" />?
        </dsmcc>
    </generated-data>
    <networkconnection available="YES|NO" />
</playoutsetdefinition>
```

The playout set definition XML files must validate against the “playoutsetDefinition.xsd” XML schema as defined in /SCHEMAS/playoutsetDefinition.xsd of the Test Suite. The referenced NIT XML file(s) must validate against the “nit.xsd” XML schema as defined in /SCHEMAS/nit.xsd of the Test Suite, and must use the <nit> element from that schema as the root element. The syntax of the NIT XML file is defined in 7.4.4.3.3. Any referenced SDT XML file must validate against the “nit.xsd” XML schema as defined in /SCHEMAS/nit.xsd of the Test Suite, and must use the <sdt> element from that schema as the root element. The syntax of the BAT XML file is defined in 7.4.4.3.5. Any referenced BAT XML file must validate against the “nit.xsd” XML schema as defined in /SCHEMAS/nit.xsd of the Test Suite, and must use the <bat> element from that schema as the root element. The syntax of the BAT XML file is defined in 7.4.4.3.4. The referenced AIT table must validate against the AIT XML format as described in TS 102 809 [4], chapter 5.4. The referenced StreamEvent description file must validate against the StreamEvent XML format as described in TS 102 809 [4], chapter with amendments in HbbTV Technical Specification section 9.3.1 [1] [20]. All other referenced data is in binary format.

If a <nit> tag is being used, and no <nitOther>, then the <nitPid> tag is optional – the <nit> tag can go directly inside <generatedData> in this (common) case. If <nitPid> is omitted, then the bitrate attribute can be specified on the <nit> tag directly.

The <bat> element shall only use the ‘enabled’ attribute if the test case is testing Operator Applications under Annex D, as described inD.4.1 Playoutset extensions.

If applicable, the generated SDT and BAT PID shall repeat all the SDT and BAT table(s) specified in rotation. The bitrate specifies the total TS bitrate for that PID. If the bitrate is not specified, then the harness shall calculate the bitrate needed to repeat all the SDT and BAT table(s) once per second.

When creating an AIT, you can assume that the test is available via a pre-defined URL. For more information, see chapter 5.1 Test Environment of this document.

NOTE: The played out AIT version number is not necessarily identical to the version number specified in the playout set definition XML file, as the test harness may add an offset (0, 8, 16, or 24) to that version number, depending on the test run. However, the offset is constant during the run of a single test case.

All bitrates are specified in bits per second (as integer values). Data should be played out with continuous bitrates (no bursts). Due to packaging reasons, the bitrate may vary +/- 1504 bits on a specific second (1504 bits per second is one TS packet per second). The overall specified bitrate should be achieved as closely as possible.

A multiplex within a playoutset shall not use any output PID more than once; if it does then this is an error and no transport stream shall be generated. For the purpose of this rule, an output PID is used when:

- It is specified as the “dst” attribute for the <pid> tag
- It is specified as the “pid” attribute for an <ait> tag
- It is specified as the “pid” attribute for an <dsmcc> tag
- PID 16 is always used. (Either it is used by a <nitPid> tag, or if the playoutset does not contain a <nitPid> tag then it is used by a <nit> tag, or if the playoutset does not contain <nitPid> or <nit> tags then it is used because the test harness generates its default NIT).
- PID 17 is used if the playoutset specifies a <sdtAndBatPid> tag.

A playout set with network connection set to NO must have a timeout value set to ensure that the Test Harness will terminate this playout set and start a new one with network connection set to YES.

As reportStepResult might require a network connection, the test implementer must make sure that the network connection is available at the end of the test, so the last playout set in a playout set definition shall always have network connection set to YES.

To support the OpenCaster stream generator, StreamEvent objects need the extension ".event" to be interpreted as and create a proper StreamEvent object in the DSM-CC. Without the ".event" extension the behaviour is undefined. So, for example the following is required in the playout set XML streamEvent element:

```
<streamEvent dst="eventObject.event" src="ste.xml"/>
```

7.4.4 Transport stream requirements

The test harness shall support the generation of transport streams as defined by valid playout set definitions as described in 7.4.3. The transport stream shall be valid according to the requirements of [18].

7.4.4.1 Overview (informative)

Generated transport streams are composed of transport stream packets taken from one or more files provided as part of the test suite (as defined by the transportStream element of the playout set definition), optionally combined with transport stream packets generated by the harness (as defined by the generatedData element of the playout set definition.) The packets generated by the harness shall define either an AIT table or a DSM-CC object carousel, both as defined in [4].

7.4.4.2 Static transport stream components

For each transportStream element in the playout set definition the file listed in the file attribute (the original transport stream) shall be repeatedly multiplexed into the generated transport stream.

Each transport stream packet from the original transport stream shall only be multiplexed into the generated transport stream if the PID of the packet is equal to the src attribute of one of the pid elements contained in the transportStream element. In this case, the PID in the packet shall be replaced with the value in the dst attribute of the pid element with a matching src attribute.

Within a single <transportStream> tag, a particular source PID value (<pid src="…">) shall be specified at most once; if it is used more than once in the same <transportStream> tag then this is an error and no transport stream shall be generated. There are no restrictions on using the same source PID value in different <transportStream> tags in the same playoutset.

In the final generated transport stream packets originating from the original transport stream shall occur at the same frequency at which they occurred in the original transport stream. The frequency of packets in the original transport stream shall be determined by reference to the location of the packets in the transport stream and the bitrate attribute, in bits per second, of the transportStream element.

7.4.4.3 Dynamic transport stream components

7.4.4.3.1 AIT

For each ait element contained in the generatedData element of the playout set definition the harness shall generate transport stream packets and multiplex them into the final generated transport stream such that the generated bits occur at the rate, in bits per second, given by the bitrate attribute of the ait element. If no bitrate attribute is defined, then the default value of 5000 bits per second shall be used. The generated transport stream packets shall have their PID set to the value defined by the pid attribute of the ait element.

The version field of the generated transport stream packets containing the AIT shall be set to the value of the version attribute of the ait element, or 0 if no version attribute is defined. The harness may optionally implement the behaviour described in 7.4.4.5.1.

The test harness shall read the file identified by the src attribute of the ait element. The harness shall encode the data read from the XML definition into the generated transport stream packets as defined in clause 7.2.3.1 of [1]. If the file identified by the src attribute does not contain a well-formed XML encoding of an AIT, as defined in 5.4 of [4] (including the <ParentalRating> extension defined in clause 7.2.3.2 of [1]), then the test harness shall not generate the transport stream.

If the XML definition includes a <ParentalRating> extension with 'region' attribute equal to '{auto}', then the test harness shall check if the value 'PARENTAL_REGION_DVB_SI' was listed in a <setting> sub-tag of the <adaptsTo> tag, as described in Section 6.3.4.1 adaptsTo. If so, the harness will use use the region code configured by the tester. If not, this is a test case bug, and the harness will cause the test case to fail automatically.

The harness shall generate the application_icons_descriptor from the <icon> tags in the XML AIT. Note that the Operator Applications specification [37] includes a correction to the XML schema in TS 102 809 [4] specification that allows an XML AIT to specify multiple icons, like the broadcast AIT; it also changes the definition of the version field.

The broadcast AIT may need to refer to the URL of the configured HTTPS server. To do this, inside <mhp:URLBase> and <mhp:BoundaryExtension> tags the string "{https}" shall expand to the URL of the HTTPS server without a trailing slash, for example "https://hbbtv1.test" or "https://operator.example.com:1234".

Typical usage:

```
<mhp:BoundaryExtension>{https}</mhp:BoundaryExtension>
```

Or:

```
<mhp:URLBase>{https}/_TESTSUITE/TESTS/org.hbbtv_123/</mhp:URLBase>
```

Tests that do not configure the HTTPS server in their implementation XML file, must not use the "{https}" extension described above.

7.4.4.3.2 DSM-CC

For each dsmcc element contained in the generatedData element of the playout set definition the harness shall generate transport stream packets and multiplex them into the final generated transport stream such that the generated bits occur at the rate, in bits per second, given by the bitrate attribute of the dsmcc element. If no bitrate attribute is specified then the default value of 100000 bits per second shall be used. The generated transport stream packets shall have their PID set to the value defined by the pid attribute of the dsmcc element.

The version field of the generated transport stream packets containing the DSM-CC shall be set to the value of the version attribute of the dsmcc element, or 0 if no version attribute is defined. The harness may optionally implement the behaviour described in 7.4.4.5.2.

The test harness shall generate a DSM-CC object carousel as defined in clause 7 of [4] containing a file system. The contents of the directory identified by the source_folder attribute of the dsmcc element shall be included at the root of the generated file system, i.e. Files contained directly in the indicated directory shall be at the root level of the carousel, subdirectories of the indicated directory shall be subdirectories of the carousel, etc. The directory identified by the source_folder attribute may be empty.

For each directory element contained in the dsmcc element, the contents of the directory indicated by the src attribute shall be available in the generated file system at the location indicated by the dst attribute. The location specified by dst is a path relative to the root of the generated file system.

For each file element contained in the dsmcc element, the file indicated by the src attribute shall be available in the generated file system at the location given by the dst attribute.

For each streamEvent element contained in the dsmcc element, the stream event described by the contents of the file identified by the src attribute shall be available in the generated file system at the location given by the dst attribute. If the file identified by the src attribute is not a valid XML document according to the schema defined in 8.2 of [4] then the transport stream shall not be generated.

The location specified by the dst attribute of the directory, file and streamEvent elements is a path relative to the root of the generated file system. All the parent directories of a location specified in a dst attribute must have been defined by either the directory structure identified by the source_folder attribute of the parent dsmcc element, or by the directory structure resulting from a sibling directory element.

For all paths in the generated file system referred to in file and directory elements the test harness shall treat the character '/' (ASCII 47 / Unicode U+002F) as a path separator. The test harness may also treat the character '\' (ASCII 92 / Unicode U+005C) as a path separator. The test harness shall support relative paths, and shall support the referencing of files from any location within the test suite associated with the playout set definition.

If the test harness is unable to locate any one of the indicated file system assets then the test harness shall not generate the transport stream. If the contents of the dsmcc element result in an ambiguous definition for the structure of the constructed carousel (e.g. the dst attribute of a file element refers to a path already defined by the contents of the source_folder attribute) then the transport stream shall not be generated. The test harness may choose not to generate the transport stream if src or source_folder attributes reference file system locations outside the test suite associated with the playout set definition.

The generated DSM-CC carousel shall use the following attributes of the dsmcc element as specified:

- association_tag: this value shall be used for the DSM-CC elementary stream's association tag. If the value is outside the legal range then the transport stream shall not be generated.
- carousel_id: this value shall be used for the DSM-CC carousel ID. If the value is outside the legal range then the transport stream shall not be generated.
- Version: this value shall be used for the version number of the module/DII. If not specified 0 shall be used. The harness may optionally implement the behaviour described in 7.4.4.5.1. If the value is outside the legal range then the transport stream shall not be generated.

7.4.4.3.3 NIT

By default, the Test Harness shall generate a NIT Actual and insert it into the DVB broadcast that is being played out, as described in 7.4.4.4. This default NIT Actual may not be suitable for all tests, so the test case can specify the NIT Actual in XML format, and the Test Harness will generate the specified NIT Actual. The test case may also specify NIT Other(s) in XML format, and the Test Harness will generate them in addition to the (default or specified-in-XML) NIT Actual.

Only a single NIT Actual is supported per generated TS, but an unlimited number of NIT Others are supported.

There are two different ways to specify the NIT in the playoutset XML file. If only specifying a single NIT Actual, then the simplest way is to put the <nit> XML tag in the <generatedData> block in the playoutset XML file, as the first thing in that block. Alternatively, the more flexible syntax that supports NIT Other is to have a <nitPid> XML tag in the <generatedData> block in the playoutset XML file, as the first thing in that block. The <nitPid> XML tag can contain an optional <nit> tag and any number of <nitOther> tags.

This is the simple Playoutset XML syntax:

```
<nit src="<rel_filename>" bitrate="1504">
```

This is the alternative, more powerful Playoutset XML syntax:

```
<nitPid bitrate="1504">
    <nit src="<rel_filename>">
    <nitOther src="<rel_filename>">
    <nitOther src="<rel_filename>">
</nitPid>
```

The “src” attribute specifies the path to the NIT XML file. It is relative to the directory containing the playoutset XML file.

The generated NIT PID shall repeat all the NIT(s) specified in rotation. The optional “bitrate” attribute on <nitPid> (or, if <nitPid> is not used, on <nit>) specifies the total TS bitrate for the NIT PID. This includes all the TS packet overhead. If the bitrate is not specified, then the harness shall calculate the bitrate needed to repeat all the NITs once per second.

NOTE: For a NIT that fits in one 188-byte TS packet and is repeated at the DVB specified minimum rate of every 10 seconds the required bitrate = 188 byes * 8 bits/byte / 10 seconds = 150.4 bits per second. The bitrate has to be specified as an integer, so you can round that, making sure to round up to stay inside the 10 second limit, giving 151 bits/sec.

The NIT XML defines a customized NIT to be generated by the Test Harness.The root element of NIT XML files must be the <nit> element defined in nit.xsd.

```
<nit nid="<0-65535>" version="<0-31>">
  <network>
    <networkNameDescriptor><text></networkNameDescriptor>
  </network>
  <transportStream onid="<0-65535>" tsid="<0-65535>">+
    <autoDeliverySystemDescriptor multiplex="<multiplex>"/>
    <serviceListDescriptor>
      <service sid="<0-65535>" type="<serviceType>"/>+
    </serviceListDescriptor>
    <privateDataSpecifierDescriptor value="<value>"/>
    <rawDescriptor tag="<0-255>">
        <hex_bytes >
    </rawDescriptor>
       <linkageDescriptor type="<0-7><32-255>" onid="<0-65535>" tsid="<0-65535>"
       sid="<0-65535>"/>
  </transportStream>
</nit>
```

All numbers in this file are decimal, except where specifically noted, because that is the standard XML Schema method of representing a number.

The <autoDeliverySystemDescriptor> tag inserts the correct delivery system descriptor for the current multiplex in the playoutset (i.e. the multiplex we’re inserting the NIT into), taking into account the configured DVB type and the configured DVB modulation settings. This allows the same test to work in DVB-T, C, and S.

The delivery system used, and hence the content of the delivery system descriptors is implementation dependent. The inserted delivery system descriptors shall be the same as those returned by the getPlayoutInformation test API defined in 7.2.4.

To refer to other multiplexes, there is an optional parameter on that tag. <autoDeliverySystemDescriptor multiplex="0"> inserts the correct delivery system descriptor for the first multiplex in the playoutset XML file, multiplex="1" refers to the DSD for the second multiplex, etc.

The descriptors inserted by <autoDeliverySystemDescriptor> are:

Modulation	Descriptor(s)
DVB-T	terrestrial_delivery_system_descriptor
DVB-T2	T2_delivery_system_descriptor
DVB-C	cable_delivery_system_descriptor
DVB-S	satellite_delivery_system_descriptor
DVB-S2	satellite_delivery_system_descriptor and S2_satellite_delivery_system_descriptor

<network> and <transportStream> are both descriptor loops, so they can include any of the following descriptor tags an unlimited number of times in any order. The generated NIT will contain the relevant descriptors in the order that they were specified in the XML.

- <networkNameDescriptor>. This inserts a network_name_descriptor. The content of this XML tag is the network name. If possible, this will be encoded into the descriptor using "character code table 00 - Latin alphabet" from ETSI EN 300 468 [18]. Otherwise it will be encoded into the descriptor using UTF-8, which will be correctly signalled as described in Annex A of ETSI EN 300 468 [18].
- <autoDeliverySystemDescriptor>. This causes the relevant delivery system descriptor(s) to be inserted, as described above. By default this refers to the the multiplex we're inserting the NIT into. To refer to other multiplexes, there is an optional "multiplex" attribute. multiplex="0" inserts the correct delivery system descriptor for the first multiplex in the playoutset XML file, multiplex="1" refers to the DSD for the second multiplex, etc.
- <serviceListDescriptor>. This inserts a service_list_descriptor. This XML tag contains a list of <service> tags describing services to encode into the descriptor. The services are encoded into the descriptor in the order they are specified in the XML NIT. Each <service> tag has a "sid" attribute for the service ID, and a "type" attribute for the service type. The "type" can be a decimal integer in range 0-255 inclusive, or a short human-readable code for the common types:
  - "mpeg2-sd-tv" = 0x1
  - "radio" = 0x2
  - "teletext" = 0x3
  - "avc-radio" = 0xa
  - "data" = 0xc
  - "mpeg2-hd-tv" = 0x11
  - "avc-sd-tv" = 0x16
  - "avc-hd-tv" = 0x19
- <privateDataSpecifierDescriptor>. Inserts a private_data_specifier_descriptor. The mandatory "value" parameter is the 32-bit ID of the organisation, in decimal.
- <rawDescriptor>. This allows inserting any descriptor. Specify the descriptor tag, in decimal, as the mandatory "tag" parameter. Specify the descriptor payload, if any, in hex as the contents of this tag. The descriptor length will be automatically calculated from the payload. In the XML, the payload can optionally contain whitespace between bytes, but not between nibbles in the same byte. I.e. "123456789abcdef0", "12 34 56 78 9a bc de f0", and "12 34 56 78 9abcDEF0" are all legal payloads and all mean the same thing, but
  "1 23456789abcdef0" is not legal. This allows you to document the payload by breaking it across lines and using XML comments.
- <linkageDescriptor>. This inserts a linkage_descriptor. All the numeric fields are decimal. The "type" attribute specifies the linkage_type field in the generated descriptor. Note that this tag only supports simple usage of the descriptor: it does not support types 8, 13 or 14, and it does not support including any private_data_byte. If you need to generate a linkage_descriptor with private_data_byte, use <rawDescriptor> instead.
- <serviceListDescriptor>. This inserts a service_descriptor. (This descriptor is intended for use in the SDT, however for consistency and simplicity the list of supported descriptors is the same for NIT, BAT and SDT). The "type" attribute has the same values as the "type" attribute on <serviceListDescriptor>. The "provider" attribute specifies the name of service provider and the "name" attribute specifies the service name. For each of these two strings, if possible, it will be encoded into the descriptor using "character code table 00 - Latin alphabet" from ETSI EN 300 468 [18]. Otherwise it will be encoded into the descriptor using UTF-8, which will be correctly signalled as described in Annex A of ETSI EN 300 468 [18].

- <hbbtvUriLinkageDescriptor>. This inserts a URI_linkage_descriptor. This tag is only supported when testing OpApps under Annex D. Refer to D.4.2 NIT extensions and D.4.3 BAT generation for the details of this tag.

See ETSI EN 300 468 [18] for the specification of the generated binary NIT and descriptors.

It is an error to specify any descriptor with a descriptor payload larger than 255 bytes.

Each generated NIT is limited to a single section. It is an error to specify a NIT larger than that.

The NIT will be signalled as a "NIT actual" or "NIT other" depending on which XML tag was used in the playoutset to include it. The "current_next_indicator" will always indicate that the NIT is current. "Next" NIT is not supported.

If the network ID is not set using the "nid" attribute on the NIT, then the harness shall fill it in automatically, using the Network ID configured in the harness if running OpApp tests as described in Annex D section D.2.2 Discovery settings or the standard Network ID used for the base stream as described in 5.2.3 Base Test Stream if running regular HbbTV tests.

NIT used by the OpApp test cases shall not use the version number of 6. For details, see section D.4.5 OpApp Launcher Stream.

7.4.4.3.4 BAT

The NIT XML schema has been extended to support generation of BATs and SDTs as well as NITs. The majority of the schema is shared. All the rules about descriptors etc in NITs also apply to BATs.

The root element of BAT XML files must be the <bat> element defined in nit.xsd.

If the Bouquet ID is not set using the "bouquetId" attribute on the BAT, then the harness shall fill it in automatically, using the Bouquet ID configured in the harness. It is an error if there is no Bouquet ID configured in the harness. For an OpApps test case testing under Annex D, if the bilateral agreement includes a Bouquet ID, then that must be the one configured in the harness. (see D.2.2 Discovery settings)

Here is an example BAT for a non-OpApps test:

```
<?xml version="1.0" encoding="utf-8"?>
<bat xmlns="http://www.hbbtv.org/2016/nit" version="0" bouquetId="12345">
    <bouquet>
        <bouquetNameDescriptor>HBBTV_A</bouquetNameDescriptor>
    </bouquet>
    <transportStream onid="99" tsid="1">
        <autoDeliverySystemDescriptor />
        <serviceListDescriptor>
            <service sid="10" type="mpeg2-sd-tv"/>
            <service sid="11" type="mpeg2-sd-tv"/>
            <service sid="12" type="mpeg2-sd-tv"/>
            <service sid="13" type="mpeg2-sd-tv"/>
            <service sid="14" type="radio"/>
        </serviceListDescriptor>
    </transportStream>
</bat>
```

7.4.4.3.5 SDT

The NIT XML schema has been extended to support generation of SDTs and BATs as well as NITs. The majority of the schema is shared. All the rules about descriptors etc in NITs also apply to SDTs.

The SDT XML file defines a SDT Actual to be generated by the Test Harness. The root element of SDT XML files must be the <sdt> element defined in nit.xsd.

NOTE: Instead of using the <sdt> tag, the test case author may choose to insert a SDT from a TS file using the <transportStream> and <pid> tags. The standard base stream includes a suitable SDT for this, and many test cases use that. However, the rules in section 7.4.3 mean that a playoutset may not have both a <pid> tag and generated data on the same (destination) PID. Since the BAT(s) and SDT are on the same PID, a playoutset that uses the <bat> tag cannot specify a SDT using <pid>, it must use <sdt> if the test case author wants to generate a valid DVB TS.

The following example SDT XML file will generate the same SDT Actual as the one in the base stream:

```
<?xml version="1.0" encoding="utf-8"?>
<sdt xmlns="http://www.hbbtv.org/2016/nit"
     tsid="1" onid="99" version="1">
  <service sid="10" eitSchedule="true"
           eitPresentFollowing="true"
           runningStatus="4" ca="false">
     <serviceDescriptor type="mpeg2-sd-tv"
                        provider="HbbTV.org"
                        name="ATE Test 10"/>
  </service>
  <service sid="11" eitSchedule="true"
           eitPresentFollowing="true"
           runningStatus="4" ca="false">
     <serviceDescriptor type="mpeg2-sd-tv"
                        provider="HbbTV.org"
                        name="ATE Test 11"/>
  </service>
  <service sid="12" eitSchedule="true"
           eitPresentFollowing="true"
           runningStatus="4" ca="false">
     <serviceDescriptor type="mpeg2-sd-tv"
                        provider="HbbTV.org"
                        name="ATE Test 12"/>
  </service>
  <service sid="13" eitSchedule="true"
           eitPresentFollowing="true"
           runningStatus="4" ca="false">
     <serviceDescriptor type="mpeg2-sd-tv"
                        provider="HbbTV.org"
                        name="ATE Test 13"/>
  </service>
  <service sid="14" eitSchedule="true"
           eitPresentFollowing="true"
           runningStatus="4" ca="false">
     <serviceDescriptor type="radio"
                        provider="HbbTV.org"
                        name="ATE Test 14"/>
  </service>
</sdt>
```

All numbers in this file are decimal, because that is the standard XML Schema method of representing a number, except where specifically noted. The <service> tags describe the services to encode into the SDT Actual. They will be encoded in the order they are specified in the XML SDT. Each <service> tag has:

- “sid” attribute for the service ID
- “eitSchedule” for flag to indicate that EIT schedule information for the service is present in the current TS
- “eitPresentFollowing” for indicating that EIT_present_following information for the service is present in the current TS
- “runningStatus” for indicating the status of the service.
- “ca” for indicating whether service is scrambled or not. If set to true, service is scrambled. Default is false.

If the SDT contains a <hbbtvUriLinkageDescriptor> with enabled="auto", then the SDT XML file is invalid. If the SDT contains a <hbbtvUriLinkageDescriptor> where the “uri” attribute contains “{opapp-fqdn}” or “{opapp-fqdn-2}” then the SDT XML file is invalid.

7.4.4.4 Construction of final generated transport stream

The final transport stream is generated by multiplexing together the dynamic and static transport stream components, which shall have been generated as described in 7.4.4.2 and 7.4.4.3, such that the components have the bitrates specified in their definitions.

If the first (or only) DVB multiplex doesn't specify an XML NIT Actual then it behaves as if the following XML NIT Actual was specified:

```
<?xml version="1.0" encoding="utf-8"?>
<nit xmlns="http://www.hbbtv.org/2016/nit"
     nid="99" version="0">
  <network>
    <networkNameDescriptor>HBBTV_A</networkNameDescriptor>
  </network>
  <transportStream onid="99" tsid="1">
    <autoDeliverySystemDescriptor />
    <serviceListDescriptor>
      <service sid="10" type="mpeg2-sd-tv"/>
      <service sid="11" type="mpeg2-sd-tv"/>
      <service sid="12" type="mpeg2-sd-tv"/>
      <service sid="13" type="mpeg2-sd-tv"/>
      <service sid="14" type="radio"/>
    </serviceListDescriptor>
    <privateDataSpecifierDescriptor value="40"/>
  </transportStream>
</nit>
```

If there is a second DVB multiplex and it doesn't specify an XML NIT Actual then it behaves as if the following XML NIT Actual was specified:

```
<?xml version="1.0" encoding="utf-8"?>
<nit xmlns="http://www.hbbtv.org/2016/nit"
     nid="65281" version="0">
  <network>
    <networkNameDescriptor>HBBTV_B</networkNameDescriptor>
  </network>
  <transportStream onid="99" tsid="2">
    <autoDeliverySystemDescriptor />
    <serviceListDescriptor>
      <service sid="15" type="mpeg2-sd-tv"/>
      <service sid="16" type="mpeg2-sd-tv"/>
      <service sid="17" type="mpeg2-sd-tv"/>
      <service sid="18" type="mpeg2-sd-tv"/>
      <service sid="19" type="radio"/>
    </serviceListDescriptor>
    <privateDataSpecifierDescriptor value="40"/>
  </transportStream>
</nit>
```

The generated transport stream should have a data rate matching that required by the parameters of the inserted delivery descriptors.

7.4.4.5 Recommended test harness behaviour

In order to support the widest range of DUTs and their different application caching models the test harness may choose to implement these behaviours. Not implementing these features may result in the test harness not being able to pass some valid implementations of the DUT without an alternative test strategy.

EXAMPLE: Some DUTs may implement an application caching model which relies on the AIT version numbering scheme. If this is not used then the DUT may need to be power cycled or service changed between tests.

7.4.4.5.1 AIT version offset

The version field of the generated transport stream packets containing the AIT should be set so as to be equivalent to the output of the following algorithm:

- test_run is a positive integer incremented by 1 every time the harness starts a test

- base_version is the value of the version attribute of the ait element, or 0 if no version attribute is defined
- version field in transport stream packets = (test_run modulo 4) × 8 + base_version

7.4.4.5.2 DSM-CC version offset

The module/DII version and the version field of the generated transport stream packets containing the DSM-CC should be set so as to be equivalent to the output of the following algorithm:

- test_run is a positive integer incremented by 1 every time the harness starts a test
- base_version is the value of the version attribute of the dsmcc element, or 0 if no version attribute is defined
- version field value = (test_run modulo 2) × 8 + base_version

7.4.4.5.3 PCR regeneration

The test harness may rewrite the PCR fields of the generated transport stream.

When the playout set includes a <pcr pid="x"> tag (where "x" is a reference to a PID in the Transport Stream). This indicates the test harness must ensure that the PCR on the specified PID increments at a constant rate relative to the stream playout and that the starting PCR value remained unchanged.

7.4.4.6 TOT/TDT synchronization

When the playout set includes a <synchronizeTotTdt/> tag this indicates that the test harness must ensure that the TOT and TDT tables in the Transport Stream are encoded so that the current UTC time is present. The time offset in the TOT shall not be changed. This must be maintained if the Transport Stream is looped during playout.

7.4.4.7 EIT time updating

When the playout set includes a <adjustEit/> tag this indicates that the test harness must update the event start times in the EIT sections in the Transport Stream.

In summary the goal is:

- Given a stream which was recorded at a specific time, the “baseStreamStartTime”, if we’re playing the stream out at a later date then the test harness has to adjust the EIT appropriately so that events that were “one hour after baseStreamStartTime” become “one hour after the test started”.
- It’s usually better if the event start times are a multiple of 1min (or even 5min), so there is a configurable “granularity” which controls the granularity of the adjustment. This defaults to 1min.
- This mechanism also allows the EIT section version numbers to be adjusted, so the RUT detects the change
- The test harness ensures that the generated EIT-schedule data complies with the DVB SI Guidelines. To do this, the EIT-schedule has to be pulled apart and regrouped into sections.

Test authors shall ensure that each playout set using <adjustEit/> uses the same section version number on all EIT-schedule sections relating to a single service. (I.e. a change in EIT-schedule data can only happen when the playout set is changed; but a change in EIT-pf can happen at any point in the stream).

The test harness shall update the EIT using the following process:

1) First calculate the ‘EIT Offset’, which is a positive number of seconds, by:
    a) Calculate the number of seconds between the date/time given by the baseStreamStartTime attribute on the <adjustEit/> tag, and the actual time the harness starts running the test. (For tests with more than one playout set, note that this is the time the test is started, not the time the playout set is started).

b) Round that down to a multiple of the value given by the granularity attribute on the <adjustEit/> tag.

2) Prepare the updated EIT-schedule sections as follows:

a) Parse all EIT-schedule sections in the generated TS, and extract the events and section version number for each service. Also remember the maximum number of sections seen in a single TS packet on the EIT PID.

b) For every event extracted in step 1, adjust the start_time value in every event loop inside the EIT by adding 'EIT Offset' seconds to the encoded date/time. If the start_time does not encode a valid date/time (e.g. if all bits of the field are set to "1", as explicitly allowed by [18]) then it is not adjusted

c) For each service, increment the version_number by the value given by the incrementVersion attribute on the <adjustEit/> tag. The addition shall be done modulo 32.

d) For each service, generate EIT-schedule sections containing the events, following the rules in ETSI TS 101 211 to sort events and to assign events to sections. Note that calculation of "last midnight" must be based on the time the test is started

3) Update the TS using the following rules:

a) TS packets that are not on the EIT PID are not modified by this algorithm

b) EIT-schedule sections are replaced with the updated EIT-schedule sections from step 2

c) EIT-pf sections are adjusted by:

a. Adjust the start_time value in every event loop inside the EIT by adding 'EIT Offset' seconds to the encoded date/time. If the start_time does not encode a valid date/time (e.g. if all bits of the field are set to "1", as explicitly allowed by [18]) then it is not adjusted

b. Increment the version_number by the value given by the incrementVersion attribute on the <adjustEit/> tag. The addition shall be done modulo 32.

c. Adjust the section CRCs accordingly. If the CRC was correct before the changes above, then the CRC shall be correct on the modified section. If the CRC did not match before the changes above, then the CRC shall not match on the modified section.

d) Other sections on the EIT PID are passed through unchanged

e) The repacking of sections into the EIT PID shall respect the maximum number of sections in a single TS packet as measured in step 2a.

f) Note that due to the repacking of sections on the EIT PID, some EIT-pf sections and non-EIT sections will slightly change position in the stream. This is expected to be negligible unless you are using extremely low bitrates for the EIT PID. (In the worst-case, the delay is approximately the size of 2 EIT sections).

7.4.5 JS-Function changePlayoutSet()

This call changes the current playout (and therefore may disable the network connection).

```
void HbbTVTestAPI.changePlayoutSet(
        playoutSetId : integer,
        callback : function or null,
        callbackObject : object);
```

ARGS: **playoutSetId:** the id of the playout set to start playing out. If the playout set with this id is not defined for the test then the test shall fail.

callback/callbackObject: a callback function to invoke after the new playout has started (also see chapter 7.2.3).

As audio/video might show some artefacts, test case implementers should not perform any broadcast audio/video checks up to 5 seconds after a switch of playout set occurs.

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: To switch from no network to an available network connection, use the timeout parameter in the playout set.

7.4.6 JS-Function setNetworkBandwidth()

This function restricts the maximum incoming application data permitted on the network interface of the device under test.

```
void HbbTVTestAPI.setNetworkBandwidth(
        bitrate : int,
        callback : function or null,
        callbackObject : object);
```

ARGS: **bitrate:** the maximum “nominal bitrate” (see below) into the device under test, in units of bits per second (bps). Values less than 1bps are not supported, and shall cause the test to fail.

callback/callbackObject: a callback function to invoke when the restriction has been applied (also see chapter 6.2.3 Callbacks).

The maximum throughput is calculated by multiplying the specified maximum nominal bitrate by 1.4.

Throughput is defined as the average number of bits per second passed in to the transmitting network interface at the Ethernet frame layer. This count includes all HTTP, TCP/UDP and IP overheads, and includes Ethernet destination MAC address, source MAC address and length fields. But it does not include the other Ethernet overheads such as the preamble, Start Frame Delimiter, the padding to the minimum payload size if necessary, the frame check sequence (CRC) field and the interpacket gap.

Note: For example, the calculated maximum throughput may be used directly as the "rate" parameter to the Linux TC TBF qdisc. In this case, the "overhead" and "mpu" parameters should not be used or should be zero.

Note: This value of 1.4 was chosen in an attempt to account for HTTP, TCP/UDP, IP etc overheads. It is not a perfect conversion factor, but cannot be changed for historical reasons - changing it would require revalidating all the test cases that use this API.The restriction only applies to data received by the device under test (i.e. data sent by the test harness.) Transmission of data from the device under test shall not be restricted.

Calls to this function set an upper limit on the permitted throughput. The maximum throughput achievable from the test harness may be limited to a lower value by other factors (e.g. bus or CPU saturation.) Test authors should take this into account when deciding appropriate values for bitrate.

7.4.7 CICAM related JS functions

The behaviour of the functions defined below is undefined if the CAM element is not present in the implementation XML. In this case harnesses may cause tests to fail.

The table below shows which functions may be used for each configurable state of the CICAM (these functions are marked ●). Behaviour of functions if they are called other than as permitted by Table 5 is undefined, and the harness may abort the test.

JS-Function	Value of 'type' attribute of 'CAM' element		
	cspgcip	cip	ci
clearAuthentication	●	●	
setScramblingEnabled	●	●	
getScramblingState	●	●	
sendURI	●	●	
sendReply	●		
sendParentalControlInfo	●		
sendRightsInfo	●		
sendSystemInfo	●		
waitForMessage	●		

Table 4: Functions permitted for different CICAM configuration states

7.4.7.1 Error Handling

If an error occurs during the interaction between the harness and the CICAM then the harness shall halt execution of the test. The harness may display an error message to the operator, and may add a meaningful human readable error to the test report. Situations that will trigger this error handling shall include, but are not limited to:

- Invalid parameters in function calls
- Use of functions that communicate with the CICAM when the CICAM is not inserted in the host
- Use of functions that communicate with the CICAM when the 'CAM' XML element (see 5.2.1.7.5) is not included in the implementation XML.

7.4.7.2 Serialisation of 'cicam' methods

The functions described in the following sections all return immediately when called. The functionality is then executed asynchronously and the specified callback (if provided) is called on completion (see 7.2.3). The CICAM shall execute API call actions sequentially. As such, if the action of one function is still being carried out when another function is called, the action from the second function shall not be carried out until the first function has completed its interaction with the CICAM. If an API call causing a CICAM action is called while the CICAM is executing a previous action then the harness may cause the JavaScript callback associated with the running action to be executed either before or after execution of the second action is complete.

Otherwise functions shall be processed as normal.

7.4.7.3 Data types

ECMAScript (and JavaScript profiles) only define basic 'Number' and 'String' types. Several of the functions defined in this section require more restricted input. Rather than define new types the functions shall accept JavaScript primitives and may perform validation to assert that they are valid. (Test developers shall not use invalid values.) The types used in this document, the corresponding primitive type to be used in an implementation, and the valid ranges are shown in the table below.

Type in this document	JavaScript primitive type	Valid values
uint8	number	Integers in range 0 ≤ value ≤ 255
uint16	number	Integers in range 0 ≤ value ≤ 65535
hexBinary	string	See below

Table 5: Interpreting CICAM parameter types

If values are not valid then error handling behaviour shall be as defined in 7.4.7.1.

7.4.7.3.1 'hexBinary' type

The hexBinary type is intended to encode arbitrary binary data, with semantics as defined in 4.2.3.4.1.1.2 of [29] (i.e. as defined for xs:hexBinary type used in XML schemas).

7.4.7.4 JS functions

The APIs are all defined as methods on an object ‘cicam’ which is a property of the object returned by the HbbTVTestAPI constructor.

If a harness does not support use of a CICAM then the ‘cicam’ property may have the value ‘undefined’. If the ‘cicam’ property does not have the value ‘undefined’ then this shall not be taken to mean that the harness supports the use of CICAM or that a CICAM is available. Tests shall request the use of a CICAM by the use of the XML syntax defined in 5.2.1.7.5.

The ‘cicam’ prefix is shown in the title of the following sections, but not in the function definitions.

7.4.7.4.1 JS-Function cicam.clearAuthentication

This function will clear all cached authentication context data, so at next insertion the CICAM shall carry out the full host authentication protocol. It may also clear other cached CICAM state, depending upon the harness implementation.

```
void HbbTVTestAPI.cicam.clearAuthentication(
        callback : function or null,
        callbackObject : object);
```

ARGS: **callback / callbackObject**: callback function invoked if the function call succeeds

In addition to the above behaviour, calling this function is equivalent to calling the following methods:

- cspgcipSetScramblingEnabled(true)
- cspgicpSendURI(copy freely)

7.4.7.4.2 JS-Function cicam.setScramblingEnabled

This function allows the CICAM’s descrambling feature to be disabled.

```
void HbbTVTestAPI.cicam.setScramblingEnabled(
        enabled : boolean,
        callback : function or null,
        callbackObject : object);
```

ARGS: **enabled**: If ‘true’ then transport stream descrambling will be enabled, subject to the other requirements of the CI+ specification (e.g. even if this has been set to true, if host authentication fails then descrambling shall not take place). If ‘false’ then transport stream descrambling shall cease if it is currently being carried out, and not restart.

callback / callbackObject: callback function invoked if the function call succeeds

The state set by this function shall not be preserved across CICAM removal / re-insertion and CICAM reboots.

7.4.7.4.3 JS-Function cicam.getScramblingState

This function allows the retrieval of information about the current state of the CICAM.

```
void HbbTVTestAPI.cicam.getScramblingState(
        callback : function,
        callbackObject : object);
```

ARGS: **callback / callbackObject:** a callback function to invoke when the information is available.

The callback function will be called with the following parameters:

```
void callback(
        callbackObject,
        cspgcipInformationObject)
```

where the cspgcipInformationObject is an associative array containing the following key/value pairs, where each value is a boolean:

tsDetected: true if a transport stream is being input to the CICAM, otherwise false

descrambling: true if the CICAM is currently descrambling a transport stream

The value returned for ‘descrambling’ indicates that the CICAM has authenticated the host, has a transport stream input and has retrieved sufficient CA system information to configure and start its descrambler. It does not guarantee that descrambling is taking place correctly or that the output video is correctly descrambled: if this information is required then it must be confirmed with an analyzeScreenExtended or analyzeVideoExtended, as appropriate.

7.4.7.4.4 JS-Function cicam.sendReply

This function shall cause the CICAM to send a SAS_async_msg APDU to the host with command_id 0x02 (reply_msg [29]) and with contents structured as 4.2.3.4.1.1.4 of [29].

```
void HbbTVTestAPI.cicam.sendReply(
        oipf_status : uint8,
        oipf_ca_vendor_specific_information : hexBinary,
        ca_system_id : uint16,
        callback : function or null,
        callbackObject : object);
```

ARGS: **oipf_status**: value to be used for the oipf_status of the reply_msg as defined in 4.2.3.4.1.1.4 of [29]

oipf_ca_vendor_specific_information: value for the oipf_ca_vendor_specific_information field of the reply_msg. May be null, in which case no oipf_ca_vendor_specific_information shall be sent to the host.

ca_system_id: value to be used for the ca_system_id of the SAS_async_msg APDU encoding the reply_msg.

callback / callbackObject: callback function invoked if the function call succeeds

7.4.7.4.5 JS-Function cicam.sendParentalControlInfo

This function shall cause the CICAM to send a SAS_async_msg APDU to the host with command_id 0x03 (parental_control_info — 4.2.3.4.1.1.1 [29]) and with contents structured as 4.2.3.4.1.1.5 of [29].

```
void HbbTVTestAPI.cicam.sendParentalControlInfo(
        oipf_access_status : uint8,
        oipf_rating_type : uint8,
        oipf_rating_value : uint8,
        oipf_country_code : array,
        oipf_control_url : string,
        ca_system_id : uint16,
        callback : function or null,
        callbackObject : object);
```

ARGS: **oipf_access_status**: 0 or 1 (see 4.2.3.4.1.1.5 of [29])

oipf_rating_type: as defined in 4.2.3.4.1.1.5 of [29]

oipf_rating_value: as defined in 4.2.3.4.1.1.5 of [29]

oipf_country_code: JavaScript Array containing 0 or more uint16 values as specified in 4.2.3.4.1.1.5 of [29]. If the array length is 0 or the value of the parameter is null then no oipf_country_code shall be sent.

oipf_control_url: either null, or a string (see 4.2.3.4.1.1.5 of [29]). If null then no oipf_control_url shall be sent. The contents of oipf_control_url shall be encoded using UTF-8 by the harness for inclusion in the SAS_async_msg APDU.

ca_system_id: value to be used for the ca_system_id of the SAS_async_msg APDU encoding the reply_msg.

callback / callbackObject: callback function invoked if the function call succeeds

7.4.7.4.6 JS-Function cicam.sendRightsInfo

This function shall cause the CICAM to send a SAS_async_msg APDU to the host with command_id 0x04 (rights_info — 4.2.3.4.1.1.1 [29]) and with contents structured as 4.2.3.4.1.1.6 of [29].

```
void HbbTVTestAPI.cicam.sendRightsInfo(
        oipf_access_status : uint8,
        oipf_rights_issuer_url : string,
        ca_system_id : uint16,
        callback : function or null,
        callbackObject : object);
```

ARGS: **oipf_access_status**: 0 or 1 (see 4.2.3.4.1.1.6 of [29])

oipf_rights_issuer_url: either null or a string as defined in 4.2.3.4.1.1.6 of [29]. If null then no oipf_rights_issuer_url shall be sent. The contents of oipf_rights_issuer_url shall be encoded using UTF-8 by the harness for inclusion in the SAS_async_msg APDU.

ca_system_id: value to be used for the ca_system_id of the SAS_async_msg APDU encoding the reply_msg.

callback / callbackObject: callback function invoked if the function call succeeds

7.4.7.4.7 JS-Function cicam.sendSystemInfo

This function shall cause the CICAM to send a SAS_async_msg APDU to the host with command_id 0x05 (system_info — 4.2.3.4.1.1.1 [29]) and with contents structured as 4.2.3.4.1.1.7 of [29].

```
void HbbTVTestAPI.cicam.sendSystemInfo(
        oipf_ca_vendor_specific_information : hexBinary,
        ca_system_id : uint16,
        callback : function or null,
        callbackObject : object);
```

ARGS: **oipf_ca_vendor_specific_information**: data to be sent for oipf_ca_vendor_specific_information data of system_info, as defined in 4.2.3.4.1.1.7 of [29].

ca_system_id: value to be used for the ca_system_id of the SAS_async_msg APDU encoding the reply_msg.

callback / callbackObject: callback function invoked if the function call succeeds

7.4.7.4.8 JS-Function cicam.waitForMessage

This function instructs the CICAM to wait for a SAS_async_msg APDU to be received from the terminal, and optionally respond with a SAS_async_msg encoding a reply_msg as 4.2.3.4.1.1.3 of [29].

```
void HbbTVTestAPI.cicam.waitForMessage(
        timeout : integer,
        oipf_status : uint8,
        oipf_ca_vendor_specific_information : hexBinary,
        ca_system_id : uint16,
        callback : function or null);
```

ARGS: **timeout**: a time (in milliseconds) after which the CICAM should stop waiting. Values of timeout must be in the range 1-300000. Values outside this range shall cause the test to fail.

oipf_status: If null then no reply_msg shall be sent. Otherwise a value to be used for the oipf_status of the reply_msg as defined in 4.2.3.4.1.1.4 of [29]

oipf_ca_vendor_specific_information: If oipf_status is not null then the value for the oipf_ca_vendor_specific_information field of the reply_msg. May be null, in which case no oipf_ca_vendor_specific_information field will be included in the reply_msg.

ca_system_id: If oipf_status is not null then the value to be used for the ca_system_id of the SAS_async_msg APDU encoding the reply_msg. May only be null if oipf_status is null.

callback/callbackObject: a callback function to invoke when the information is available.

The callback function will be called with the following parameters:

```
void callback(
        waitState : integer,
        receivedInformationObject : object or null,
        callbackObject : object);
```

The callback shall be called when:

- the CICAM has begun to wait for a message to be received. In this case the waitState parameter will have the value 1, and the receivedInformationObject parameter shall be null.
- a message has been received by the CICAM from the host. In this case the waitState parameter will have the value 2, and receivedInformationObject will be an associative array containing the following key/value pairs;
  - ca_system_id will give the value of ca_system_id in the received SAS_async_msg, as a JavaScript number.
  - oipf_ca_vendor_specific_information: the value of oipf_ca_vendor_specific_information from the received send_message, passed as a hexBinary string (see 7.4.7.3.1).
- the timeout expires. In this case the waitState parameter will have the value 3, and receivedInformationObject shall be null.

The callback shall be called a maximum of twice in response to any single call to waitForMessage, with each value of waitState passed at most once. The first call to the callback shall always be with a waitState value of 1.

NOTE: This function only waits for a single SAS_async_msg and only sends a single reply_msg.

7.4.7.4.9 JS-Function cicam.sendURI

This function shall cause the CICAM to initiate the Usage Rules Information (URI) refresh protocol (as specified in 5.7.5.1 of [14]) using the supplied URI.

```
void HbbTVTestAPI.cicam.sendURI(
        uri : bytestring,
        callback : function or null,
        callbackObject : object);
```

ARGS: **uri**: a URI as defined in 5.7.5.2 of [14]. The URI specified shall be in v2 format.

callback / callbackObject: callback function invoked if the function call succeeds

7.4.8 JS-Function setSignalLevel()

This call changes the signal level of a multiplex.

```
void HbbTVTestAPI.setSignalLevel(
        data : float[],
        callback : function or null,
        callbackObject : object);
```

ARGS: **data:** Data about the multiplexes for which to set the signal level. Must be an array, with a length that matches the length of the array retrieved by getPlayoutInformation(). Each entry in the "data" array corresponds to the multiplex in the same position in the array retrieved by getPlayoutInformation(). If a multiplex is not a DVB multiplex, then the corresponding entry in "data" must be null. Otherwise the corresponding entry in "data" must be a number (fractions allowed) and is the signal level in dBm. See the getPlayoutInformation() API for the valid range

and for the caveats on how this is set. The harness shall use the closest supported signal level to the specified value (i.e. values above the max are set to the max, values below the minimum are set to the minimum, and values are rounded to the closest supported step).

callback/callbackObject: a callback function to invoke after the new playout has started (also see chapter 7.2.3). The callback function will be called with the same parameters as for getPlayoutInformation().

7.5 Additional Notes

7.5.1 Test implementation guidelines

When implementing a test, the test case author should stick to the following guidelines as closely as possible. Failure to adhere to these guidelines may cause the Test Case to be rejected during the implementation review.

7.5.1.1 JS API implementation

Keep in mind that the testsuite.js file needs to be replaced by the test harness provider. Editing or replacing this file is not possible. Packaging this file into a pre-packaged DSM-CC (delivered as transport stream) is not possible. In case a pre-packaged DSM-CC needs to be created, the testsuite.js file needs to be referenced via a HTTP URL, e.g. http://hbbtv1.test/_TESTSUITE/RES/testsuite.js

In this case, the application shall contain a simple_application_boundary_descriptor giving access to the web server (in above case to http://hbbtv1.test/).

7.5.1.2 PASS requires endTest()

A test will only pass if endTest() is called, otherwise the test may time out (Test Harness dependent) or will be unresolved or failed. This makes the implementation of test cases that should pass if an application was killed a bit complicated, but not impossible, as the following solutions exist:

- Have an AIT with an AUTOSTART app that stores a cookie which counts the times that the application was started - doing this will make it possible to verify that the application was started twice, and therefore it must have been killed in between. This solution will only work if cookies are supported (only for broadband connection).
- Have an AIT with an AUTOSTART app that, as soon as it is suitable for the test to give the correct result, changes AIT on the current service (by changing the playout set). The new AIT then carries the following applications:
  - the currently running app as PRESENT, and
  - an AUTOSTART app which will be the app that is fired up once the currently running app has been killed to verify that the previous app got killed.
- By using multiple services (this avoids changing the playout set): service A contains the application 1 as AUTOSTART, which is responsible for performing the test. The application tunes to service B, which contains application 1 as PRESENT and application 2 as AUTOSTART, which will do the endTest() call. The application 1 will keep on running until it is killed, which then will start application 2.

7.5.1.3 Write tests for automation

Avoid using the JS API functions analyzeScreenExtended, analyzeAudioExtended, and analyzeVideoExtended. Whenever possible, design your test to use the functions analyzeScreenPixel and analyzeAudioFrequency instead, which support automation. Ideally Test Cases should not use analyze calls at all, as there are often other means of assessing the pass criteria.

7.5.1.4 Delayed analysis

All JS API analyze functions might be implemented in different ways:

- automated online analysis
- manual online analysis
- manual/automated offline analysis

For manual analysis, always make sure that the changes to the screen/audio/video only happen after the callback is received, which tells you that the analysis is done.

Calls to the analyze functions of the JS API shall be made only when network connection is enabled to allow the Testing API to trigger the capturing of the screen or audio signal. The test might fail otherwise

As analysis may happen offline, the analysis result is not known to the test implementation and will not be reported back in the callback. If analysis fails, the complete test will fail. So if you perform an analyze call (e.g. check whether a specific pixel has a red colour) that fails (pixel is not red), this will make sure that the complete test case fails.

7.5.1.5 Restoring network connection / Playout Set timeout

After changing to a playout set that has the network connection disabled, you should use the playout set's timeout feature to restore the network connection after a specified amount of time by switching to the next playout set after the timeout (which then should have the network connection enabled).

In most other cases, a playout set should not have a timeout in order to allow slow implementations to still pass the test.

Except for very special test cases (with a good reason not to do so), all playout sets shall signal and include all services ATE test 10 – ATE test 14 (see chapter7.2.2).

7.5.1.6 Always include basic SI tables in Playout Set definition

A playout set definition requires the referencing of a PAT, SDT, TDT/TOT. Whenever possible, use the SI table definitions from the default TS file RES/BROADCAST/TS/generic-HbbTV-Teststream.ts.

The playout set definition should NOT include a NIT, which is inserted by the test harness for the tested delivery type (e.g. DVB-S, DVB-C, or DVB-T).

In addition to that, the playout set shall contain the PMTs and referenced elementary streams for the various services that are used by your test.

7.5.1.7 Choose the correct application and organization ID for your application

Test applications should stick to the following ID guidelines:

- The organization ID should be the ID assigned to the HbbTV consortium. This is: 0x70.
- The application ID should be computed using the following algorithm:

1) Start by taking the local part of the test ID.

   g) If it cannot be interpreted as a hex number, take the SHA-1 hex digest of the local part

   h) Interpret the resulting string as a hex number.

2) When a test case requires more than one application ID, add multiples of 0x100 (256 decimal) for each additional application ID required.

3) While the resulting ID is greater than 0x3fff (16383 decimal), subtract 0x3fff from the ID.

4) If application is intended to be in signed range (for trusted API calls), add the offset 0x4000 to the resulting application ID.

7.5.1.8 Only register the key events you need

Only change the application's keyset object if actual interaction with the remote control is required. When changing the keyset, ensure that you only register the keys that are required for the test. Following these guidelines makes it easier for the test harness to return to the pre-defined state while the test is running.

7.5.1.9 Implementing portable OIPF / HbbTV applications

Where a test case is valid for both HbbTV and OIPF (as identified in the <appliesTo> element), care should be taken to ensure that implementations are compatible with both specification requirements. This means that implementations:

1) Must use the OIPF DOCTYPE. OIPF DAE 1.2 only permits the 'strict' or 'transitional' XHTML doctypes, as defined in CEA-2014-A (Annex G). OIPF DAE 2.3 permits HTML5 doctypes as defined in 8.1.1 of HTML5 (W3C Candidate Recommendation 6 August 2013).

2) Must EITHER have an initial page called index.html or index.cehtml in the test directory, OR must use the OIPF XML extensions in implementation.xml to name the initial page

3) Must not use any HbbTV extensions

7.5.1.10 reportStepResult stepID

The initial reportStepResult upon initialisation of an application should be used to indicate whether that application has started correctly. This should be 'true' except in the case of the application failing to correctly initialise, such as that which may cause an exception to be raised. The initial reportStepResult with stepID 0 should not be used to indicate an expected test failure, such as that caused be the current application being opened in error.

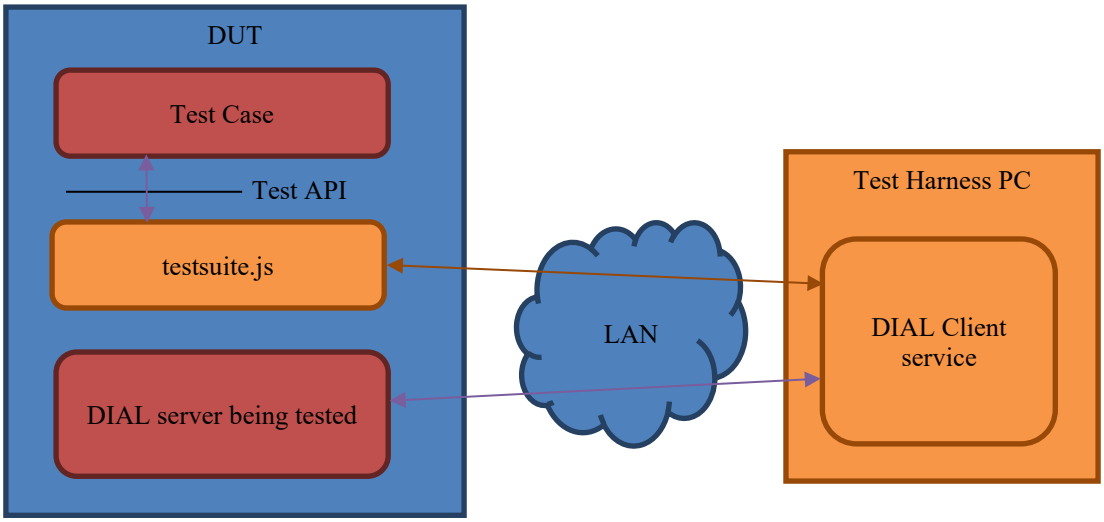
7.5.2 Things to keep in mind

When writing tests, the test implementers should keep the following information in mind:

- Analyze calls may be performed in offline mode. You don't know the analyze result when the callback is performed, you only know that you may continue with your test
- Wait for the callback when using an analyze call before changing the output (e.g. when calling analyzeScreenPixel wait until the callback has returned before changing the screen) to make sure that the analysis can be performed on the correct output.
- Make sure network connection is available when calling any API call except endTest(), reportStepResult(), sendMessage() and waitForCommunicationCompleted().
- If an analyze call fails, this means that the test fails. The test environment then may terminate the test while it continues to run, or it may wait until the test terminates itself
- HbbTV does not require monitoring of the AIT while broadband video is played back. A good test should not change the AIT during playback (only when testing very special cases).
- Before calling endTest(), the test should try to unregister all listeners and stop all broadband video playback. This makes it easier to run the following test.

7.6 APIs for testing DIAL

The Test Harness shall implement a partial DIAL Client, supporting DIAL [30] as profiled, clarified and extended by HbbTV [1]. The APIs in this section allow that DIAL Client to be controlled.



The diagram shows how a Test Case running on the DUT can make calls to testsuite.js to ask for DIAL operations to perform. The Test Harness uses some proprietary protocol (likely an XmlHttpRequest) to pass those requests from testsuite.js to the DIAL Client service that is part of the Test Harness running on the Test Harness PC. The DIAL Client service then actually does the requested DIAL operations (such as DIAL searches and sending DIAL requests). The Test Harness then communicates the result to the Test Case.

The supported DIAL operations include:

- Doing a search for DIAL servers on the network
- Parsing a URL reported by that search to get the URL
- Resolving hostnames in URLs reported by that search
- Getting the UPnP Device Description file from a DIAL server, which gets the URL for DIAL applications
- Getting the HbbTV Application Description from a DIAL server
- Starting an HbbTV Application via DIAL
- Checking that starting an HbbTV Application via DIAL is allowed from a different Origin

These APIs are accessed via the ‘dial’ property on the HbbTVTestAPI object created by the test. The ‘dial’ property is an object with (at least) the methods specified here.

7.6.1 JS-Function dial.doMSearch()

This function causes the test harness to send a SSDP M-SEARCH request to the DUT as defined in section 5.1 of DIAL [30], and get a response.

This function returns immediately. On success, it calls the provided callback function. On failure, the test harness shall cause the complete test to fail automatically and shall not call the callback. Failure can be caused by:

- no SSDP responses,
- multiple different SSDP responses when the harness has not been configured to use a specific one (see next two paragraphs), or
- no LOCATION header or otherwise malformed responses.

As defined in DIAL, the test harness may transmit the SSDP M-SEARCH request multiple times. This may result in multiple SSDP responses for the same device (i.e. with the same LOCATION header). In this case, the test harness shall accept such duplicate SSDP responses.

The test harness shall accept SSDP M-SEARCH responses conformant to either UPnP Device Architecture 1.0 [35] or UPnP Device Architecture 1.1 [36].

Optionally, a test harness may support filtering SSDP responses, so only the correct one is returned to the test. E.g. this may be useful if the Companion Screen happens to implement a DIAL server, and the Companion Screen is left on the network when tests using this API are run. If supported, the mechanism for configuring the specific SSDP response to use is harness-dependent and out of scope for this specification. Test harnesses are not required to implement filtering SSDP responses, because there are no tests that both use this API and require a real Companion Screen, so the tester could always turn the Companion Screen off while running tests that use this API.

```
void HbbTVTestAPI.dial.doMSearch(
        callback : function,
        callbackObject : object);
```

ARGS: **callback / callbackObject**: callback function invoked when the harness has a SSDP response as:

```
void callback(
        callbackObject : object,
        locationHeader : string);
```

where:

locationHeader: value of the LOCATION header of the M-SEARCH response (i.e. the URL).

7.6.2 JS-Function dial.resolveIpV4Address ()

This function tries to resolve anything that is a valid hostname into an IPv4 address.

This function returns immediately. When resolving the address is finished, whether successful or not, it calls the provided callback function.

This function shall handle all the hostname formats that are allowed in URIs (see section 3.2.2 of RFC 3986 [22]). This means dotted decimal IPv4 addresses, IPv6 addresses enclosed in square brackets, and DNS names.

```
void HbbTVTestAPI.dial.resolveIpV4Address(
        hostname : string,
        callback : function,
        callbackObject : object);
```

ARGS: **hostname**: string defining the hostname.

callback / callbackObject: callback function invoked when the harness has finished resolving the IP address as:

```
void callback(
        callbackObject : object,
        dottedIpv4 : string or null);
```

where,

dottedIpv4: string containing a dotted IPv4 address in the usual format (e.g. “123.245.67.189”) if the specified hostname could be resolved, or null if the hostname was not resolvable. This address shall not have leading zeros on any component (i.e. it can be “1.2.0.3” but not “001.002.000.003”).

Some examples that are not resolvable and shall call the callback with null include: IPv6 addresses enclosed in square brackets, DNS names that do not exist (i.e. NXDOMAIN response), DNS names that cannot be resolved due to DNS timeouts, and any string that cannot be interpreted as a valid hostname according to RFC 3986 [22].

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.6.3 JS-Function dial.getDeviceDescription()

This function does a HTTP GET for the UPnP device description file at the specified URL. This request shall conform to the DIAL specification [30] (see Section 5.3) as profiled in HBBTV [1] (section 14.7.2), except the harness shall follow up to 10 HTTP redirects. If the request fails (i.e. after following redirects if needed, the last request has anything other than a HTTP 200 response) then the harness shall fail the test automatically and shall not call the callback. If no response is received by the Test Harness within 10 seconds of making the HTTP GET request, the Test Harness shall fail the test automatically and shall not call the callback.

This function shall not do any Same-Origin or CORS [28] security checks. However, the parameters and results of this function are sufficient for a test case to test the DUT's Cross-Origin support (see HBBTV [1], Section 14.8).

```
void HbbTVTestAPI.dial.getDeviceDescription(
        url : string,
        origin : string or null,
        callback: function,
        callbackObject : object);
```

ARGS: **url**: string defining the DIAL LOCATION URL. Typically, this URL will have been obtained from an earlier call to the dial.doMSearch() function.

origin: either null, in which case the request shall not include an Origin header, or else a string, in which case the request shall include an Origin header with the value specified in that string.

callback / callbackObject: callback function invoked when the harness has successfully completed the HTTP request as:

```
void callback(
        callbackObject : object,
        applicationUrl : string or null,
        didRedirect : bool,
        allowOrigin : string or null);
```

where,

applicationUrl: string containing the value of the Application¬-URL header returned with the UPNP device description file, or null if the DUT did not return that header.

didRedirect: true if the DUT did a HTTP redirect (in violation of the DIAL specification [30]), or false otherwise.

allowOrigin: string containing the value of the Access-Control-Allow-Origin header returned with the UPNP device description file, or null if the DUT did not return that header.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.6.4 JS-Function dial.getHbbtvAppDescription()

This function does a HTTP GET for the DIAL HbbTV Application description at the specified URL. This request shall conform to the DIAL specification [30] (see Section 6.1.1). If the request fails then the harness shall fail the test automatically and shall not call the callback. HTTP redirects are prohibited and therefore if they are required this shall cause the request to fail. If no response is received by the Test Harness within 10 seconds of making the HTTP GET request, the Test Harness shall fail the test automatically and shall not call the callback.

This function shall not do any Same-Origin or CORS [28] security checks. However, the parameters and results of this function are sufficient for a test case to test the DUT's Cross-Origin support (see HBBTV 1.4.1 [1], Section 14.8).

```
void HbbTVTestAPI.dial.getHbbtvAppDescription(
```

```
        applicationUrl : string,
        schemaValidation : bool,
        origin : string or null,
        callback : function,
        callbackObject : object);
```

ARGS: **applicationUrl**: string defining the DIAL Application Resource URL. Typically, this URL will be based on the DIAL Application URL obtained from an earlier call to the dial.getDeviceDescription() function, plus a trailing slash ('/') character if not already present, and followed by the Application Name 'HbbTV' (see [1], Section 14.7.2).

schemaValidation: if true, the harness shall do XML Schema Validation on the returned XML document, using the HbbTV DIAL Application resource schema (see [1], Section 14.7.2). In that case, if schema validation fails, then the harness shall fail the test automatically and shall not call the callback. Else if false the harness shall not do XML Schema Validation.

origin: either null, in which case the request shall not include an Origin header, or else a string, in which case the request shall include an Origin header with the value specified in that string.

callback / callbackObject: callback function invoked when the harness has successfully completed the HTTP request as:

```
void callback(
        callbackObject : object,
        app2appUrl : string or null,
        interdevSyncUrl : string or null,
        userAgent : string or null,
        allowOrigin : string or null);
```

where,

app2appUrl: string containing the value of the X_HbbTV_App2AppURL element from the HbbTV namespace in the DIAL HbbTV Application description, or null if the DUT did not return that element.

interdevSyncUrl: string containing the value of the X_HbbTV_InterDevSyncURL element from the HbbTV namespace in the DIAL HbbTV Application description, or null if the DUT did not return that element,

userAgent: string containing the value of the X_HbbTV_UserAgent element from the HbbTV namespace in the DIAL HbbTV Application description, or null if the DUT did not return that element.

allowOrigin: string containing the value of the Access-Control-Allow-Origin header returned with the DIAL HbbTV Application description, or null if the DUT did not return that header.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.6.5 JS-Function dial.startHbbtvApp()

This function sends a HTTP POST to the DUT, to request it to start an HbbTV application. This request shall conform to the DIAL specification [30] (see Section 6.1.2). If the POST request cannot be made (e.g. web server rejects connection) then the harness shall fail the test automatically and shall not call the callback. Note that a HTTP response with an error status code shall be reported to the callback, it shall not cause the test to fail automatically. If no response is received by the Test Harness within 10 seconds of making the HTTP POST request, the Test Harness shall fail the test automatically and shall not call the callback.

This function shall not do any Same-Origin or CORS [28] security checks. However, the parameters and results of this function are sufficient for a test case to test the DUT's Cross-Origin support (see HBBTV 1.4.1 [1], Section 14.8).

```
void HbbTVTestAPI.dial.startHbbtvApp(
        applicationUrl : string,
        pathToAitXml : string,
        origin : string or null,
        callback : function,
```

```
        callbackObject : object);
```

ARGS: **applicationUrl**: string defining the DIAL Application Resource URL. Typically, this URL will be based on the DIAL Application URL obtained from an earlier call to the dial.getDeviceDescription() function, plus a trailing slash ('/') character if not already present, and followed by the Application Name 'HbbTV' (see [1], Section 14.7.2).

pathToAitXml: string defining path to AIT XML relative to the directory containing the testcase XML file, using a trailing slash ('/') as a path separator. This XML file is sent to the DUT as the body of this HTTP POST.

origin: either null, in which case the request shall not include an Origin header, or else a string, in which case the request shall include an Origin header with the value specified in that string,

callback / callbackObject: callback function invoked when the harness has successfully completed the HTTP request as:

```
void callback(
        callbackObject : object,
        returnCode : int,
        contentType : string or null,
        body : string or null,
        allowOrigin : string or null);
```

where,

returnCode: integer return code in the HTTP response.

contentType: string containing the value of the content type header in the HTTP response, or null if a HTTP response is not received.

body: string containing the value of the body in the HTTP response, or null if a HTTP response is not received.

allowOrigin: string containing the value of the Access-Control-Allow-Origin header returned, or null if the DUT did not return that header.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.6.6 JS-Function dial.sendOptionsRequest()

This function sends a HTTP OPTIONS request to the DUT, to check that starting an HbbTV app can be done cross-origin (see section 14.8 of [1]). If the OPTIONS request cannot be made (e.g. web server rejects connection, or HTTP error code) then the harness shall fail the test automatically and shall not call the callback. If no response is received by the Test Harness within 10 seconds of making the HTTP OPTIONS request, the Test Harness shall fail the test automatically and shall not call the callback.

This function shall not do any Same-Origin or CORS [28] security checks. However, the parameters and results of this function are sufficient for a test case to test the DUT's Cross-Origin support (see HBBTV 1.4.1 [1], Section 14.8).

```
void HbbTVTestAPI.dial.sendOptionsRequest(
        applicationUrl : string,
        origin : string or null,
        callback : function,
        callbackObject : object);
```

ARGS: **applicationUrl**: string defining the DIAL Application Resource URL. Typically, this URL will be based on the DIAL Application URL obtained from an earlier call to the dial.getDeviceDescription() function, plus a trailing slash ('/') character if not already present, and followed by the Application Name 'HbbTV' (see [1], Section 14.7.2).

origin: either null, in which case the request shall not include an Origin header, or else a string, in which case the request shall include an Origin header with the value specified in that string.

callback / callbackObject: callback function invoked when the harness has successfully completed the HTTP request as:

```
void callback(
        callbackObject : object,
        allowOrigin : string or null,
        maxAge : string or null,
        allowMethods : string or null,
        allowHeaders : string or null);
```

where,

allowOrigin: string containing the value of the Access-Control-Allow-Origin header returned, or null if the DUT did not return that header.

maxAge: string containing the value of the Access-Control-Max-Age header returned, or null if the DUT did not return that header.

allowMethods: string containing the value of the Access-Control-Allow-Methods header returned, or null if the DUT did not return that header.

allowHeaders: string containing the value of the Access-Control-Allow-Headers header returned, or null if the DUT did not return that header.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.7 APIs for Websockets

HbbTV uses Websockets for app2app communications with a Companion Screen, and for media synchronization with a Companion Screen. So the Test Case needs a way to make the Test Harness (in its role as a "fake Companion Screen") open a Websocket to the DUT.

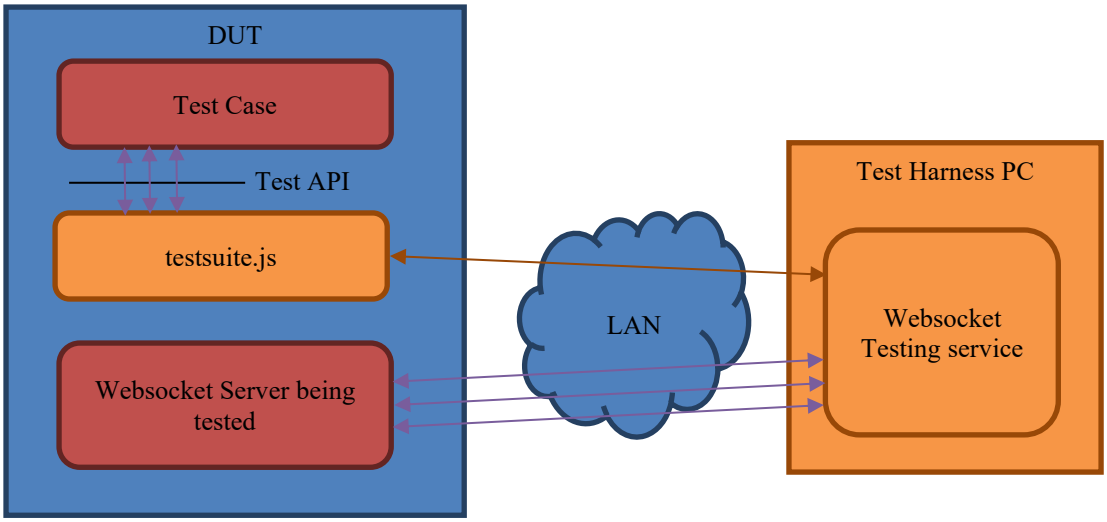
There are 3 reasons why test cases cannot just use the standard W3C Websockets API [i.6], which browsers provide and which is required by the HbbTV specification [1].

First, the W3C Websockets API is insufficient for certain HbbTV Test Cases. There are some extra low-level details that Test Cases need to be able to control and test:

- Sending 'Ping' frames and receiving the resulting 'Pong' frame
- Websocket extension testing (the 'Sec-WebSocket-Extensions' request and reply headers)
- Message fragmentation control for transmitted messages
- CORS headers (setting the 'Origin' header to arbitrary values and checking the 'Access-Control-Allow-Origin' header)
- Access to the HTTP status code when the WebSocket connection is refused

Second, if the Test Case is running as a harness-based test, the harness-based test environment is not required to provide the W3C Websockets API. Harness-based tests must use this API instead.

Thirdly, if the Test Case is running on the DUT, it's not sufficient to merely open a Websocket directly from the DUT to the media synchronization or app2app endpoints. We need to test that those endpoints are accessible via the network port on the DUT. So this API allows a Test Case to command the Test Harness to open Websockets and forward data to the test case:



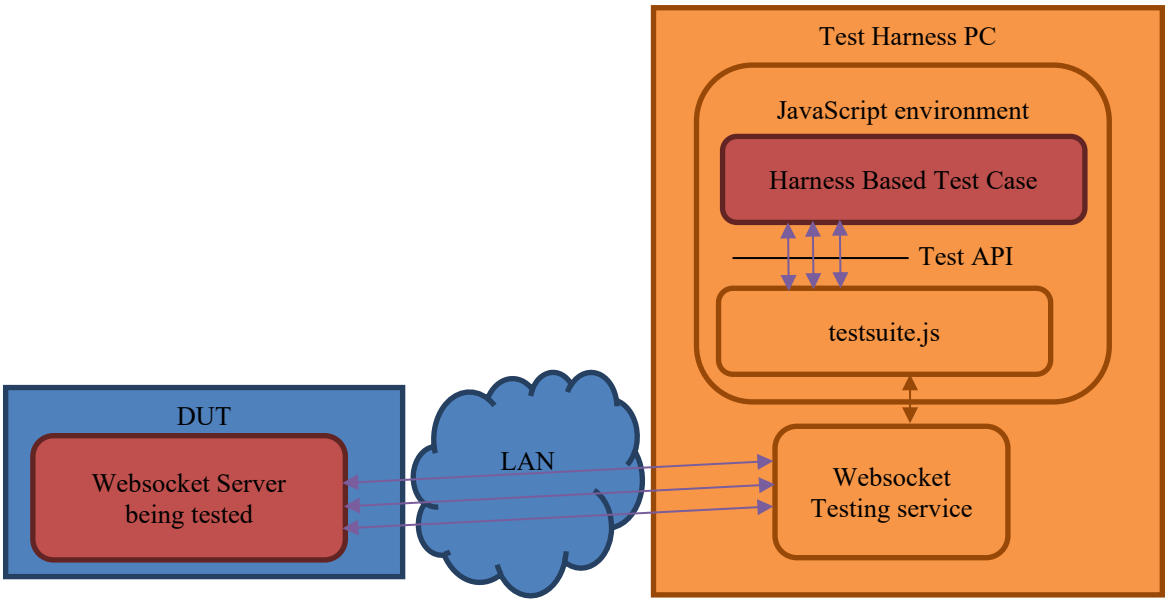
The diagram shows how a Test Case running on the DUT can make calls to testsuite.js to ask it to open Websockets. The Test Harness uses some proprietary protocol to pass those requests from testsuite.js to the Websocket Testing service that is running on the Test Harness PC. The Websocket Testing service then actually opens the Websockets. The Test Harness then forwards messages in both directions between the Test Case and the Websocket Server being tested.

Where this Test API specifies the details of Websocket messages, it refers to the Websocket between the Websocket Testing service and the Websocket Server being tested. It does NOT refer to the "proprietary protocol" part of the system, even if that happens to be implemented using a Websocket.

NOTE 1: The "proprietary protocol" is likely to be a Websocket connection. This should be a single Websocket connection, with all the requests multiplexed down it. It should NOT be one Websocket for each call to openWebsocket(). Using a single Websocket here ensures that, when the Test Case tries to open 400 Websockets at once, the only DUT restrictions that are hit are restrictions in the Websocket Server that's being tested, not restrictions on how many Websockets you can open in the browser.

NOTE 2: The "proprietary protocol" is just an RPC mechanism – i.e. when the Test Case requests a particular Websocket message be sent with certain options, testsuite.js may just take the message and all the options, and send them as JSON down a Websocket to the Websocket Testing service.

This API can also be used by a test case running as a Harness Based Test:



For security reasons, if the harness follows the design described here, then when there is a new “proprietary protocol” Websocket connection to the the Websocket Testing Service in the Test Harness, that service should check that either the Origin header of the HTTP request is one of the test domains listed in section 5.2.1.2 (i.e. hbbtv1.test etc), or the connection is coming from a harness-based test case. See RFC 6454 and the HbbTV specification [1] for details of how a web browser determines the Origin of a page.

NOTE 3: This means that test pages loaded from DSM-CC or from a server on the Internet cannot use the openWebsocket() API. Tests that use the openWebsocket() API must be HTML pages loaded from http://hbbtv1.test/ or from the other DNS names in section 5.2.1.2.

NOTE 4: The Websocket Testing service is a Websocket proxy that allows spoofing the Origin header. This is a powerful tool that we do not want to be used by attackers. The Origin check ensures that only Test Cases can use it.

Without the Origin check, there is the following security vulnerability: Suppose an organization happens to have a Test Harness installed somewhere. Suppose an attacker who wishes to attack that organization knows that they have a Test Harness installed, and knows either the IP address of the Test Harness or a range of IP addresses that will include the Test Harness, and knows the proprietary protocol used by that Test Harness. Further, suppose that the organization has an intranet server inside their firewall that uses Websockets. The attacker first tricks someone at the organization into visiting a malicious webpage (which is trivial for the attacker to arrange via a malicious link in an email). The malicious webpage instructs their desktop web browser to make a Websocket connection to the Test Harness, and then sends commands over that Websocket instructing the Test Harness to connect to a secure internal server using a spoofed Origin header. That allows the attacker to get Websocket access to an internal server (inside the corporate firewall). To prevent this vulnerability, the Test Harness has to reject the initial Websocket connection from the malicious page, based on the Origin header supplied by the browser.

Of course, an Origin check does not preclude connections made by non-browser based clients.

At the start of a testcase execution, there shall not be any Websockets open via this mechanism.

NOTE 5: When a test case stops running, and the terminal destroys the test application, the terminal will close any Websockets open via the normal W3C Websocket API. (See the W3C Websocket API spec for details). If the test harness uses a W3C Websocket in its implementation of this API, then the Websocket Testing service can detect that W3C Websocket being closed and close any related Websocket connections it's created.

For harness-based tests, a similar process can occur but it will be triggered by the harness-based test execution environment being torn down.

It is not possible for a test case to guarantee to close all Websocket connections made with this API in all possible failure cases. Therefore the harness must close them. Because the harness has to close them anyway, it's not worth writing extra code in every test case to close them in the success case and/or certain failure cases.

7.7.1 Encoding of binary data

Where binary data is returned from this API, it is encoded as follows: It is converted to a string by converting each byte in turn using these rules:

- Bytes 0x20 to 0x24 inclusive are mapped to the corresponding characters U+0020 to U+0024 inclusive.
- Bytes 0x26 to 0x7E inclusive are mapped to the corresponding characters U+0026 to U+007E inclusive.
- Other bytes are mapped to the three character sequence consisting of a percent character followed by two uppercase hex characters, e.g. byte 0 maps to “%00” and byte 0xAB maps to “%AB”.

Where binary data is passed into this API, it is passed as a string which is decoded as follows:

- Characters U+0020 to U+0024 inclusive are mapped to the corresponding bytes 0x20 to 0x24 inclusive.
- Characters U+0026 to U+007E inclusive are mapped to the corresponding bytes 0x26 to 0x7E inclusive.
- The three character sequence consisting of a percent character followed by two uppercase hex characters maps to the corresponding byte, e.g. “%00” maps to byte 0 and “%AB” maps to byte 0xAB.
- A percent character that is not followed by two uppercase hex characters means the string is malformed.
- If the string contains a character outside the range U+0020 to U+007E inclusive then the string is malformed

Test Cases shall not pass malformed strings (as defined above) to APIs that are expecting to be able to convert the string to binary data. The handling of malformed strings is test-harness-dependent (but a reasonable choice would be to fail the test with an error).

NOTE 1: This uses a different string encoding from getPlayoutInformation. Experience has shown that test case authors try to print the strings and pass them to test API calls, which does not work for the binary strings returned by getPlayoutInformation. A human-readable string representation is easier to use. Additionally, the string returned by getPlayoutInformation is intended to be passed to an OIPF API that defines the encoding; that restriction does not apply here.

NOTE 2: These APIs uses a string encoding rather than an ArrayBuffer because that is simpler and easier for test case authors to use.

7.7.2 JS-Function openWebsocket()

This function establishes a WebSocket connection (RFC 6455 [31]) to the specified Websockets URL.

```
void HbbTVTestAPI.openWebsocket(
        url : string,
        onConnect : function or null,
        onMessage : function or null,
        onPong : function or null,
        onClose : function or null,
        onFail : function or null,
        callbackObject : object,
        originHeader : string or null or undefined,
        websocketsExtensionHeader: string or null or undefined);
```

ARGS: **url**: string defining the Websockets URL. If this URL does not start "ws://" then the connection shall fail, and the onFail callback shall be called.

onConnect: function called when "The WebSocket Connection is Established" as defined in RFC 6455 [31]. If the connection fails then this function will not be called – onFail will be called instead. If the connection succeeds, this is always the first callback that is called.

```
void onConnect(
        callbackObject : object,
        websocketExtensionHeader : string or null,
        websocket : WebsocketClient);
```

where,

websocketExtensionHeader: the value of the Sec-WebSocket-Extensions header sent by the server, or null if that header was not sent. If the server sends multiple Sec-WebSocket-Extensions headers, they are combined into a single value as described in RFC 6455 [31] section 9.1. This is the value sent by the server, regardless of whether the extensions are supported by the test harness or not.

websocket: a new WebsocketClient object.

onMessage: function called when "A WebSocket Message Has Been Received" as defined in RFC 6455 [31]:

```
void onMessage(
        callbackObject : object,
        data : string,
        binary : boolean,
        websocket : WebsocketClient);
```

where,

data: if the message is a text message, 'data' is the message which has been decoded from UTF-8 into a JavaScript string. If the message is a binary message, then "data" is the message encoded as per section 7.7.1.

binary: true if the message is a binary message, or false if the message is a text message.

onPong: Called when a Pong frame is received. This may be an unsolicited Pong frame, or it may be a reply to a Ping frame that was sent via WebsocketClient.sendPing().

```
void onPong(
        callbackObject : object,
        data : string,
        websocket : WebsocketClient);
```

where,

data: the data from the Pong frame encoded as per section 7.7.1.

websocket: the WebsocketClient object.

onClose: function called when "The WebSocket Connection is Closed" as defined in RFC 6455 [31], except this is not called if onFail has already been called. Once this method is called, the test must not make any further method calls on this WebsocketClient object, and the harness must not invoke any other callbacks on this WebsocketClient object.

```
void onClose(
        callbackObject : object,
        statusCode : integer,
        reason : string,
        websocket : WebsocketClient);
```

where,

statusCode: "The WebSocket Connection Close Code" as defined in RFC 6455 [31]

reason: "The WebSocket Connection Close Reason" as defined in RFC 6455 [31]

websocket: the WebsocketClient object.

onFail: function called when the Websocket implementation has to "Fail the WebSocket Connection" as defined in RFC 6455 [31]. Once this method is called, the test must not make any further method calls on this WebsocketClient object, and the harness must not invoke any other callbacks on this WebsocketClient object.

```
void onFail(
        callbackObject : object,
        statusCode : integer,
        reason : string,
        websocket : WebsocketClient);
```

where,

statusCode: If the Websocket handshake fails due to the HTTP status code not being 101, then this shall equal the HTTP status code that was received (e.g. 503). In all other failure cases it shall equal null.

reason: A human-readable string describing the error. E.g. the test case may choose to fail the test and use this string as part of the failure reason.

websocket: the WebsocketClient object.

originHeader: string to be sent as the value of the WebSockets "Origin" header field. If it is null, or not specified, then that header is not sent.

websocketExtensionHeader: string to be sent as the value of the WebSockets "Sec-WebSocket-Extensions" header field. If it is null, or not specified, then that header is not sent. If this header specifies extensions that are not supported by the test harness, and the server decides to use such an extension, then the test harness shall proceed as if that extension had not been negotiated. (Informative note: In that case, depending on the extension, it is likely that the Websocket connection will be established successfully, but fail with a call to onFail as soon as the server sends data that is not valid according to RFC 6455 [31], but would have been valid according to that extension).

The WebsocketClient object passed to the onConnect() and other callbacks has the methods defined in the following subsections.

NOTE 1: The openWebsocket function does not return a WebsocketClient object.

Tests that use this openWebsocket() API must either be a web page with an Origin that is one of the test domains listed in section 5.2.1.2 (i.e. http://hbbtv1.test/ etc), or be a harness-based test case. (See section 7.7 for rationale).

If the test harness receives a ping frame on a Websocket opened with this API, it shall automatically respond with a pong frame as defined in RFC 6455 [31]. Those ping requests and pong responses are not exposed via this API.

The test harness shall not send unsolicited pong frames. The test harness shall not send ping frames unless instructed to do so via WebsocketClient.sendPing().

The test harness is not required to implement any Websockets extensions.

This API shall support having at least 500 simultaneous open Websockets.

NOTE 2: For tests running on the DUT, if the Test Harness provided implementation uses Websockets internally, this implies some sort of multiplexing, because the DUT is only required to support 20 Websockets connections (See HbbTV [1] section 10.2.1).

A test case may call this function multiple times without waiting for a response, e.g. if testing what happens with 400 rapid Websocket connection attempts.

Test implementers should not call this function from a test case running on the DUT when the network connection is configured to be down. Test implementers should not take the network connection down when a test case running on the DUT still has a Websocket open vial this API. Those actions may cause the complete test to fail automatically. These restrictions do not apply to test cases running as a Harness Based Test.

7.7.3 JS-Function WebsocketClient.sendMessage()

This function sends a Websocket message.

```
void WebsocketClient.sendMessage(
        data : string,
        binary : boolean,
        fragments : array of integers or null or undefined);
```

ARGS: **data**: If 'binary' is false, the message as a string, which will be UTF-8 encoded for transmission. If 'binary' is true, the message encoded as specified in section 7.7.1.

binary: true if the message should be sent as a binary message, or false if the message should be sent as a text message.

fragments: if null or undefined or not specified, guarantees not to fragment the message. If specified, it shall be an array of strictly positive integers that add up to the size of the message in bytes. The message is fragmented into the requested size fragments.

7.7.4 JS-Function WebsocketClient.sendPing()

This function sends a Ping frame (as per section 5.5.2 of the WebSocket protocol [31]) containing the requested data. If the server correctly replies with a Pong frame, then onPong() will be called.

```
void WebsocketClient.sendPing(
        data : string);
```

ARGS: **data**: string containing the binary data to use as the payload of the Ping frame, encoded as specified in section 7.7.1.

7.7.5 JS-Function WebsocketClient.close()

This function shall "Start the WebSocket Closing Handshake" as defined in RFC 6455 [31], with no code or reason. When the close is complete, the onClose callback will be called if the connection could be closed gracefully, or onFail will be called if the connection fails.

NOTE: It is always possible to close a Websocket connection by closing the underlying socket, so a close cannot fail as such. However, there is a 'graceful close' protocol defined by RFC 6455, that should be followed unless there is an error. In a 'graceful close', both sides send a Close frame and shut down the sending side of their side of their socket, then the socket is closed and onClose is called. If either side resorts to closing the underlying socket without sending a Close frame then onFail is called.

```
WebsocketClient.close();
```

After calling WebsocketClient.close() the test must not make any further method calls on this WebsocketClient object. However, the harness may invoke further callbacks on this WebsocketClient object (e.g. if a message is received before the server's close frame, then the onMessage callback will be called).

7.7.6 JS-Function WebsocketClient.tcpClose()

This function shall cause the harness to close the Websocket without sending a close frame but by simply closing the underlying TCP/IP socket.

```
void WebsocketClient.tcpClose(
        callback: function or null,
        callbackObject : object);
```

ARGS: **callback / callbackObject**: callback function invoked when the Test Harness has successfully closed the Websocket's underlying TCP/IP socket.

```
callback(callbackObject : object, websocket : WebsocketClient)
```

where

websocket: the WebsocketClient object.

This function shall return immediately, and arrange for the harness to close the Websocket's underlying TCP/IP socket as soon as possible. When the underlying TCP/IP socket is closed, a callback will be called (see below for which callback).

After calling WebsocketClient.tcpClose() the test must not make any further method calls on this WebsocketClient object. However, the harness may invoke further callbacks on this WebsocketClient object to report events that happened before the socket was actually closed (e.g. if a message is received before the underlying socket was closed, then the onMessage callback will be called).

After tcpClose() is called, then either:

- If the Websocket connection fails or is gracefully closed before the harness could close the TCP/IP socket, then the onFail or onClose callback shall be called as normal. In that case, the callback passed to tcpClose() shall not be called.
- Otherwise, the harness closes the Websocket's underlying TCP/IP socket in response to the tcpClose() call, then calls the callback that was passed to tcpClose().In this case, the onFail and onClose callbacks shall not ever be called.

Once the harness has called onFail, onClose, or the callback passed to tcpClose(), then the harness shall not make any further callbacks for this WebsocketClient object.

7.8 APIs for Media Synchronization testing

7.8.1 JS-Function analyzeAvSync ()

This function checks the synchronization between two or three of: video, second video, subtitles, and audio. It is used with video and subtitles streams containing flashes, and audio streams containing tone bursts, as described in section 5.2.1.13.

```
void HbbTVTestAPI.analyzeAvSync(
            step_number : integer,
            comment : string,
            checkLight1AgainstAudio : boolean,
            checkLight2AgainstAudio : boolean,
            checkLight1AgainstLight2 : boolean,
            maxDifference : integer,
            callback : function or null,
            callbackObject : object);
```

This function is deprecated and must not be used in new test cases.

Its behaviour shall be identical to:

```
var temp = this.createAnalyzeAvSync(step_number, comment);

if (checkLight1AgainstAudio) {
    temp.checkLight1AgainstAudio(maxDifference, maxDifference);
}
if (checkLight2AgainstAudio) {
    temp.checkLight2AgainstAudio(maxDifference, maxDifference);
}
if (checkLight1AgainstLight2) {
    temp.checkLight1AgainstLight2(maxDifference, maxDifference);
}

temp.analyze(callback, callbackObject);
```

If updating a test case to not use this API, you are recommended to use chained calls rather than a temporary variable, e.g.:

```
testApi.createAnalyzeAvSync(step_number, comment).checkLight1AgainstAudio(maxDifference,
maxDifference).analyze(callback, callbackObject);
```

See section 7.8.8 for details of those APIs.

7.8.2 JS-Function analyzeStartVideoGraphicsSync ()

This function checks the synchronization between graphics and either video or subtitles.

```
void HbbTVTestAPI.analyzeStartVideoGraphicsSync(
            stepId : integer,
            comment : string,
            callback : function,
            callbackObject : object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step

callback/callbackObject: a callback function to invoke when the analysis was started (also see chapter 7.2.3 Callbacks). The callback function will be called with the following parameters: callback(callbackObject, analyzeFinishedVideoGraphicsSync) where analyzeFinishedVideoGraphicsSync is a function that can be called as defined below.

The analyzeFinishedVideoGraphicsSync function is provided via the callback, it is NOT a function on the HbbTVTestAPI class. (This design allows Test Harness implementers to return a closure that links the 'finished' call back to the corresponding 'start' call).

```
void analyzeFinishedVideoGraphicsSync(
            expectedDeltaMillis : integer,
            maxDifferenceMillis : integer,
            callback : function or null,
            callbackObject : object)
```

ARGS: **expectedDeltaMillis**: the number of milliseconds that the flash on light sensor 1 is expected to be ahead of the flash on light sensor 2. May be negative if light sensor 2 is expected to detect a flash first.

maxDifferenceMillis: maximum allowed error in the measured delta, in milliseconds.

callback/callbackObject: a callback function to invoke when the observation is complete and, if the Test Harness chooses to do the analysis immediately, the analysis is also complete (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must causes the test harness to fail the complete test, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording.

When calling these functions, the typical procedure is:

1) The Test Case arranges for a video player to be positioned so it completely covers light detection area 1, and a black graphics rectangle to be positioned so it completely covers light detection area 2. The video player may be larger than the light detection area, e.g. if the video contains other information to help debugging as well as the flash area.

2) The Test Case starts playback of a video with a single video flash at a known timeline position. (In this section 7.8.2, "timeline" should be read generally as referring to any way that the Test Case can know the video playback position; it does not refer to any particular way of doing that).

3) The Test Case calls analyzeStartVideoGraphicsSync()

4) That causes the Test Harness to start analysing the two light sensors

5) The Test Harness then calls the callback passed to analyzeStartVideoGraphicsSync(), to indicate that analysis has started.

6) The Test Case checks the video's timeline time. If the call to analyzeStartVideoGraphicsSync() took too long, and the video flash has been missed, then the Test Case calls reportStepResult() to fail the test case, and terminates.

7) The Test Case regularly monitors the video's timeline time. When the video's timeline time gets 'close' to the video flash time, the Test Case records the current video timeline time, initiates a graphics flash by turning the graphics rectangle white, and then records the current video timeline time again. (Note: It's not possible for an HbbTV application to reliably change the graphics at a particular timeline time, so we cannot use analyzeAvSync. The best we can do is 'close', but this procedure carefully measures and compensates for this error).

8) A short time later (typically 2-5 video frames later), the Test Case clears the graphics flash by turning the graphics rectangle black.

9) The Test Case waits for sufficient time for both the following to happen:

    a. the graphics changes to get through all output buffering and actually be displayed as light

    b. the video flash to happen, (remembering to add the acceptable synchronization inaccuracy to the time it's expected to happen), and that flash to get through all output buffering and actually be displayed as light.

10) The Test Case calculates the average of the two timeline times around when it actually initiated the graphics flash, and subtracts that from the timeline time the video flash is encoded, and converts the result to milliseconds. It may then need to apply some corrections (e.g. see the Note below). This is the expectedDeltaMillis.

11) The Test Case converts to milliseconds and adds up: the allowed tolerance as specified in the assertion, half the difference between the two timeline times measured around when it actually initiated the graphics flash, and any other corrections (e.g. see the Note below). This is the maxDifferenceMillis.

12) The Test Case calls the analyzeFinishedVideoGraphicsSync function provided by the Test Harness. The Test Case passes the calculated expectedDeltaMillis and maxDifferenceMillis values.

13) That causes the Test Harness to stop analysing the two light sensors

14) The Test Harness either:

    a. Determines that each light sensor detected a single flash, and the difference between the start times of the flashes was the expected value (within the specified tolerance). In this case, the analysis has passed, and the Test Harness calls the callback provided by the Test Case.

    b. Determines that the conditions in (a) do not hold. In this case, the analysis has failed, and the Test Harness shall fail the entire Test Case immediately and shall not call the callback.

    c. Records the information needed to make that determination for later off-line analysis (e.g. if using a video camera to capture the screen and doing manual analysis later), and calls the callback provided by the Test Case. In this case, the Test Case may be failed when the analysis is carried out, which may be long after the Test Case has finished executing.

The Test Case shall ensure that the time that the Test Harness is observing the light sensors (between step 4 and step 13 in the above procedure) is less than 20 seconds. If this limit is exceeded (because the Test Case does not call analyzeFinishedVideoGraphicsSync quickly enough), the Test Harness *may* fail the entire test case. In that case, if/when the Test Case does eventually call analyzeFinishedVideoGraphicsSync, the callback shall not be called. (Rationale: this places a limit on the memory the Test Harness needs to do the capture).

The Test Harness shall ensure that, given reasonable assumptions about the performance of the DUT, a call to analyzeStartVideoGraphicsSync() takes less than 5 seconds, measured from the start of the call until the callback function is called. The time the Test Harness starts analysing the light sensors shall be somewhere between those two points.

The Test Harness shall ensure that, given reasonable assumptions about the performance of the DUT, a call to analyzeFinishedVideoGraphicsSync takes less than 2 seconds, measured from the start of the call until the Test

Harness stops analysing the two light sensors. However, there may be a significant delay before the Test Harness calls the callback – perhaps minutes if a human analysis is involved.

(Informative Rationale: the guarantees provided by these 2 paragraphs are needed to allow the Test Case author to choose where in the video file the flash should occur, and so the Test Case author can ensure they meet the 20 second limit. E.g. the Test Case author may place the flash at 12 seconds into the video file, and call analyzeFinishedVideoGraphicsSync at approximately 15 seconds into the video file. If after starting the video file they get a callback immediately to say playback has started, and then analyzeStartVideoGraphicsSync runs instantly, but analyzeFinishedVideoGraphicsSync takes 2 seconds to stop the observation, then they are still well inside the 20 second limit. Conversely, if after starting the video file it takes 5 seconds for the DUT to make the callback to say playback has started, and then 5sec for analyzeStartVideoGraphicsSync to run, then those steps are still complete 2 seconds before the flash).

If either light sensor detects no flashes, or if either light sensor detects more than 1 flash, then the test harness shall treat that as an analysis failure.

See section 5.2.1.12 for details of the light sensor positions. See section 5.2.1.13 for details of media requirements – of the two alternatives for media described in that section, this API requires media with a single flash.

NOTE: When calculating the tolerance to allow for this function, the Test Case should bear in mind that some of the HbbTV APIs give the time of the last video frame that was composited with graphics, and by the time the video/graphics compositor runs again it may or may not be onto the next video frame. (E.g. with 50fps video, a DUT with a 50fps display will always go onto the next video frame, and a DUT with a 200Hz display that chooses to composite graphics with video at 200Hz will only have a 25% chance of going onto the next video frame). If you are particularly unlucky, the video/graphics compositor may be running in parallel with your JavaScript that changes the graphics, so your graphics change may have to wait 2 video frames. To allow for this potential error of -0 / +2 video frames, you may wish to adjust the expected difference by 1 frame and increase the tolerance by 1 frame.

The entry/entries in the Test Report XML for this step shall include the measured difference in milliseconds between observed and expected times.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: This function checks the start time of the flash, unlike analyzeAvSync/analyzeAvNetSync. Experimental results show that typical TV “picture improvement” processing can affect the detection of the flash start and end times, and when testing audio/video sync, choosing the mid-point makes the tests more robust. However, that does not apply to this API because it is not testing audio; instead it is testing the video and graphics planes which will probably be equally affected. Choosing to test the start of the flash makes the test cases simpler, as the end of the graphics flash does not need to be timed accurately.

7.8.3 JS-Function analyzeAvNetSync()

For a DUT that is in inter-device synchronization master mode, this function checks the synchronization between the timeline reported by the DUT and video, audio and/or subtitles.

```
void HbbTVTestAPI.analyzeAvNetSync(
            stepId : integer,
            comment : string,
            wcUrl : string,
            tsUrl : string,
            contentIdStem : string,
            timelineSelector : string,
            unitsPerTick : integer,
            unitsPerSecond : integer,
            patternStart : integer,
            patternRepeatLength : integer,
            pattern : integer[],
            checkLight1AgainstTimeline : boolean,
            checkLight2AgainstTimeline : boolean,
            checkAudioAgainstTimeline : boolean,
```

```
maxDispersionMillis : integer,
maxExtraDifferenceMillis : integer,
callback : function or null,
callbackObject : object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step

wcUrl: the URL of the CSS-WC service that the Test Harness shall connect to. The Test Case should obtain this value from the DUT's CSS-CII service using the Websockets API in section 7.7. This must be a "udp://" URL.

tsUrl: the URL of the CSS-TS service that the Test Harness shall connect to. The Test Case should obtain this value from the DUT's CSS-CII service. This must be a "ws://" URL.

contentIdStem: The "contentIdStem" the Test Harness shall specify in the Setup Data when it connects to the CSS-TS service. The Test Case may have this hardcoded, or it may retrieve it from the DUT's CSS-CII service. See [34] section 5.7.3 for the meaning of this parameter.

timelineSelector: The "timelineSelector" the Test Harness shall specify in the Setup Data when it connects to the CSS-TS service. The Test Case may have this hardcoded, or it may retrieve it from the DUT's CSS-CII service. See [34] section 5.7.3 for the meaning of this parameter.

unitsPerTick: The "unitsPerTick" that the Test Harness shall use to interpret the timeline it receives from the CSS-TS service. The Test Case may retrieve this from the DUT's CSS-CII service. See [34] section 5.5.9.5 for the meaning of this parameter.

unitsPerSecond: The "unitsPerSecond" that the Test Harness shall use to interpret the timeline it receives from the CSS-TS service. The Test Case may retrieve this from the DUT's CSS-CII service. See [34] section 5.5.9.5 for the meaning of this parameter.

patternStart: the time the repeating pattern of flashes and tones starts, in timeline ticks. This refers to the mid-point of the first flash and tone in the pattern.

patternRepeatLength: the length of the repeating pattern of flashes and tones, in timeline ticks

pattern: the mid-point times of the repeating pattern of flashes and tones, in timeline ticks; the first point should be 0.

checkLight1AgainstTimeline: if true, then light sensor 1 is checked against the timeline. (See section 5.2.1.12 for details of the light sensor positions).

checkLight2AgainstTimeline: if true, then light sensor 2 is checked against the timeline.

checkAudioAgainstTimeline: if true, then the audio sensor is checked against the timeline.

maxDispersionMillis: maximum allowed CSS-WC dispersion, in milliseconds.

maxExtraDifferenceMillis: maximum allowed synchronisation error, in milliseconds, before accounting for inaccuracies in the CSS-WC time. The Test Harness shall calculate the CSS-WC dispersion and add the dispersion to maxExtraDifferenceMillis to get the maximum allowed synchronization error.

callback/**callbackObject**: a callback function to invoke when the observation is complete and, if the Test Harness chooses to do the analysis immediately, the analysis is also complete (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis causes the test harness to fail the complete test, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording.

The Test Case must set at least one of the 3 'check…' options, and may set 1, 2, or all 3 of them.

This function shall return immediately.

The Test Harness shall ensure that, given reasonable assumptions about the performance of the DUT, a call to analyzeAvNetSync() starts analysing the light and/or audio sensor within 5 seconds from the start of the call. Note: That 5 seconds includes the time needed to do the initial synchronization of the Test Harness's clock with the CSS-WC server, and the time to establish the CSS-TS connection and get the first timeline information.

The Test Harness shall analyze the light and/or audio sensors for a period of 15 seconds.

The media playing on the DUT shall contain a repeating pattern of tones and flashes. This may be the pattern described in section 5.2.1.13, or it may be a different pattern. The Test Case describes the pattern to the Test Harness by specifying the 'patternStart', 'patternRepeatLength', and 'pattern' parameters. If there are N entries in the 'pattern' array, then the mid-points of the tones and flashes are expected at the following times (measured in timeline ticks):

- patternStart + pattern[0]
- patternStart + pattern[1]
- patternStart + pattern[2]
- …
- patternStart + pattern[N-2]
- patternStart + pattern[N-1]
- patternStart + patternRepeatLength + pattern[0]
- patternStart + patternRepeatLength + pattern[1]
- …
- patternStart + patternRepeatLength + pattern[N-2]
- patternStart + patternRepeatLength + pattern[N-1]
- patternStart + patternRepeatLength * 2 + pattern[0]
- patternStart + patternRepeatLength * 2 + pattern[1]
- …
- patternStart + patternRepeatLength * 2 + pattern[N-2]
- patternStart + patternRepeatLength * 2 + pattern[N-1]
- patternStart + patternRepeatLength * 3 + pattern[0]
- ...

The pattern keeps repeating until at least 20 seconds after this function was called. (The 20 seconds here is calculated from the 5 second limit to start observing the sensors plus the 15 seconds of observation).

All entries in the pattern[] array must be non-negative and strictly less than patternRepeatLength. The pattern[] array must be strictly increasing. I.e.:

0 <= pattern[0] < pattern[1] < pattern[2] …. < pattern[N-1] < patternRepeatLength.

The decision about whether this analysis passes or fails is made as follows:

- If the Test Harness could not obtain the time from the CSS-WC service on the DUT, the test fails.
- If the calculated CSS-WC dispersion, in milliseconds, exceeds maxDispersionMillis the test fails.
- If the Test Harness could not connect to the CSS-TS service or could not obtain the timeline from that service, then the test fails.

- If there is a flash or tone detected, and its mid-point is more than 100ms from both the start and end of the 15 second observation period, then the Test Harness checks the pattern to see if a flash or tone is expected at that time (within the calculated tolerance). If not, then the test fails.
- If there is a flash or tone that is expected, and the time it is expected is more than the calculated tolerance plus 100ms from both the start and end of the 15 second observation period, and there is no flash or tone detected at that time (within the calculated tolerance), then the test fails. (Note: The special handling of the start and end of the observation period accounts for cases where the synchronization is not perfect but is within the specified tolerance, and the Test Harness starts monitoring just before a flash is expected but just after the flash has happened, so the flash is missed).
- If two flashes or two tones are detected from the same source within a 400ms window, that also causes the test to fail. (Note: that means spurious flashes have been detected, either due to a failure of the DUT or a failure in the test harness).
- If none of the above conditions apply, then this step passes.

The Test Harness shall not send any Actual, Earliest or Latest Presentation Timestamp to the DUT's CSS-TS service.

The Test Harness shall close the CSS-TS connection when capture is complete, and before calling the callback.

The entry/entries in the Test Report XML for this step shall include the calculated CSS-WC dispersion, and the maximum observed difference in milliseconds between observed and expected times for a flash or beep.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

NOTE: This function checks the mid-point of the flash. Experimental results show that typical TV "picture improvement" processing can affect the detection of the flash start and end times. Experiments suggest that the start time of the flash is affected more than the mid-point is. Particularly when testing audio/video sync, choosing the mid-point makes the tests more robust.

7.8.4 JS-Function startFakeSyncMaster()

Starts a fake inter-device synchronization master running on the Test Harness, which the DUT can connect to.

```
void HbbTVTestAPI.startFakeSyncMaster(
            ciiData : object,
            timelineStartValue : integer,
            callback : function,
            callbackObject : object,
            timelineStartSeconds : integer or null,
            timelineStartMicroseconds : integer or null);
```

ARGS: **ciiData**: a JavaScript dictionary containing the data to be returned by the CSS-CII service, except the timeline service and wallclock service URLs do not need to be specified (and will be ignored if they are specified). The implementation of this API must convert the ciiData to JSON using the standard JavaScript JSON.stringify() method, with no replacer specified.

timelineStartValue: the initial value of the timeline time reported by the fake CSS-TS service.

callback/**callbackObject**: a callback function to invoke when the fake synchronization master is started (also see chapter 7.2.3 Callbacks). The callback function will be called with the following parameters: callback(callbackObject : object, ciiUrl : string, syncTestUrl : string)

timelineStartSeconds/**timelineStartMicroseconds**: If both these parameters are non-null, they represent a UNIX timestamp in the past. The microseconds value shall be in the range 0-999999 inclusive. This timestamp is interpreted using the same clock as getPlayoutStartTime(). This timestamp specifies when the timeline starts. If both these parameters are null, the timeline starts when the harness processes this API call, which must be after this function is called and before the callback is called. Test cases shall either specify both of these values as null or specify both of them as integers. Test cases shall not pass a time in the future.

In the description below, a timeline time is computed as follows:

- First, we have to determine the wallclock time when the timeline started. If the timelineStartSeconds and timelineStartMicroseconds parameters are non-null, then they specify this time directly. Otherwise, the test harness chooses the wallclock time when the timeline starts; this start time shall be after this function is called and before the harness calls the callback.

- Next, we have to choose a timeline. The possible timelines are listed in the 'timelines' array in the ciiData passed to startFakeSyncMaster(). For a connection to the CSS-TS service, the timeline will be chosen by comparing the 'timelineSelector' in the setup data against the 'timelineSelector' from each of the entries in the 'timelines' array; the matching entry is used. For the syncTestUrl service, the first timeline in the 'timelines' array is always used. A Test Case that does not list any timelines in the 'timelines' array must not use the syncTestUrl.

- Then the timeline time is calculated as follows: measure the time since the timeline started, in milliseconds; multiply that by 'unitsPerSecond', divide by 'unitsPerTick' and divide by 1000; round the result down to an integer (rounding towards negative infinity); add 'timelineStartValue'. For this calculation. 'unitsPerTick' and 'unitsPerSecond' are taken from the matched entry in the 'timelines' array.

The ciiUrl parameter passed to the callback is the URL of the CSS-CII service provided by the Test Harness. The Test Case can use this URL to configure the DUT as a slave.

The Test Case can make a Websocket connection to the Test Harness using the URL passed as the syncTestUrl parameter to the callback. The protocol on this Websocket is as follows: The only valid message from the Test Case to the Test Harness is a text message where the payload is the string "X". Sending that message causes the Test Harness to respond with the timeline time that the X was received, calculated as described above, as a single decimal integer encoded as text in a Websocket message. The Test Harness will not send any other messages over the Websocket. If the Test Case sends any other message over the Websocket then the behaviour of the Test Harness is not defined by this specification. Future versions of this specification may define other messages.

The fake master services provided by the Test Harness behaves as follows:

- CSS-CII: When a client connects, sends a single notification containing the ciiData specified in the call to startFakeSyncMaster(), with the addition of the correct URLs for the fake CSS-TS and CSS-WC services. Never reports any changes (i.e. does not send any other notifications).

- CSS-WC: Reports a wallclock time advancing at the correct rate. This service shall meet the requirements of HbbTV [1] section 13.7.2 and 13.7.3, with the word "terminal" replaced by "Test Harness".

- CSS-TS: Behaves as follows:

    1. On getting a connection, it waits for the client to send setup data

    2. It checks if the 'contentIdStem' in the setup data matches or is a prefix of the 'contentId' specified in the ciiData passed to startFakeSyncMaster(). It also checks if the 'timelineSelector' in the setup data matches the 'timelineSelector' from one of the entries in the 'timelines' array specified in the ciiData passed to startFakeSyncMaster().

    3. If both the conditions from step 2 hold, then: Within 500ms of receiving the setup data it sends a Control Timestamp to the client. This shall contain the current 'wallClockTime'. It shall contain 'timelineSpeedMultiplier' set to 1. It shall also contain a 'contentTime' that is a timeline time calculated as described above. The harness shall ensure that the wallClockTime and contentTime refer to the same instant in time.

    4. If the conditions from step 2 do not both hold, then: Within 500ms of receiving the setup data it sends a Control Timestamp to the client. This shall contain the current 'wallClockTime'. It shall contain a 'contentTime' and 'timelineSpeedMultiplier' which are both null.

    5. Any Actual, Earliest and Latest Presentation Timestamp sent by the client are read and ignored

The Test Harness shall automatically terminate the fake inter-device synchronization master when it stops running the test, before starting the next test.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.8.5 JS-Function getPlayoutStartTime()

Gets the time that DVB TS playout started. This is the time that the first byte of the relevant DVB TS was sent out of the modulator as a RF signal. The "relevant" DVB TS is the TS generated by the Test Harness from the current playoutset,

```
void HbbTVTestAPI.getPlayoutStartTime(
            multiplexIndex : integer,
            callback : function,
            callbackObject : object);
```

ARGS: **multiplexIndex**: the index of the multiplex to query. The Test Case must pass zero for this parameter to refer to the first multiplex and must pass one to refer to the second multiplex. If the Test Case specifies an invalid value, the Test Harness may fail the complete test case.

callback/callbackObject: a callback function to invoke with the playout start time (also see chapter 7.2.3 Callbacks). The callback function will be called with the following parameters: callback(callbackObject : object, seconds : integer, microseconds : integer, maxErrorMicroseconds : integer)

This function returns immediately and arranges for the callback to be called as soon as possible.

NOTE 1: Some implementations may involve asking the user to type in the value, so "as soon as possible" may be in a minute or two. If the Test Harness automates TS playout then it may be able to respond in significantly less than a second.

The timestamp passed to the callback specifies a UNIX timestamp, split into an integer number of seconds and 0 to 999999 microseconds. The Test Harness must ensure this time is accurate within +/- 250 milliseconds. The error bound shall be passed to the test as maxErrorMicroseconds, which must be between 0 and 250000 inclusive. This allows the Test Case to make allowance for the error.

NOTE 2: Although this call allows microsecond precision, the value is not expected to be that accurate. If a human is starting the playout, an accuracy of +/- 250 millisecond is achievable. If the Test Harness automates TS playout then it may be able to report the value more accurately.

PHP scripts in the Test Case can access a clock via the PHP time() or microtime() functions. The Test Harness must ensure that time(), microtime(), and getPlayoutStartTime() use the same clock or synchronized clocks. Any potential error due to the clocks not being perfectly synchronized must be accounted for in the value of maxErrorMicroseconds reported by the Test Harness.

NOTE 3: A Test Case may use this to synchronise the availability of DASH segments with the broadcast timeline. To do that:

1. The Test Case calls this JavaScript function, then when the callback is called it passes the timestamp to a first PHP script via XmlHttpRequest, and that first PHP script saves the timestamp in the PHP session.

2. Then the Test Case plays the DASH video, with the MPD URL pointing to a second PHP script. The second PHP script can calculate the segment availability start and end times based on the timestamp in the PHP session and the Test Case's chosen constraints on segment availability, and can include those values in the returned MPD data.

3. The MPD data can contain appropriate timing elements to ensure the DASH player requests the time from the Test Harness via a HTTP call to a third PHP script, which can use time() or microtime() to get the current time. This ensures that the DASH availability start/end times are compared against the correct clock.

4. If necessary, the segment URL in the MPD can point to a fourth PHP script. This PHP script can calculate the segment availability start and end times based on the timestamp in the PHP session, the position of the segment in the DASH video, and the Test Case's chosen constraints on segment availability. This PHP script can then check the current time (obtained using microtime()) against those segment availability times, to determine if the segment should be available or not.

Test implementers should not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.8.6 JS-Function analyzeCssWcPerformance()

Causes the Test Harness to make a number of requests to the specified CSS-WC server and check that the terminal responds to each request within a defined time limit.

```
void HbbTVTestAPI.analyzeCssWcPerformance(
            stepId : integer,
            comment : string,
            cssWcUrl : string,
            numberOfClients : integer,
            numberOfMessages : integer,
            transmitDurationMillis : integer,
            maxDroppedRequests : integer,
            maxResponseTimeMillis : integer,
            callback : function or null,
            callbackObject : object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step

cssWcUrl: URL to the CSS-WC server to be tested. This must be a udp:// URL as used in HbbTV.

numberOfClients: Number of distinct clients to create. Each IP address and port combination is considered a single distinct client. Must be positive and nonzero.

numberOfMessages: Total number of messages to send. Must be positive and nonzero.

transmitDurationMillis: The duration of the "transmit window" when CSS-WC requests are being sent by the Test Harness. Note that the responses to the last few requests will be received after the end of the transmit window.

maxDroppedRequests: The maximum number of CSS-WC requests that can be "dropped" (defined below) if the analysis step is to pass.

maxResponseTimeMillis: If the time taken to receive a CSS-WC response is greater than this value, the analysis step fails. This is measured from the time the CSS-WC request is sent on the network until the CSS-WC response is received on the network. In the case of a 2 part response where both parts are received in the correct order, the arrival time of the second part of the response is measured.

callback/**callbackObject**: a callback function to invoke when the analysis is complete (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis causes the test harness to fail the complete test, independent on the test result reported back by the reportStepResult function. If the analysis step fails, then the callback shall not be called.

The test harness shall send **numberOfMessages** CSS-WC requests to the specified CSS-WC server.

If **numberOfMessages** is 1, then the test harness shall immediately send a single CSS-WC request and **transmitDurationMillis** is ignored; in this case the transmit window has a duration of zero.

Otherwise, the test harness shall send a CSS-WC request immediately, and shall send new CSS-WC requests are every (**transmitDurationMillis** / (**numberOfMessages** - 1)) milliseconds. The first **numberOfClients** requests

are all sent by distinct clients. After that, they repeat, with the client than sent the Nth message also sending the (N + **numberOfClients**)th message.

All CSS-WC messages sent by the server shall use a 64-bit random number for the originate_timevalue field. This shall be generated by a good-quality random number generator. The test harness shall ensure that all the originate_timevalue values used during a single call to analyzeCssWcPerformance() are unique.

After the end of the transmit window, the Test Harness shall continue to listen for responses for (2 * **maxResponseTimeMillis**) milliseconds.

The Test Harness shall consider the analysis step to have failed if any of the following conditions occur:

- Any UDP packet, received for the IP address and port of any of the CSS-WC clients created by this call, does not match the syntax defined for CSS-WC responses, or does not have a message_type of 1, 2, or 3.
- Any CSS-WC response has an originate_timevalue field that does not match a CSS-WC request sent by that client. This includes the case where the response is received on the wrong IP address or port.
- More than two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request sent by that client.
- Two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request sent by that client, and the first one received by the Harness is not message_type 2 (“response that will be followed by a follow-up response”).
- Two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request by that client, and the second one received by the Harness is not message_type 3 (“follow-up response”).
- If the number of CSS-WC requests that are "dropped" is greater than **maxDroppedRequests**. A request is considered to have been "dropped" if the Test Harness does not receive a CSS-WC response with an ID matching that request, or if the Test Harness receives a response with message_type 2 but does not receive the corresponding follow-up response with message type 3, or if the Test Harness receives a response with message_type 3 but does not receive the corresponding response with message type 2.
- If the "response time" for any CSS-WC request exceeds **maxResponseTimeMillis** milliseconds. Requests that are dropped are not counted. The “response time” for a CSS-WC request is measured from the time the CSS-WC request is sent on the network until the corresponding CSS-WC response is received on the network. In the case of a 2 part response where both parts are received in the correct order, the arrival time of the second part of the response is measured.

The Test Harness shall consider the analysis step to have passed if none of the above conditions occur.

The entry/entries in the Test Report XML for this step shall include the number of dropped requests, and the maximum CSS-WC response time that was observed during the analysis.

Test harnesses are only required to support the following values for the parameters:

numberOfClients = 5

numberOfMessages = 25

transmitDurationMillis = 1000

maxDroppedRequests = 0

maxResponseTimeMillis = 200

Support for other values for these parameters is optional in a Test Harness. (It may be helpful when debugging test failures, or for future tests).

7.8.7 JS-Function makeCssWcRequest()

Causes the Test Harness to send a single Wall Clock Synchronization request message to the specified CSS-WC server and retrieves the information from the correspondent response messages body.

```
void HbbTVTestAPI.makeCssWcRequest(
            stepId : integer,
            comment : string,
            cssWcUrl : string,
            requestMessageConfig : constant,
            callback : function or null,
            callbackObject : object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step

cssWcUrl: URL to the CSS-WC server to be tested. This must be a udp:// URL as used in HbbTV.

requestMessageConfig: a configuration value which defines the content of CSS-WC request message sent by the API call. The options are defined as integer constants on the test API object. The options are:

CSS_WC_REQUEST_DEFAULT - Normal request message, use the current system time.

CSS_WC_REQUEST_LARGE_ORIG_NANOS - Same as CSS_WC_REQUEST_DEFAULT, except originate_timevalue_nanos sub-field is 0xFF FF FF FF,

callback/**callbackObject**: a callback function to invoke when the analysis is complete (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis causes the test harness to fail the complete test, independent on the test result reported back by the reportStepResult function. If the analysis step fails, then the callback shall not be called.

The callback function will be called with the following parameters:

```
callback(callbackObject,
       results: Results)
```

Where the results object has the following two sub-objects defined:

```
requests : array
responses : array
```

requests: a Javascript array contains list of the information of each CSS-WC request message sent under this API call

responses: a Javascript array contains list of the information of each CSS-WC request message received under this API call

Each item in these arrays is an JSON-compatible value with following fields:

```
version : int
messageType : int
precision : int
reserved : int
maxFreqError : int
originateTimevalue : {
```

```
        secs : int
        nanos : int}
    receiveTimevalue : {
        secs : int
        nanos : int}
    transmitTimevalue : {
        secs : int
        nanos : int}
    harnessTimestamp : {
        secs : int
        nanos : int}
```

version, messageType, precision, reserved, maxFreqError, originateTimevalue, receiveTimevalue, transmitTimevalue: The corresponding field from CSS-WC Synchronization request/response message, as defined in TS 103 286-2 V1.2.1, section 8.3, "Wall Clock Protocol"

harnessTimestamp: For a CSS-WC request message, it is the timestamp when the Test Harness sent the message. For a CSS-WC response message, it is the timestamp when the Test Harness received the message. This timestamp is in "Unix time" format – i.e. it is the number of seconds and nanoseconds since 1 Jan 1970 00:00 UTC, not counting leap seconds.

The test harness shall immediately send a CSS-WC request to the specified CSS-WC server. The information of the CSS-WC request shall be available in the `requests` object. After the request is sent, the test harness shall listen on the same IP address and port for responses for 500 milliseconds. The information from the CSS-WC responses received by the test harness shall be available in the `responses` object

All CSS-WC request messages sent by the the test harness shall use the Test Harness's current system time for the originate_timevalue field, unless a different behaviour is speicified by `requestMessageConfig` field. The time value used here is in "Unix time" format – i.e. it is the number of seconds and nanoseconds since 1 Jan 1970 00:00 UTC, not counting leap seconds.

The Test Harness shall consider the analysis step to have failed if any of the following conditions occur:

- Any UDP packet, received for the IP address and port used to send any of the CSS-WC requests sent by this call, does not match the syntax and length defined for CSS-WC responses, or does not have a message_type of 1, 2, or 3.
- Any CSS-WC response has an originate_timevalue field that does not match a CSS-WC request sent by this function.
- For one of the CSS-WC requests sent by this call, there are no CSS-WC responses that have an originate_timevalue field that match.
- More than two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request sent by this call.
- Only one CSS-WC response has an originate_timevalue field that matches the CSS-WC request sent by this call, and it is not message_type 1 ("response that will not be followed by a follow-up response").
- Two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request sent by this call, and the first one received by the Harness is not message_type 2 ("response that will be followed by a follow-up response").
- Two CSS-WC responses have an originate_timevalue field that match the same CSS-WC request by this call, and the second one received by the Harness is not message_type 3 ("follow-up response").

The Test Harness shall consider the analysis step to have passed if none of the above conditions occur.

Example of the result object:

```
{
    "requests": [
        {
```

```
            "version": 0,
            "messageType": 0,
            "precision": 0,
            "reserved": 0,
            "maxFreqError": 0,
            "originateTimevalue":{
                "secs": 10,
                "nanos": 10
            },
            "receiveTimevalue":{
                "secs": 0,
                "nanos": 0
            },
            "transmitTimevalue":{
                "secs": 0,
                "nanos": 0
            },
            "harnessTimestamp":{
                "secs": 10,
                "nanos": 10
            }
        }
    ],
    "responses": [
        {
            "version": 0,
            "messageType": 1,
            "precision": 9,
            "reserved": 0,
            "maxFreqError": 256,
            "originateTimevalue":{
                "secs": 10,
                "nanos": 10
            },
            "receiveTimevalue":{
                "secs": 10,
                "nanos": 2000
            },
            "transmitTimevalue":{
                "secs": 11,
                "nanos": 0
            },
            "harnessTimestamp":{
                "secs": 11,
                "nanos": 5000
            }
        }
    ]
}
```


7.8.8 JS-Function createAnalyzeAvSync()

This allows checking the synchronization between two or three of: video, second video, subtitles, and audio. It is used with video and subtitles streams containing flashes, and audio streams containing tone bursts, as described in section 5.2.1.13.

The checkXxx functions modify the object they are called on, then return the same object. This allows chained calls, e.g.:

```
testApi.createAnalyzeAvSync(3, "check").checkLight1AgainstAudio(50, 35).analyze(callback)
```

For each AnalyzeAvSync object returned by a call to createAnalyzeAvSync(), test cases must call at least one of the checkXxx functions before calling analyze(). Test cases must not call a checkXxx function to request a check between a particular pair of sensors, if the test has already requested a check between the same pair of sensors. For the purposes of the previous sentence, sensor order does not matter, e.g. "light 1 and audio" is the same pair as "audio and light 1".". (E.g. you can call checkLight1AgainstAudio() and checkLight2AgainstAudio(), but you cannot call checkLight1AgainstAudio() twice).

Throughout this API:

- "Off state" means light sensor area black / audio off.

- "On state" means light sensor area white / audio tone playing.
- "Off transition" means the transition from “On state” to “Off state”.
- "On transition" means the transition from “Off state” to “On state”.

All time parameters to functions are in milliseconds.

```
AnalyzeAvSync HbbTVTestAPI.createAnalyzeAvSync(
            step_number : unsigned integer,
            comment : string,
            max_millis_between_flashes : unsigned integer or null or undefined)
```

Creates and returns a new AnalyzeAvSync object. Note that the AnalyzeAvSync constructor is not exposed to HbbTV test cases.

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

comment: a comment from the test developer describing the purpose of the step.

max_millis_between_flashes: The maximum time allowed, in milliseconds between **captured** flashes or beeps, typically set to about 500 milliseconds more than the longest gap between flashes and beeps **in the media being played out** for this analysis. This is intended to detect when a flash or beep is missed completely. A number between 500 and 12000 inclusive, or null or undefined, or this parameter may be omitted. If null or undefined or omitted, then the default value of 3500 is used.

7.8.8.1 Checking flashes and tones

```
AnalyzeAvSync AnalyzeAvSync.checkLight1AgainstAudio(
            maxLight1ThenAudioDelay: unsigned integer,
            maxAudioThenLight1Delay: unsigned integer)
```

Requests that, when analyze() is called, then light sensor 1 is checked against the audio (See section 5.2.1.12 for details of the light sensor positions). Modifies the object it is called on, then return the same object. Once this function has been called on an AnalyzeAvSync object, test cases must not call this function again on the same object.

ARGS: **maxLight1ThenAudioDelay**: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from a flash being detected on light sensor 1 to the audio tone being detected. Note that if the audio tone is detected first, then this parameter is ignored. Must be >= 0.

maxAudioThenLight1Delay: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from the audio tone being detected to a flash being detected on light sensor 1. Note that if the flash on light sensor 1 is detected first, then this parameter is ignored. Must be >= 0.

```
AnalyzeAvSync AnalyzeAvSync.checkLight2AgainstAudio(
            maxLight2ThenAudioDelay: unsigned integer,
            maxAudioThenLight2Delay: unsigned integer)
```

Requests that, when analyze() is called, then light sensor 2 is checked against the audio (See section 5.2.1.12 for details of the light sensor positions). Modifies the object it is called on, then return the same object. Once this function has been called on an AnalyzeAvSync object, test cases must not call this function again on the same object.

ARGS: **maxLight2ThenAudioDelay**: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from a flash being detected on light sensor 2 to the audio tone being detected. Note that if the audio tone is detected first, then this parameter is ignored. Must be >= 0.

maxAudioThenLight2Delay: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from the audio tone being detected to a flash being detected on light sensor 2. Note that if the flash on light sensor 2 is detected first, then this parameter is ignored. Must be >= 0.

```
AnalyzeAvSync AnalyzeAvSync.checkLight1AgainstLight2(
            maxLight1ThenLight2Delay: unsigned integer,
            maxLight2ThenLight1Delay: unsigned integer)
```

Requests that, when analyze() is called, then light sensor 1 is checked against light sensor 2 (See section 5.2.1.12 for details of the light sensor positions). Modifies the object it is called on, then return the same object. Once this function has been called on an AnalyzeAvSync object, test cases must not call this function again on the same object.

ARGS: **maxLight1ThenLight2Delay**: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from a flash being detected on light sensor 1 to a flash being detected on light sensor 2. Note that if the a flash on light sensor 2 is detected first, then this parameter is ignored. Must be >= 0.

maxLight2ThenLight1Delay: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from a flash being detected on light sensor 2 to a flash being detected on light sensor 1. Note that if the a flash on light sensor 1 is detected first, then this parameter is ignored. Must be >= 0.

7.8.8.2 Checking on and off transitions

```
AnalyzeAvSync AnalyzeAvSync.checkOnOff(
        sensorA,
        sensorB,
        maxOnBThenA : unsigned int,
        maxOffBThenA : unsigned int,
        maxOnAThenB : unsigned int,
        maxOffAThenB : unsigned int,
        minAOn : unsigned int,
        minAOff : unsigned int,
        maxAOn : unsigned int,
        maxAOff : unsigned int);
```

Requests that, when analyze() is called, then the specified sensor A is checked against the specified sensor B. Modifies the object it is called on, then return the same object.

ARGS: **sensorA**: First sensor to check. Must be one of the AVSYNC_SENSOR_xxx constants as defined in section 7.8.9.

sensorB: Second sensor to check. Must be one of the AVSYNC_SENSOR_xxx constants as defined in section 7.8.9. Must be different from **sensorA**.

maxOnBThenA: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from an “on” transition (black to white, or silence to tone) being detected on sensor B to an “on” transition being detected on sensor A. Note that if the transition on sensor A is detected first, then this parameter is ignored and **maxOnAThenB** is used instead. Must be >= 0 and <=15000.

maxOffBThenA: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from an “off” transition (white to black, or tone to silence) being detected on sensor B to an “off” transition being detected on sensor A. Note that if the transition on sensor A is detected first, then this parameter is ignored and **maxOffAThenB** is used instead. Must be >= 0 and <=15000.

maxOnAThenB: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from an “on” transition (black to white, or silence to tone) being detected on sensor A to an “on” transition being detected on sensor B. Note that if the transition on sensor B is detected first, then this parameter is ignored and **maxOnBThenA** is used instead. Must be >= 0 and <=15000.

maxOffAThenB: maximum allowed synchronisation error, in milliseconds. This is the maximum delay from an “off” transition (white to black, or tone to silence) being detected on sensor A to an “off” transition being detected on sensor B. Note that if the transition on sensor B is detected first, then this parameter is ignored and **maxOffBThenA** is used instead. Must be >= 0 and <=15000.

minAOn: minimum allowed duration of a continuous “on” state (white or tone) on sensor A, in milliseconds. If a shorter “on” pulse is detected, then either media playback has failed or the sensor is not working properly, so the test case will fail. Must be >=20 and <14799.

minAOff: minimum allowed duration of a continuous “off” state (black or silence) on sensor A, in milliseconds. If a shorter “off” pulse is detected, then either media playback has failed or the sensor is not working properly, so the test case will fail. Must be >=20 and <14799.

maxAOn: maximum allowed duration of a continuous “on” state (white or tone) on sensor A, in milliseconds. If a longer “on” pulse is detected, then either media playback has failed or the sensor is not working properly, so the test case will fail. Must be >20 and <=14799.

maxAOff: maximum allowed duration of a continuous “off” state (black or silence) on sensor A, in milliseconds. If a longer “off” pulse is detected, then either media playback has failed or the sensor is not working properly, so the test case will fail. Must be >20 and <=14799.

The parameters must satisfy all the following rules:

- **maxAOn > minAOn**
- **maxAOff > minAOff**
- 20 <= **minAOn – (maxOnAThenB + maxOffBThenA)**
- 20 <= **minAOff – (maxOffAThenB + maxOnBThenA)**
- 14799 >= **maxAOn + maxOnBThenA + maxOffAThenB**
- 14799 >= **maxAOff + maxOffBThenA + maxOnAThenB**

7.8.8.3 Starting the analysis

```
void AnalyzeAvSync.analyze(
            callback : function or null,
            callbackObject : object)
```

Performs the requested analysis. Test cases must not use the AnalyzeAvSync object after calling analyze() on it.

ARGS: **callback**/**callbackObject**: a callback function to invoke when the observation is complete and, if the Test Harness chooses to do the analysis immediately, the analysis is also complete analysis was made successful (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis causes the test harness to fail the complete test, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording.

Before calling analyze() the test case must call at least one of the ‘check...()’ functions.

NOTE: "Check light sensor 1 against audio, and light sensor 2 against audio, with a 10ms tolerance" is NOT the same as "check all 3 with a 10ms tolerance", since if light sensor 1 flashes 8ms earlier than the audio tone, and light sensor 2 flashes 8ms later than the audio tone, then there is 16ms between light sensor 1 and light sensor 2, so the first example passes and the second example fails.

For a period of 15 seconds, starting within 5 seconds of the time this API is called, this monitors the selected light and audio sensors. The harness performs all the requested checks on the captured data.

Test implementers must not call this function when network connection is configured to be down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

7.8.8.4 Analysis – flashes and tones

This section applies if the test case called any of the checkXxx APIs to check flashes and tones. This analysis is carried out on the pair of sensors specified in the call. If the test cast requested multiple flashes and tones checks with different pairs of sensors, then this analysis is repeated for each requested pair of sensors.

For tests using this analysis method, the media should be mostly black/silence, with short flashes/tones.

The harness shall check that all the detected flashes and tones are synchronised within the specified tolerance.

In this section, the time of a flash or tone is the time of the mid-point of the flash or tone. (There is no requirement for flashes or tones to have the same length).

The decision about whether this analysis passes or fails is made as follows:

- If there is a flash or tone that does not have a corresponding flash or tone with a centre within the specified tolerance, and the centre of that flash or tone is more than the specified tolerance plus 100ms from both the start and end of the 15 second observation period, then the test fails. (Note: The special handling of the start and end of the observation period accounts for cases where the synchronization is not perfect but is within the specified tolerance, and the Test Harness starts monitoring just after a flash and just before the corresponding tone, so the tone is detected but the flash is missed).

- If there is no flash or tone detected for any period of **max_millis_between_flashes** millisecondsseconds within the 15 seconds observation period, then that also causes the test to fail. (Note: that means some flashes or tones are not being detected at all, either due to a failure of the DUT - perhaps it's not playing the media at all - or a failure in the test harness – perhaps the light sensor fell off the screen).

- If two flashes or two tones are detected from the same source within a 400ms window, that also causes the test to fail. (Note: that means spurious flashes have been detected, either due to a failure of the DUT or a failure in the test harness).

- If none of the above conditions apply, then this step passes.

NOTE: The algorithm above is designed to work with media files containing the repeating pattern of flashes and tones described in section 5.2.1.13. However, other patterns of flashes and tones can also be used.

The entry/entries in the Test Report XML for this step shall include the maximum measured difference in milliseconds between the flash/tone.

NOTE: The format of the entry in the Test Report XML is not defined. It is for visual reference only. Possible locations for the entry are the testStepComment or testStepData tags.

NOTE: This section checks the mid-point of the flash. Experimental results show that typical TV "picture improvement" processing can affect the detection of the flash start and end times. Experiments suggest that the start time of the flash is affected more than the mid-point is. Particularly when testing audio/video sync, choosing the mid-point makes the tests more robust.

NOTE:

7.8.8.5 Analysis – on and off transitions

This section applies if the test case called the **checkOnOff** API. This analysis is carried out on the pair of sensors specified in the call. If the test cast requested multiple "on and off transitions" checks with different pairs of sensors, then this analysis is repeated for each requested pair of sensors.

For tests using this analysis method, the media should be a mix of black/silence and white/tones. The time of the transitions are analysed. The transitions are black to white, white to black, silence to tone, and tone to silence.

NOTE: This checks that the transitions found by each sensor can be matched to transitions in the other sensor, within the specified limits. At the start or end, unmatched transition(s) are allowed so long as they could plausibly match a transition that was outside the time period that was analyzed.

The analysis period is 15 seconds long. In the following description, the "trimmed analysis period" refers to the analysis period except for the first 100 milliseconds and the last 100 milliseconds; i.e. the "trimmed analysis period" is 14.8 seconds long. (This accounts for the fact that detecting a transition at the very start or end of the analysis period may not be possible, so we don't try to do that).

The decision about whether this analysis passes or fails is made as follows:

- First, interpret and check the sensor A data:
    - Examine the sensor A data to determine the transition times that are within the trimmed analysis period. (I.e. any transitions within 100ms of the start or end of the 15 second analysis period are completely ignored by this entire algorithm).

    - If no transitions were detected, the analysis has failed. (This is because we have 14800 milliseconds without a transition, and both **maxAOn** and **maxAOff** are less than that).
    - For each consecutive pair of transitions, check that the time between transitions is within the limits set by **minAOn**, **minAOff**, **maxAOn**, and **maxAOff** as appropriate.
    - Calculate the time between the start of the trimmed analysis period, and the first transition. Check this does not exceed **maxAOn** or **maxAOff** as appropriate.
    - Calculate the time between the last transition and the end of the trimmed analysis period. Check this does not exceed **maxAOn** or **maxAOff** as appropriate.
- Calculate the limits for sensor B as follows:
    - **minBOn = minAOn – (maxOnAThenB + maxOffBThenA)**
    - **minBOff = minAOff – (maxOffAThenB + maxOnBThenA)**
    - **maxBOn = maxAOn + maxOnBThenA + maxOffAThenB**
    - **maxBOff = maxAOff + maxOffBThenA + maxOnAThenB**
- Now, interpret and check the sensor B data, as was done for sensor A. Use the computed limits for sensor B instead of the passed-in limits for sensor A.
- Compare the interpreted data from the two sensors. The analysis will pass if there are correction factors, CFAS, CFBS, which are non-negative integers that satisfy all the following checks. Otherwise the analysis fails. A set of correction factors is valid if all the following checks hold:
    - Check CFAS = 0 or CFBS = 0 or both.
    - Check CFAS < number of transitions on sensor A.
    - Check CFBS < number of transitions on sensor B.
    - Let CTA be the list of transitions on sensor A, except with the first CFAS transitions excluded.
    - Let CTB be the list of transitions on sensor B, except with the first CFBS transitions excluded.
    - Check the first transition in CTA is the same type as the first transition in CTB – either both "on" or both "off".
    - If CFAS is nonzero, then transitions were not included at the start of CTA. For each of those transitions, check that the time difference between the beginning of the trimmed analysis period and the excluded transition is less than or equal to the appropriate limit – either **maxOnBThenA** or **maxOffBThenA**. (I.e. this checks that the transition in sensor B that we didn't detect, could plausibly have occurred at or before the beginning of the trimmed analysis period).
    - If CFBS is nonzero, then transitions were not included at the start of CTB. For each of those transitions, check that the time difference between the beginning of the trimmed analysis period and the excluded transition is less than or equal to the appropriate limit – either **maxOnAThenB** or **maxOffAThenB**. (I.e. this checks that the transition in sensor A that we didn't detect, could plausibly have occurred at or before the beginning of the trimmed analysis period).
    - Pair up each transition in CTA with the corresponding transition in CTB, starting at the start of both lists. This is a straightforward "dumb" pairing of the Nth element in CTA with the Nth element in CTB. If one of the lists is longer, ignore the excess transitions for now. For every pair of transitions, check the difference between the transition times is less than or equal to the appropriate limit. The "appropriate limit" is the applicable one of **maxOnAThenB**, **maxOnBThenA**, **maxOffAThenB** and **maxOffBThenA**.
    - If, in the previous step, excess transitions were ignored from the end of CTA, check that the time difference between each ignored transition and the end of the trimmed analysis period is less than or equal to the appropriate limit – either **maxOnAThenB** or **maxOffAThenB**. (I.e. this checks that the transition in sensor B that we didn't detect, could plausibly have occurred at or after the end of the trimmed analysis period).
    - If, in the previous step, excess transitions were ignored from the end of CTB, check that the time difference between each ignored transition and the end of the trimmed analysis period is less than or equal to the appropriate limit – either **maxOnBThenA** or **maxOffBThenA**. (I.e. this checks that the transition in sensor A that we didn't detect, could plausibly have occurred at or after the end of the trimmed analysis period).

Note that the harness is not required to implement exactly the algorithm above, it may use any implementation that provides exactly the same results as the algorithm above.

7.8.9 JS-Constants AVSYNC_SENSOR_xxx

The following constants shall exist:

- HbbTVTestAPI.AVSYNC_SENSOR_AUDIO
- HbbTVTestAPI.AVSYNC_SENSOR_LIGHT_1
- HbbTVTestAPI.AVSYNC_SENSOR_LIGHT_2

Their values, and type are harness-dependant.

These shall be defined both on the HbbTVTestAPI constructor, and on all instances of that class.

See section 7.8.8 for how these constants are used.

See section 5.2.1.12 for details of the light sensor positions.

7.8.10 Subtitle synchronisation

A subtitle synchronisation accuracy has no fixed time requirement. For testing purpouses it was decided that subtitles may appear/disappear 500 ms ahead an expected time and 900 ms after the expected time.

7.9 APIs for network testing

7.9.1 JS-Function analyzeNetworkLog()

For certain tests, analyzing network log is necessary in order to check whether expected network issues are occurring. This function allows test developers to invoke recording of network traffic and to specify type of analysis that should be performed on data recorded by network analysis tool (described in section 5.2.1.15).

```
void HbbTVTestAPI.analyzeNetworkLog (
                stepId : integer,
                expectedBehavior : integer,
                URL : String or array,
                timeout : integer,
                comment : String,
                check : String,
                minLogCount : integer,
                maxLogCount : integer,
                analysisStartedCB : function or null,
                callback : function or null,
                callbackObject : object);
```

ARGS: **stepId**: the step number that has been performed (same as stepId in reportStepResult).

expectedBehaviour: Type of behaviour that is expected to see when analyzing network log (integer that represent type of behaviour, see the table below):

Expected behaviour (name of constant)	Integer value	Description
RESOLVED_IP_UNREACHABLE	1	Attempt to access URL results in network log entry showing request to an IP address that is not reachable
DNS_FAIL	2	Attempt to access URL results in network log entry showing a DNS request for the hostname from the URL, and a corresponding NXDOMAIN DNS response
CONNECTION_REFUSED	3	Attempt to access URL results in network log entry showing a refused TCP/IP connection to the specified URL (IP and port).

DNS_LOOKUP	4	The harness logs and analyzes DNS requests to its DNS server. It ignores requests that include wildcards. It counts the number of requests made against each specified name from the URL parameter, considering wildcards in the names, and then fails the test if the number of requests to any name is not between its specified min and max. It also sums the total count of all such requests and fails the test if the total is not between minLogCount and maxLogCount. It does not consider whether a response was received from the wider DNS system or the nature of that response. It considers both IPv4 (QTYPE: A) requests and IPv6 (QTYPE: AAAA) requests.

URL – For expectedBehaviour 1 to 3, represents the URL that should be tracked in network log

NOTE: In case of low-level TCP errors, such as a refused TCP/IP connection, the full URL will not be available in the recorded traffic log. Host part of the URL will be resolved to an IP and a port number. IP is determined by parsing the DNS request, which occurs when the DUT resolves host's name to its local network IP.

For expectedBehaviour 4, each element of the array is, itself, an array of length three, representing a DNS name (or range of names) of interest and how many requests are expected. Each has three elements:

i. a String, representing the name. It may start **or** end (but not both) with an asterisk character, which acts as a wildcard. For example, *.example.com (which would match both www.example.com or a.very.densely.nested.example.sub.domain.example.com) or hbbtv1.* (which would match both hbbtv1.example.com and hbbtv1.www.example.com).

ii. a non-negative integer, representing the minimum occurrences of DNS lookups to the domain name(s) specified by the string element, with 0 indicating no minimum. This value shall not be greater than the value represented by the third array element (the maximum).

iii. a non-negative integer or null, representing the maximum occurrences of DNS lookups to the domain name(s) specified by the string element, with null indicating no maximum.

The min shall not be greater than the max.

So, for example:

```
URL = [
  ["*.example.com", 5, null],
  ["hbbtv1.*", 2, 6],
  ["hbbtv1.example.com", 2, 2],
  ["hbbtv2.example.com", 0, 0]
];
```

means five or more lookups for names ending "example.com"; 2 to 6 for names starting "hbbtv1.", exactly 2 requests for name "hbbtv1.example.com"; and no lookups for name "hbbtv2.example.com"

timeout – Time in milliseconds to log network traffic. Logging of network traffic starts sometime after analyzeNetworkLog() was called but before the callback function analysisStartedCB() is invoked, and runs for this length of time.

comment: a comment from the test developer describing what the analysis actually does (same as reportStepResult)

check: a textual description detailing which checks to perform on the network log. This is the only criteria that shall be used for the assessment of this analysis call. This allows the network log to be recorded (duration specified by timeout parameter) and then processed later on. An empty or null string shall cause the test to fail.

minLogCount: Minimum number of URL occurrences related to expected behaviour found in network log in timeout interval. If number of occurrences is below this number, test automatically fails.

maxLogCount: Maximum number of URL occurrences related to expected behaviour found in network log in timeout interval. If number of occurrences is above this number, test automatically fails.

analysisStartedCB: a callback function which is invoked once that network log capturing/monitoring has been started. Function does not have any parameters. Function is invoked once that harness starts recording/analysis of the network log. This callback function is suitable place in which test implementer should initiate process which causes network activity that needs to be tracked (for example, starts DASH playback).

callback/callbackObject: a callback function to invoke when the logging required for the analysis has finished (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail, independent on the test result reported back by the reportStepResult function. This is due to the fact that the analysis can also be performed off-line on a taken recording network traffic log.

Just before invocation of callback function ***analysisStartedCB***, network log recording/analysis was started, and it should end after ***timeout*** milliseconds. After specified timeout, test harness should stop recording/analysis of the network log. Once analysis is completed, ***callback*** function is invoked.

Invocation of the callback does not mean that analysis was successful (it may be postponed for later stage). In case when specified network problem occurs fewer times than specified by ***minLogCount*** or more times than specified by ***maxLogCount***, test harness should automatically fail the test.

7.10 APIs for Targeted Advertising

7.10.1 Introduction

This section defines a test API that can be used to measure a switch from one AV stream to another. I.e. changing from a broadcast audio and video to a broadband audio and video, or vice-versa. This API requires the streams to have special video and audio contents, which are also defined in this section. Note that the API does not impose any restrictions on stream codecs, frame rates, resolutions* etc, the stream requirements only apply to the video shown on the TV screen and the sound emitted from the TV's audio out (typically it's headphone jack).

(* The resolution is required to be SD or better, just to have enough pixels for the QR codes. Sub-SD resolutions are not supported).

7.10.2 Overview

Each video contains QR codes giving the timestamp (i.e. the playback position in milliseconds). A camera is used to capture this, and the video is analysed to detect which timestamps were played. Due to the nature of PC

webcam capture, including the low frame rate (50fps == 20ms per frame), this cannot reliably detect exactly when playback stopped and started.

Each video also contains two rectangles, one white and one black. One video has the rectangle 1 being white, the other has the rectangle 2 being white. A light sensor is positioned over each rectangle, being careful that the light sensor cable does not cover the QR codes. The light sensors are used to determine when (according to UTC time) the first video ceases to be displayed (rectangle goes black) and when the second video starts to be displayed (other rectangle goes white).

Each audio contains a continuous tone, with a changing volume that encodes the audio timestamps. The two audios use different frequencies, so the switch from one audio to the other can be detected by just looking at the audio frequency. The TV audio is both captured by the PC's audio in, which provides a good recording that is not synchronized to anything else, and is to some extent measured by the same device that is measuring the light sensors. The precise timing information from the device is used to synchronize the recording with the UTC time, and that recording will then be analysed to determine when the first audio stopped and when the second started, both as UTC times and as timeline times.

This gives measurement of:

Xv_timeline	The time the first video stopped, measured on its own timeline, in milliseconds.
Xv_UTC	The time the first video stopped, measured on an independent clock (set to UTC).
Sv_timeline	The time the second video started, measured on its own timeline, in milliseconds.
Sv_UTC	The time the second video started, measured on an independent clock (set to UTC).
Xa_timeline	The time the first audio stopped, measured on its own timeline, in milliseconds.
Xa_UTC	The time the first audio stopped, measured on an independent clock (set to UTC).
Sa_timeline	The time the second audio started, measured on its own timeline, in milliseconds.
Sa_UTC	The time the second audio started, measured on an independent clock (set to UTC).

The measured values listed above can be used to verify the switch happened correctly, given the desired switch time, the expected playback media timeline time, and the values of A, Dmin, Dmax. Here "A", "Dmin" and "Dmax" have the same meaning as the values in the HbbTV TA specification called either A1, D1min and D1max or A2, D2min and D2max, depending on whether the switch being analysed is broadcast-to-broadband or broadband-to-broadcast.

7.10.3 Detailed stream specification - audio

The audio contains a single-frequency constant-frequency tone, switching between two volume levels, with the "high volume" level being approximately double the volume of the "low volume" level.

The frequency indicates whether the stream is broadcast or broadband. The frequencies are:

- Broadcast: 2500 Hz
- Broadband: 1500 Hz

The two volume levels are used to encode a sequence of binary data:

- 80ms high volume
- 20ms low volume
- 22 bits of data, with each bit encoded as:
    - For a "0" bit: 20ms low volume followed by 20ms high volume
    - For a "1" bit: 20ms high volume followed by 20ms low volume

  (This is a modified "Manchester encoding").
- 20ms low volume
- This sequence then repeats. Note this repeating sequence is exactly 1 second long.

The data shall be encoded with an error correcting code (FEC) consisting of a Hamming(31,26) code truncated to 21 bits, followed by an Even parity bit. This contains 16 bits of data, and allows 1-bit errors to be corrected and 2-bit errors to be detected.

Details of the FEC:

- For details of the Hamming code see https://en.wikipedia.org/w/index.php?title=Hamming_code&oldid=940024473 , the table under "This general rule can be shown visually:" defines the first 20 bits, the 21st bit is an obvious extension of that table
- Bits are transmitted in the "bit position" order shown in that table, bit position 1 first
- The even parity bit is transmitted last (in "bit position 22")
- d1 is the MSB of the data, d16 is the LSB

To put that definition of the FEC another way, the bits transmitted are calculated as follows:

Transmission order	**Bit name**	**Calculation** (d1..d16 represent the data bits. d1 is the MSB of the data, d16 is the LSB. "^" is the binary XOR operator. All variables are 1-bit binary)
1 (first)	p1	d1 ^ d2 ^ d4 ^ d5 ^ d7 ^ d9 ^ d11 ^ d12 ^ d14 ^ d16
2	p2	d1 ^ d3 ^ d4 ^ d6 ^ d7 ^ d10 ^ d11 ^ d13 ^ d14
3	d1	d1
4	p4	d2 ^ d3 ^ d4 ^ d8 ^ d9 ^ d10 ^ d11 ^ d15 ^ d16
5	d2	d2
6	d3	d3
7	d4	d4
8	p8	d5 ^ d6 ^ d7 ^ d8 ^ d9 ^ d10 ^ d11
9	d5	d5
10	d6	d6
11	d7	d7
12	d8	d8
13	d9	d9
14	d10	d10
15	d11	d11
16	p16	d12 ^ d13 ^ d14 ^ d15 ^ d16
17	d12	d12
18	d13	d13
19	d14	d14
20	d15	d15

21	d16	d16
22 (last)	pe	p1 ^ p2 ^ d1 ^ p4 ^ d2 ^ d3 ^ d4 ^ p8 ^ d5 ^ d6 ^ d7 ^ d8 ^d9 ^ d10 ^ d11 ^ p16 ^ d12 ^ d13 ^ d14 ^ d15 ^ d16 (Note this calculation uses the parity bits p1, p2, p4, p8, p16 that were calculated earlier in this table).

From the decoded 16 bits of data, the most significant 4 bits are unused and shall be set to zero. The remaining 12 bits are the "embedded timestamp".

The "embedded timestamp" is measured in seconds, and indicates the media timeline time precisely at the start of the 80ms high volume tone that preceded it, modulo 4096.

If an underlying timeline (e.g. PTS or TEMI) wraps, the media timeline time keeps counting up.

Note: For the switchMediaPresentation() API, the Targeted Advertising specification allows an app to specify the switch point using this “keeps counting up” representation of the media timeline, or a representation where the media timeline time goes to 0 at the wrap point. For analyzeAvSwitchPerformance(), we only support the “keeps counting up” representation. This applies to the timestamp encoded in the audio, the timestamp encoded in the video, and the API parameters. If necessary, it is fine to pass different representations of the same timestamp to these functions.

7.10.4 Decoder implementation (informative)

Notes on implementing a decoder, for harness authors:

- First split the audio into the separate streams:
  - Silence indicates no stream was playing, decode the audio before and after the silence separately.
  - Check frequencies. A change from one of the frequencies listed above to the other indicates a new stream, hence a discontinuity in the timestamp signal – decode the two parts separately.
- The signal is designed to contain a volume transition every 20-40ms, except for the header at the start which has up to 80ms between transitions. This allows for the decoder to recover the clock easily.
- Once the decoder has the clock it can group the audio into 20ms groups. It can then decide if each group is high or low volume.
- The 80ms (4 group) run of high volume, with at least 20ms (1 group) of low volume each side, indicates the header.
- Check the rest of the header and trailer. Check that the data bits are high-low or low-high, not high-high or low-low.
- The 22 data bits can then be extracted.
- The FEC (Hamming+Parity) then allows you to extract the 12-bit timestamp with good confidence that you have decoded it correctly.
- Adjacent repeats will have the timestamp increase by 1 (modulo 4096) each time, which can assist further in ensuring you have decoded the audio timestamp correctly
- To measure a start or end timestamp with sub-second accuracy, decode the first whole timestamp signal after or before it, then predict what the truncated timestamp signal would have been, and compare with what you actually recorded. If everything is good the actual signal should match the prediction except that it is truncated at some point.

7.10.5 Detailed stream specification - video

The video contains:

- A solid box at the bottom left of the screen, where Light Sensor 1 is positioned (see the Synchronization part of the specification). This shall be solid black for broadcast content, and solid white for broadband content.

- A solid white box at the bottom right of the screen, where Light Sensor 2 is positioned (see the Synchronization part of the specification). This shall be solid black for broadband content, and solid white for broadcast content.
- Elsewhere on the screen, QR codes indicating whether the video is broadcast or broadband, and the current timeline time.
- Elsewhere on the screen, to aid debugging, human-readable text saying it is broadcast and giving the current timeline time.

The QR codes shall be placed in a grid that is 6 wide and 2 high. There shall not be any visible grid lines. The space between QR codes shall be exactly 6 QR modules in size. The space between a QR code and the edge of the QR area shall be at least 6 QR modules in size. The background of the QR area, including the Quiet Zones and wherever a QR code is not displayed, shall be black. When a QR code is displayed, it shall be displayed in white-on-black (i.e. inverted). QR codes shall be displayed the normal way up, i.e. with the three main alignment marks (the nested squares) at the top left, top right, and bottom left. The QR module size shall be an integer number of pixels, and modules shall be pixel-aligned. The QR modules should be as large as possible. The QR codes must be in the title-safe area of the screen, although the Quiet Zone may extend outside that area. It is expected that the QR grid will be displayed at different sizes for different video resolutions.

All QR codes shall be version 1.

7.10.5 Corner QR Codes

The 4 corner QR codes are to allow the harness to identify the stream and locate the QR code area. They are always displayed. They have the following fixed data contents, consisting of 4 characters in the Alphanumeric encoding mode:

Top Left		**Top Right**
S30A	**For broadcast media**	S30R
S40A	**For broadband media**	S40R
Bottom Left		**Bottom Right**
S30L	**For broadcast media**	S30Z
S40L	**For broadband media**	S40Z

7.10.6 Timestamp QR Codes

Of the other 8 QR codes, they are each drawn for 4 frames then omitted for 4 frames. They contain the timestamp of the first frame they are drawn. After each 8 frame cycle they are drawn again but with an updated timestamp.

The QR codes are on different phases, so one new QR code is drawn every frame and one old QR code is removed each frame. So there will always be 4 QR codes drawn each frame and the other 4 are missing.

The order they are drawn is as follows. These numbers are also the QR code "position ID" which is encoded into the QR code:

Corner	**1**	**2**	**3**	**4**	Corner
Corner	**8**	**7**	**6**	**5**	Corner

Note that the first frame requires special handling – you have to pretend that you drew the preceding 3 frames, and generate 4 QR codes with 3 of them being "in the past" by different amounts.

The data in these QR codes is 9 characters in the Numeric encoding mode:

	Media ID	Position	Timecode
For broadcast media	**3**	1-digit QR code position ID, see above	7-digit time code, see below
For broadband media	**4**		

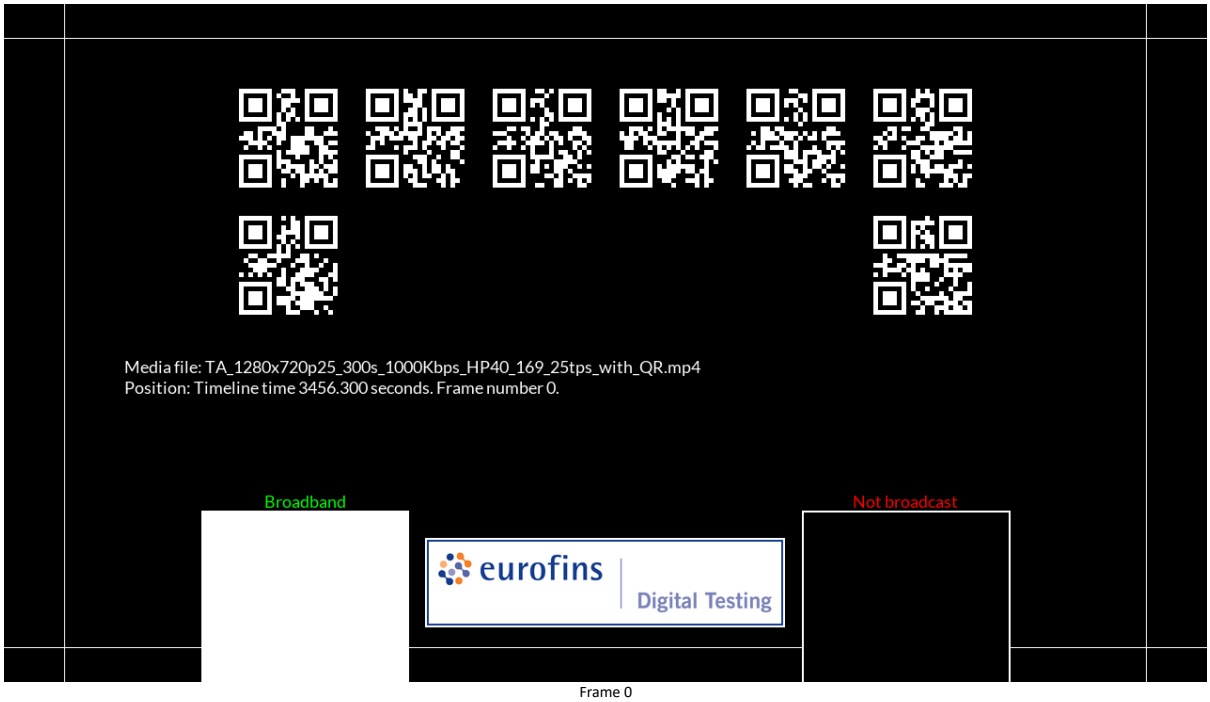
The timecode is a number from `0000000` to `4095999` inclusive. It is always 7 digits long, pad with zeroes on the left if needed. It gives the media timeline time when the frame starts being displayed, in milliseconds, modulo 4096000 milliseconds. (This number is chosen so the video and audio embedded timestamps wrap at the same time).

If an underlying timeline (e.g. PTS or TEMI) wraps, the media timeline time keeps counting up.

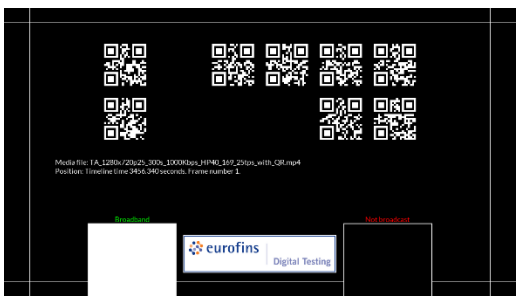
Note: For the switchMediaPresentation() API, the Targeted Advertising specification allows an app to specify the switch point using this "keeps counting up" representation of the media timeline, or a representation where the media timeline time goes to 0 at the wrap point. For analyzeAvSwitchPerformance(), we only support the "keeps counting up" representation. This applies to the timestamp encoded in the audio, the timestamp encoded in the video, and the API parameters. If necessary, it is fine to pass different representations of the same timestamp to these functions.

7.10.7 Example Video (informative)

As an example, here are the first 11 frames of a sequence:



Frame 0



1

6

2

7

3

8

4

9

5

10

© Copyright 2011-2025

7.10.8 JS-Function getCurrentTime()

Gets the POSIX timestamp representing the current UTC time, according to the test harness's clock.

```
void HbbTVTestAPI.getCurrentTime(
        callback : function,
        callbackObject : object);
```

The callback is called as:

```
void callback(
        callbackObject : object,
        seconds : integer,
        microseconds : integer);
```

The timestamp passed to the callback specifies a UNIX timestamp, split into an integer number of seconds and 0 to 999999 microseconds.

This function shall be accurate to within +0/-1000ms. I.e. when the callback is called, the passed timestamp may be up to 1 second earlier than the actual time, but will not be later than the actual time.

NOTE: This function is expected to be accurate to within about +0/-100ms normally, if the terminal has reasonably good performance.

The network connection to the harness must be available when this function is called, otherwise it fails the entire test case automatically.

7.10.9 JS-Constants Switch direction constants

The following constants shall exist:

```
HbbTVTestAPI.SWITCH_BROADCAST_TO_BROADBAND
HbbTVTestAPI.SWITCH_BROADBAND_TO_BROADCAST
```

Their values, and type are harness-dependant.

These shall be defined both on the HbbTVTestAPI constructor, and on all instances of that class.

7.10.10 JS-Function analyzeAvSwitchPerformance()

```
void HbbTVTestAPI.analyzeAvSwitchPerformance(
        stepId : integer,
        direction,
        expected_switch_millis : unsigned integer,
        from_timeline_time_millis : finite number or null,
        to_timeline_time_millis : finite number or null,
        from_fps : unsigned integer,
        to_fps : unsigned integer,
        required_accuracy_millis : unsigned integer or null,
        required_duration_min_millis : unsigned integer or null,
        required_duration_max_millis : unsigned integer or null,
        optional : boolean,
        callback : function or null,
        callbackObject : object,
        to_timeline_time_early_millis: unsigned integer or null or undefined,
        to_timeline_time_late_millis: unsigned integer or null or undefined)
```

This analyses a switch from one AV to another AV. The first AV is referred to as the "from" AV and the second is referred to as the "to" AV

ARGS: **stepId:** the step number for this analysis (same as stepId in reportStepResult).

direction: either HbbTVTestAPI.SWITCH_BROADCAST_TO_BROADBAND for a broadcast to broadband switch, or HbbTVTestAPI.SWITCH_BROADBAND_TO_BROADCAST for a broadband to broadcast switch. This controls which light sensor is used for the "from" video and

which is used for the “to” video. It is also used to check the audio frequencies are correct, and to validate that detected QR codes are from the correct video.

expected_switch_millis: When the switch is expected to happen, measured in milliseconds from when this API is called. Must be accurate to within +/-5 seconds. Must be at least 10 seconds after this API is called, and less than 3 minutes after this API is called. This is not intended to be a precise number that is validated as part of the analysis; it is intended that it may be used by the harness to start and stop video and audio capture at the right time. If it is wrong by more than 5 seconds, then the harness may fail to capture the switch, in which case it will fail the test case.

from_timeline_time_millis: The requested switch time, as a timeline time on the “from” video and audio, or null if the switch is being done based on the “to” timeline time. Note that the timeline time repeats every 4,096,000 milliseconds (4096 seconds), so this is interpreted modulo 4,096,000. As a convenience for test case authors, non-integer values are accepted but may be rounded to the nearest integer. When switching from broadcast to broadband, the rounding up to align to a broadcast video frame boundary needs to be considered by a test application.

to_timeline_time_millis: Where the “to” video and audio should start playing, as a timeline time on the “to” video and audio, or null if this is not to be tested. This has two different meanings: If from_timeline_time_millis is null then this is the switch time, if from_timeline_time_millis is not null then this is the playback start time. Those two values have different tolerances. Note that the timeline time repeats every 4,096,000 milliseconds (4096 seconds), so this is interpreted modulo 4,096,000. As a convenience for test case authors, non-integer values are accepted but may be rounded to the nearest integer.

from_fps: The “from” video frame rate, in frames per second. Used to calculate some tolerances. Values greater than 1000 are not allowed. This does not imply that the API will function correctly with values up to 1000.

to_fps: The “to” video frame rate, in frames per second. Used to calculate some tolerances. Values greater than 1000 are not allowed. This does not imply that the API will function correctly with values up to 1000.

required_accuracy_millis: The required accuracy, in milliseconds. This is “A1” or “A2” as defined in a HbbTV TA performance profile, or null to disable checking of the accuracy (e.g. if the terminal does not support a performance profile). Values greater than 10000 are not allowed. This does not imply that the API will function correctly with values up to 10000; the total switch time (required_accuracy_millis + required_duration_max_millis) should be less than or equal to 1000ms.

required_duration_min_millis: The required minimum switch duration, in milliseconds. This is “D1min” or “D2min” as defined in a HbbTV TA performance profile, or null to disable checking of the minimum switch duration (e.g. if the terminal does not support a performance profile). Values greater than 10000, or greater than required_duration_max_millis, are not allowed. This does not imply that the API will function correctly with values up to 10000.

required_duration_max_millis: The required maximum switch duration, in milliseconds. This is “D1max” or “D2max” as defined in a HbbTV TA performance profile, or null to disable checking of the maximum switch duration (e.g. if the terminal does not support a performance profile). Values greater than 10000 are not allowed. This does not imply that the API will function correctly with values up to 10000; the total switch time (required_accuracy_millis + required_duration_max_millis) should be less than or equal to 1000ms.

optional: Whether the switch should happen or not. If set to true, it checks that either the switch has happened (within the accuracy for the requested profile) or it hasn’t happened at all. If set to false, then it checks that the switch has happened (within the accuracy for the requested profile).

callback/callbackObject: a callback function to invoke when the analysis was made (also see chapter 7.2.3 Callbacks). The analysis result is not passed back to the application. A failed analysis must cause the complete test to fail. This is because the analysis may be performed after the test has completed.

to_timeline_time_early_millis: If not null and not undefined, this is the allowed inaccuracy of the playback start time. The actual playback start time may be this number of milliseconds before the specified to_timeline_time. If to_timeline_time_ millis is null, then this parameter must be null or undefined. Values greater than 600000 milliseconds (10 minutes) are not allowed.

to_timeline_time_late_millis: If to_timeline_time_early_millis is null or undefined, then this parameter must be null or undefined. Otherwise, this is the allowed inaccuracy of the playback start time. The actual playback start time may be this number of milliseconds after the specified to_timeline_time. Values greater than 600000 milliseconds (10 minutes) are not allowed.

NOTE: Test cases can calculate expected_switch_ millis in several ways, including:

- if switching broadcast to broadband:

  getPlayoutStartTime() + (from_timeline_time_millis - broadcast video start timeline time in milliseconds) - getCurrentTime()

- if switching broadband to broadcast using a broadcast timeline:

  getPlayoutStartTime() + (to_timeline_time_millis - broadcast video start timeline time in milliseconds) - getCurrentTime()

- if switching broadband to broadcast using a broadband timeline or at the end of the broadband media:

  (from_timeline_time_millis - current broadband video timeline time in milliseconds)

The test harness shall:

1. At an appropriate time at least (expected switch UTC accuracy + minimum audio duration to guarantee decoding) == (5 + 2) == 6 seconds before the expected switch UTC time, start monitoring the audio and video outputs of the terminal
2. Continue monitoring until either at least 2 seconds after the switch has completed, or at least (expected switch UTC accuracy + max switch duration + minimum audio duration to guarantee decoding) == (5 + 5 + 2) = 12 seconds after the expected switch UTC time.

   NOTE: The max switch duration of 5 seconds here is fairly arbitrary, it is chosen to be "longer than any reasonable terminal could take". If (required_accuracy_millis + required_duration_max_millis) exceed this value, then the analysis becomes unreliable.

3. Either immediately, or at a later time after the test has finished running:
   a. Analyse the captured data as described below.
   b. Report the step as passed or failed.
   c. If failed, mark the complete test case as failed.
4. Unless the step is known to have failed, call the JS callback function.

The analysis referred to above is:

1. Analyse the QR codes. (Note: This analysis step can be done e.g. on a webcam capture):
   a. Run QR recognition on the video.
   b. Check that the QR codes in the video at the start of the capture (the "from" video) indicate broadcast video if HbbTVTestAPI.SWITCH_BROADCAST_TO_BROADBAND was passed, or broadband video if HbbTVTestAPI.SWITCH_BROADBAND_TO_BROADCAST was passed
   c. Check that the "from" video ceased being displayed.
   d. Check that the last frame of "from" video was not displayed for an excessive duration (based on the video frame rate passed as a parameter).

      NOTE: The HbbTV TA spec says 1 frame duration, but the harness is not required to measure that accurately.

e. Check the "from" video was playing at a normal frame rate without discontinuities or rate changes.
f. Record the timeline time for the end of the last frame of "from" that was displayed. This is Xv_timeline.
g. Find the next capture frame containing recognizable QR codes. (This is the start of the "to" video).
h. Check the QR codes in the "to" video indicate broadband video if HbbTVTestAPI.SWITCH_BROADCAST_TO_BROADBAND was passed, or broadcast video if HbbTVTestAPI.SWITCH_BROADBAND_TO_BROADCAST was passed
i. Record the timeline time for the start of the first frame of "to" video that was displayed. This is Sv_timeline.
j. Check the "to" video was playing at a normal frame rate without discontinuities or rate changes.

2. Analyse the Light Sensor areas on the video:

    a. For HbbTVTestAPI.SWITCH_BROADCAST_TO_BROADBAND mode, check that:
        i. at the start of the capture, Light Sensor 1 reads black and Light Sensor 2 reads white
        ii. then Light Sensor 2 goes black. Remember the UTC time of this, and call it Xv_UTC.
        iii. then (at a later time, or at the same time or approximately the same time) Light Sensor 2 goes white. Remember the UTC time of this, and call it Sv_UTC.
        iv. the light sensors don't change again during the capture period.
    b. For HbbTVTestAPI.SWITCH_BROADBAND_TO_BROADCAST mode, do the above but swap Light Sensor 1 for Light Sensor 2 and vice-versa

2. Analyse the audio capture
    a. Analyse the captured audio, decoding the time code patterns. Check there is:
        i. a continuous pattern at the start, with the appropriate frequency depending on the passed “direction”.
        ii. then possibly some silence
        iii. then a second continuous pattern to the end of the capture, with the appropriate frequency depending on the passed “direction”.
    b. The harness calculates the time the first audio stopped, both in timeline time (Xa_timeline) and UTC (Xa_UTC). (The data decoded from the audio gives timeline time; the timings measured by the harness give UTC).
    c. The harness calculates the time the second audio started, both in timeline time (Sa_timeline) and UTC (Sa_UTC). (The data decoded from the audio gives timeline time; the timings measured by the harness give UTC).

3. Do the analysis:
    a. We have 8 measurements: Xv_timeline, Xv_UTC, Sv_timeline, Sv_UTC, Xa_timeline, Xa_UTC, Sa_timeline, Sa_UTC
    b. The test results XML file should include all 8 of those measurements.
    c. We were passed in these parameters:
        i. from_timeline_time_millis
        ii. to_timeline_time_millis
        iii. required_accuracy_millis
        iv. required_duration_min_millis
        v. required_duration_max_millis
        vi. from_fps
        vii. to_fps
        viii. to_timeline_time_early_millis
        ix. to_timeline_time_late_millis
    d. In all the following, conversions between seconds, milliseconds etc are implied. I.e. you need to convert all the time variables to seconds (or convert them all to milliseconds) before doing each check. Also, all timeline times are modulo 4096 seconds (4096000 milliseconds).
    e. Calculate:
        i. A = required_accuracy_millis
        ii. Dmin = required_duration_min_millis

iii. Dmax = required_duration_max_millis

iv. X_UTC = min(Xv_UTC, Xa_UTC)
v. S_UTC = min(Sv_UTC, Sa_UTC)
vi. D = S_UTC - X_UTC

vii. S_tolerance = (1 second / to_fps)
(This is a 1 frame tolerance. The spec requires playback to start precisely where requested, but the harness can't necessarily measure that accurately, so we allow this tolerance)

viii. av_sync_tolerance = 32ms (max audio frame duration)
ix. new_audio_appearance_limit = 500ms

f. If A is not null, and from_timeline_time_millis is not null, check that all the following are true:
   i. from_timeline_time_millis – av_sync_tolerance <= Xa_timeline <= from_timeline_time_millis + A+ av_sync_tolerance
   ii. from_timeline_time_millis <= Xv_timeline <= from_timeline_time_millis + A
g. If Dmin is not null, check that all the following are true:
   i. Dmin < D
h. If Dmax is not null, check that all the following are true:
   i. D <= Dmax
i. If A is not null, and from_timeline_time_millis is null, and to_timeline_time_millis is not null, and to_timeline_time_early_millis is null or undefined, check that all the following are true:
   i. to_timeline_time_millis - av_sync_tolerance <= Sa_timeline - (Sa_UTC – Xa_UTC) <= to_timeline_time_millis + new_audio_appearance_limit
   ii. to_timeline_time_millis <= Sv_timeline - (Sv_UTC – Xv_UTC) <= to_timeline_time_millis + S_tolerance + A + D + av_sync_tolerance
j. If from_timeline_time_millis is not null, and to_timeline_time_millis is not null, and to_timeline_time_early_millis is null or undefined, check that all the following are true:
   i. to_timeline_time_millis - av_sync_tolerance<= Sa_timeline <= to_timeline_time_millis + new_audio_appearance_limit
   ii. to_timeline_time_millis <= Sv_timeline <= to_timeline_time_millis + S_tolerance + D
k. If to_timeline_time_early_millis is not null and not undefined, check that all the following are true:
   i. to_timeline_time_millis - to_timeline_time_early_millis <= Sa_timeline <= to_timeline_time_millis + to_timeline_time_late_millis + new_audio_appearance_limit
   ii. to_timeline_time_millis - to_timeline_time_early_millis <= Sv_timeline <= to_timeline_time_millis + to_timeline_time_late_millis + D
l. Xv_UTC <= min(Sv_UTC, Sa_UTC)
m. Xa_UTC <= min(Sv_UTC, Sa_UTC)

A harness is free to use any implementation it likes, so long as the results are the same as the procedures above.

NOTE 1: A fully automated implementation of this test API is possible, where the harness determines pass or fail without any user input.

NOTE 2: The test implementers should take care of fade-in and fade-out effects, being part of a pre-noise technology, effect might be significant - in case of MPEG-H, the fade-in might be even 100 ms. (see **ETSI TS 101 154**)

7.11 APIs for Application Discovery over Broadband (ADB)

7.11.1 Introduction

The Application Discovery over Broadband (ADB) specification enables HbbTV terminals to discover and launch broadcast-related applications via a broadband internet connection when traditional broadcast signalling, such as the Application Information Table (AIT), is unavailable. This situation commonly arises for terminals receiving content via HDMI: When an HbbTV terminal is connected to an external set-top box (STB) via HDMI and receives broadcast content indirectly, it lacks access to the broadcast signalling required for application discovery.

ADB addresses this challenge by allowing the terminal to extract unique identifiers from the broadcast channel—either from watermarks embedded in the audio or video streams. The terminal then uses these identifiers to perform DNS queries to locate and retrieve the appropriate AIT over a broadband connection. This process ensures that users can access interactive services and applications associated with the broadcast content, even in the absence of traditional broadcast signalling.

Testing ADB functionalities, especially in HDMI scenarios, presents unique challenges. To effectively test ADB in HDMI scenarios, testers need the ability to control the content fed into the HDMI input of the device under test (DUT). This includes specifying the exact audio/video streams, including any embedded watermarks or identifiers required for ADB processing.

To facilitate this, a new JavaScript API is introduced within the HbbTV Test Harness. This API allows testers to specify the audio/video content to be played into the HDMI input by providing the path to a media file (e.g., an .mp4 file). By controlling the HDMI input stream programmatically, testers can simulate various broadcast scenarios, including those involving specific watermarks required for ADB processing. This capability is essential for comprehensive testing of ADB functionalities in HDMI-connected environments.

7.11.2 JS-Function setHdmiInputStream()

This function instructs the test harness to play a specified audio/video stream into the HDMI input of the DUT, in full screen, without any alteration nor cropping. It is used when testing scenarios where the HbbTV terminal receives content via HDMI, such as from an external set-top box (STB), and needs to process ADB functionalities based on the HDMI input.

```
void HbbTVTestAPI.setHdmiInputStream(
        streamPath: String,
        callback : function or null,
        callbackObject: object);
```

ARGS: **streamPath:** The file path to the media file (e.g., .mp4) containing the audio and video content to be played into the device's HDMI input. An empty or null string shall stop any ongoing HDMI input. If the specified streamPath is invalid or the file cannot be played, the test harness shall fail the test.

callback/callbackObject: a callback function to invoke when the action has been completed by the test operator (also see chapter 7.2.3 Callbacks).

Test implementers should not call this function when network connection is down, as it may fail when network connection is not available. In this case, this will cause the complete test to fail automatically.

8 Versioning

This chapter contains the version control rules for the Test Specification document, the Test Cases (XML files incl. test material) and the Test Suite.

The version control of these parts is affected by the following events:

1) Challenges to existing test cases, see [27].

2) Filling gaps where test material is missing.

3) Areas where the Test Specification isn't detailed enough.

4) New versions of the Technical Specification (system specification)

8.1 Versioning of Technical Specification and Test Specification documents

Each HbbTV Test Specification shall contain the associated HbbTV Technical Specification Version in its title.

Test Specifications under development shall indicate their intermediate status by a draft number.

When a new version of the HbbTV Technical Specification has been created and approved, a new version of the Test Specification document shall be created.

The Test Specification document also includes references to the several parts of the Test Suite. This includes a list containing all necessary Test Cases (stored in the HbbTV Test Repository) which must be successfully performed by the DUT, in order to finish the technical certification process. This list is "List_of_approved_Test_Material__x_xx.txt", where "x_xx" in the filename refers to the Test Suite Version.

During the incremental process of HbbTV testing, above mentioned events 1, 2 or 3 might happen. This means that either:

- New Test Cases and new test material is created,
- Existing Test Cases are refined or
- Existing Test Cases are challenged the therefore excluded from the list of approved Test Material documents.

All this will lead to a new version of the Test Specification document including the list of approved Test Material which refers to the applicable Test Cases stored in the Test Repository.

8.1.1 Initial status of the Test Specification

The first formal released HbbTV Test specification shall start with a version number 1. Figure 7 depicts an example of the version structure of the initial test spec. release.

In this example the HbbTV technical specification has a version 1.1.1, which is bound to a version 1 HbbTV test specification which makes use of Test Cases that shall have a version 1.

Test Specification and Test Case version numbers shall only have integer numbers.

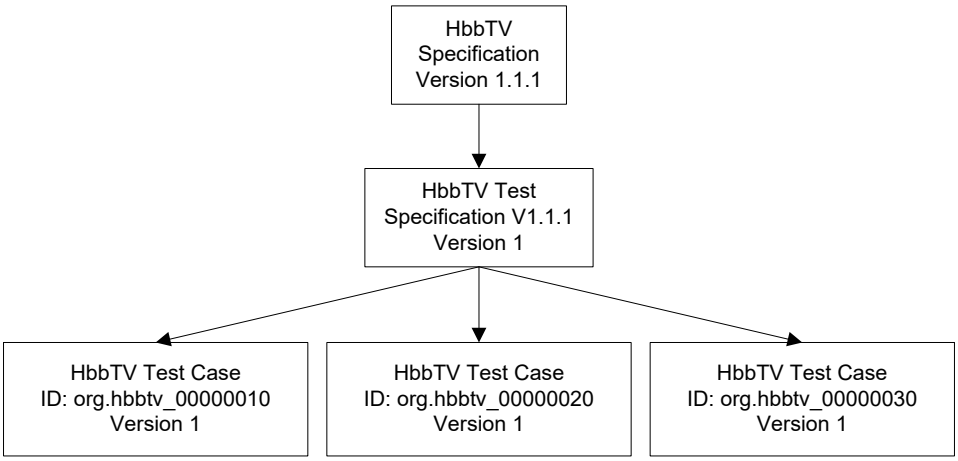


Figure 7: Version structure of the initial Test Specification release

8.1.2 Updating Test Specification, keeping the existing Technical Specification version

Test Cases may be developed to further improve the compliance testing and certification process:

- Test Cases may be improved, upgraded or corrected due to a challenge. In this case the ID number will be kept identical and the version number increases.
- Test Cases may be added to further increase the coverage of testing. In this case a new ID number will be assigned and the version number starts with 1.
- Test Cases may be deleted due to a successful challenge without a replacement.

Existing ID numbers from deleted Test Cases shall not be reused.

In any of the above cases a new version of the Test Specification shall be created and the existing version shall be made obsolete once the new version Test Specification has been formally released.

An example of the creation of a new Test Specification version is indicated in Figure 8.

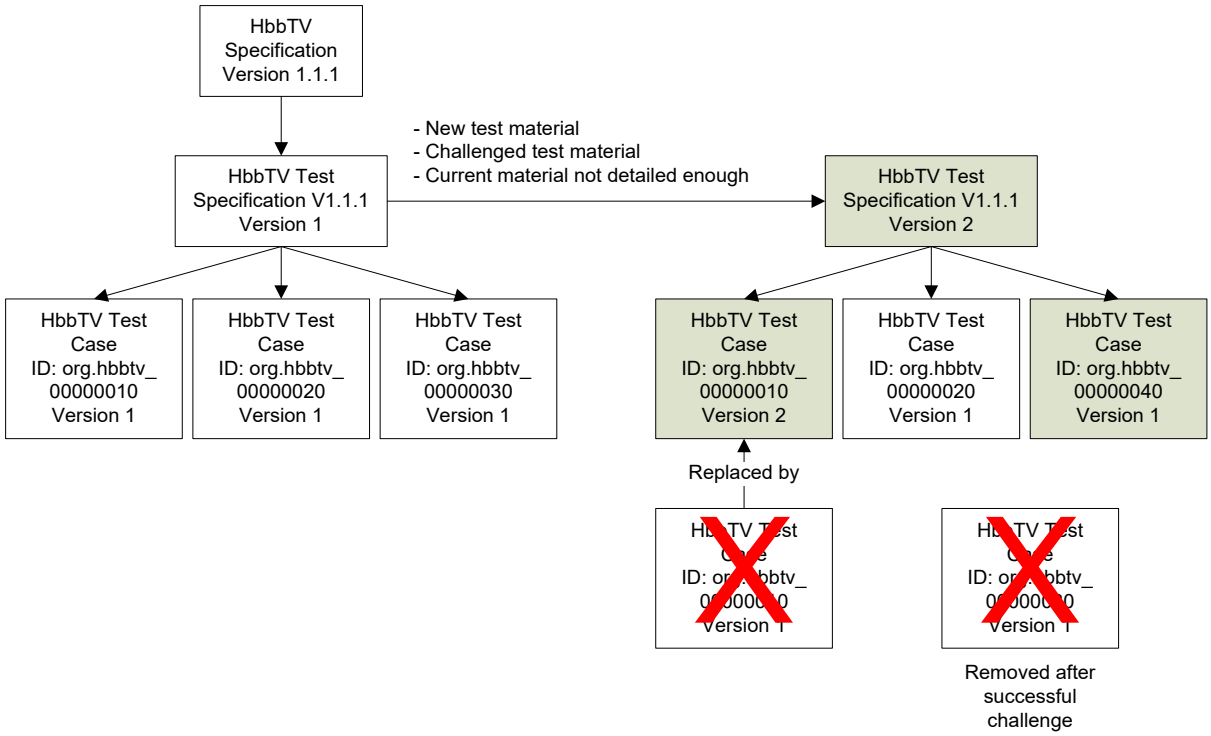


Figure 8: Example of creating a new version Test Specification while keeping the existing version of the Technical Specification

8.1.3 Updating Test Specification after creating a new Technical Specification version.

After a new version of the Technical Specification has been released, a new Test Specification shall also be created and released. New Test Cases and improved Test Cases may be added, as well as deleting obsolete or erroneous Test Cases due to the updated Technical Specification.

- Test Cases may be improved, upgraded or corrected due to a change in the new Technical Specification. In this case the ID number will be kept identical and the version number increases.
- Test Cases may be added due to new technical requirements in the Technical Specification. In this case a new ID number will be assigned and the version number starts with 1.
- Test Cases may be deleted due to obsolete technical requirements.

In any of the above cases a new version of the Test Specification shall be created. The new HbbTV Test Specification shall contain the associated HbbTV Technical Specification Version in its title.

An example of such new Test Specification is given in Figure 9.

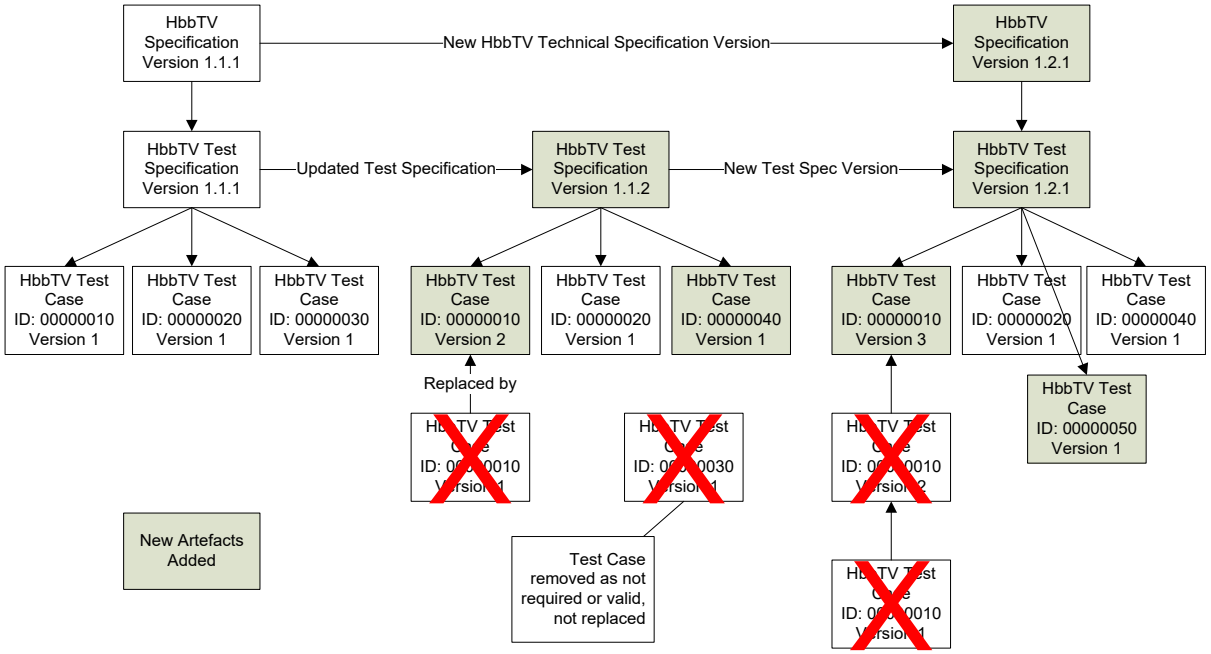


Figure 9: Example of creating a new version Test Specification after creating a new version of the Technical Specification

8.2 Versioning of Test Cases

Central in the Test Case versioning control are the related Test Case items:

- Test Case ID: If the test is changed (e.g. because a problem is identified, or an implementation is added), the test case ID stays the same. If the complete test case must be changed to reflect a modified section of a new HbbTV specification version, a new test case (with a new ID) is created.
- Test Case Version: The version attribute is deprecated. The Test Case ID shall be unique, different from any other HbbTV Test Case.

For Test Case IDs starting “org.hbbtv_”, control of the use of these IDs is administrated by the Test Group.

9 Test Reports

9.1 XML Template for individual Test Cases Result

After the execution of each test case against a DUT, the test results for that test case will be collated into an XML document as defined in /SCHEMAS/testCaseResult.xsd of the Test Suite. The Test Case Result contains:

- Test Case ID: Identifier for the test case being reported (as specified in 6.3.1.1).
- Test Case Version: Version of the test case executed (as specified in 6.3.1.2).

The remainder of the Test Case Result consists of a sequence of the following major sections:

- Device Under Test
- Test Performed by
- Test Procedure Output
- Remarks
- Verdict

These are defined in more detail below.

9.1.1 Device Under Test (mandatory)

This lists the information required to identify the DUT that was tested and under which profile.

9.1.1.1 Model

Text string providing a name for the DUT referring to a specific model or a family name of derivative models (e.g. the same platform with different screen sizes).

9.1.1.2 Hardware Version

Text string providing identification of version of hardware typically would include CPU, chassis type, etc.

9.1.1.3 Software Version

Text string providing identification of version of software typically would include software build number, etc.

9.1.1.4 Company

Name of the company for whom the test is being executed. Typically this would be the manufacturer of the DUT.

9.1.1.5 HbbTV Version

ETSI standard reference for the HbbTV Standard supported by the DUT, as a dot separated set of three integers without spaces (e.g. 1.1.1).

NOTE: The HbbTV Version given here doesn't necessarily match the version of the Test suite used. (E.g. a DUT supporting HbbTV 1.2.1 could try to run the tests from the HbbTV 1.1.1 Test Suite).

This field is mandatory if a Test Report is going to be used for HbbTV compliance purposes. (If the XML schema is being reused by another testing group, and they do not require HbbTV compliance, then this field is optional).

9.1.1.6 HbbTV Capabilities

The terminal options tested on the DUT. It contains a combined list of option strings as defined in section 10.2.4 of the HbbTV Specification. The format of the HbbTV Capabilities value is a text field without spaces.

Available options are +DL for download functionality, +DRM for DRM functionality, +PVR for PVR functionality, +SYNC_SLAVE for slave operation in inter-device synchronisation. Multiple requirements are concatenated to a single string without spaces in between. Example: +DL+PVR

NOTE: This should list the options tested, not necessarily those claimed supported.

9.1.1.7 HbbTV Optional Features

Terminal features which are tested on the DUT. Available features are described in 6.3.3.2.

9.1.1.7 Settings

The value of any settings configured in the harness and adapted to by the test case as described in Section 6.3.4.1 adaptsTo.

9.1.2 Test Performed By (mandatory)

The name of the test operator who executed this test, in the following format:

9.1.2.1 Name

This should be the test operator's full name.

9.1.2.2 Company

The test operator's company name.

9.1.2.3 Email

The test operator's email address or another contact email address for issues regarding the test result.

9.1.3 Test Procedure Output (mandatory)

This includes the trace of test execution both on the DUT and also on the Test Harness. It also includes the following timestamp information.

9.1.3.1 Start Time

UTC time stamp for time at which the test procedure started for this test

9.1.3.2 End Time

UTC time stamp for time at which the test procedure ended for this test

9.1.3.3 Test Step Output

Results of the execution of a test step, this includes a start and end timestamp of the test execution (as UTC timestamp)

9.1.3.3.1 Index

Index number of the executed test step as defined in the Test Case.

9.1.3.3.2 Start Time

UTC timestamp for the time at which this test step started for this test

9.1.3.3.3 End Time

UTC timestamp for the time at which this test step ended for this test

9.1.3.3.4 Step Result

Either “successful” if the test step completed correctly or “not successful” otherwise.

9.1.3.3.5 Test Step Comment

Comments from the execution of the test step given as a parameter in the reportSetResult() and analyze… calls.

9.1.3.4 Test Step Data (conditional)

This element is mandatory for Test Steps which are triggered by calls to analyzeScreenPixel() and analyzeScreenExtended(). If using either of these calls, the MIME type must be either image/png or image/jpeg. This must be an image of the video output of the RUT.

Specific results from the execution of the test step. This includes the following mandatory attributes:

9.1.3.4.1 id

Index (integer) of the result being reported.

9.1.3.4.2 type

The MIME type of the result being reported. E.g. image/png or text/plain.

9.1.3.4.3 href

Relative hyperlink to the result data in a sub-directory. E.g. "./images/screenshot1.png". This path shall:

- use the “/” character to denote the directory,
- not contain the parent directory “..” indicators,
- use only characters that are valid in filename paths on Windows, Unix/Linux and Apple machines,
- start with either a valid folder name, filename or the “.” character, and not with “/”, “//” or any OS dependent drive or machine specifier. I.e. it must be a relative path.

9.1.3.5 Test Server Output (mandatory)

This may contain the output of the harness related to the test that was executed for this result. Ideally the server output will also contain timestamp information for the information it is logging.

9.1.3.5.1 Timestamp

UTC timestamp for time at which Test Harness output was started for this test

9.1.3.5.2 Freeform Server Output

Freeform server output in a way that the generated XML Test Case Result remains valid.

9.1.4 Remarks (mandatory)

Remarks and comments on the execution of this test case. The format of the remarks is a text field. It may be empty.

9.1.5 Verdict (mandatory)

Verdict assigned for the test case, based on the criteria as defined in 6.4:

- PASSED: Test met the pass criteria specified in the Test Case.

- FAILED: Test failed to meet the pass criteria specified in the Test Case.

9.2 Test Report

A Test Report contains Test Case Results for one or more Test Cases in one or more Test Suites. Results shall be stored in a ZIP file containing the following directory structure in its root directory:

- The root directory shall contain one or more Test Suite Results directories, where each directory corresponds to results from a single Test Suite. A typical Test Report for the official HbbTV Test Suite shall contain a single directory.
- Each Test Suite Results directory contains only the results of tests from the corresponding Test Suite. Note that the HbbTV official tests are only contained in one Test Suite, therefore there will only be one Test Suite Results directory needed. However, other standards or trademark licensors may require results from multiple Test Suites in their Test Report.
- The name of the Test Suite Results directory is unspecified, but an informative name that contains the version of the Test Suite is recommended e.g. HbbTV-1_2_1-TestSuite-v1_0_0.
- Each Test Suite Results directory shall only contain a Test Case Result directory for each test case that has been executed.
- The name of each Test Case Result directory shall be in the following format, where {test_case_id} is the Test Case ID of the test case that the results pertain to:
  - {test_case_id}
- The Test Case Result directory shall contain one Test Case Result XML document in the format specified in section 9.2.
- The filename of the Test Case Result XML document shall be in the following format:
  - {test_case_id}.result.xml
- All files referenced by Test Step Data shall be stored in the Test Case Result directory where the Test Case Result XML document is located or a subdirectory therein.

Example:

HbbTV-1_2_1-TestSuite-v1_0_0

org.hbbtv_TEST1

org.hbbtv_TEST1.result.xml

org.hbbtv_TEST2

org.hbbtv_TEST2.result.xml

screenshot1.jpg

Trademark Licensor X special test suite-v2

[…]

ANNEX A File validation of HbbTV Test Material (informative)

Purpose of this activity is to check the validity of files that are part of the Test Material in an automatic and objective fashion wherever possible. E.g. is the HTML valid according to the W3C validator, has a Transport Stream been run through a TS analyzer, etc.

The following table lists the file types that are expected to be validated. The right hand column indicates tools that are used. The Test Coordination Group reserves the right to use alternative tools.

File Type	Example tool
JavaScript	JSLint (http://jslint.comcom/)
HTML	W3C Markup Validation Service (http://validator.w3.org/)
CSS	W3C CSS Validation Service (http://jigsaw.w3.org/css-validator/)
XML-Validation	W3C XML Validator (http://www.w3.org/2001/03/webdata/xsv)
MPEG-TS (ETR 101 290)	No recommendation
MP4 file format	No recommendation
MPEG-2/H.264 video content	No recommendation
Image (PNG/JPEG)	No recommendation
MPEG-2 Transport Streams	MPEG-2 Transport Stream packet analyzer: http://www.pjdaniel.org.uk/mpeg/

Table 6: File type validator example tools

ANNEX B Development management tools

To support development and maintenance of the test suite HbbTV hosts a set of collaboration tools.

B.1 Redmine

Redmine (https://redmine.hbbtv.org/) is an online project management tool used by HbbTV for many of its specification and testing project management needs. There are currently four projects used by the Testing Group for test suite management:

1) HbbTV Testing is the main group coordination project,
2) HbbTV Test Review is used to track faults identified during test case development and review
3) Test Challenge is used to log challenges on released test cases
4) Test Material Support is used by test suite customers to request help in use of the released test suites.

Members of the Testing Group are eligible to use all of these projects. If you require access please contact support@hbbtv.org.

B.2 Subversion

Subversion is a well-known version control system. It is used to store Test Cases and tools for their development management. This is also the location of schema files for the various standard file formats used for test case creation and test harness configuration. The repository is located at https://eu-subversion.assembla.com/svn/hbbtv-association^HbbTV.Test.

B.2.1 Access to the subversion repository

Access to the subversion repository is controlled. Firstly, any member company who wishes to access the repository must complete the Test Repository Access Agreement. Details for obtaining and completing this are to be found at https://www.hbbtv.org/resource-library/#testing-information-and-support. The page also describes how to create login credentials, using htpasswd, to enable you to login.

ANNEX C External hosts

For testing TLS the test system must allow the DUT to access a number of external hosts, as listed in the table below.

https://valid.sfsg2.catest.starfieldtech.com
https://aaacertificateservices.comodoca.com
https://addtrustclass1caroot.comodoca.com
https://addtrustexternalcaroot-ev.comodoca.com
https://addtrustpubliccaroot.comodoca.com
https://addtrustqualifiedcaroot.comodoca.com
https://comodocertificationauthority-ev.comodoca.com
https://comodorsacertificationauthority-ev.comodoca.com
https://securecertificateservices.comodoca.com
https://trustedcertificateservices.comodoca.com
https://usertrustrsacertificationauthority-ev.comodoca.com
https://utnuserfirsthardware-ev.comodoca.com
https://baltimore.omniroot.com
https://ev.omniroot.com
https://assured-id-root.digicert.com/testroot
https://assured-id-root-g2.digicert.com
https://global-root.digicert.com/testroot
https://global-root-g2.digicert.com
https://ev-root.digicert.com/testroot
https://trusted-root-g4.digicert.com
https://2021.globalsign.com
https://2029.globalsign.com
https://2028.globalsign.com
https://valid.gdi.catest.godaddy.com
https://valid.gdig2.catest.godaddy.com
https://valid.sfi.catest.starfieldtech.com
https://valid.sfig2.catest.starfieldtech.com
https://ssltest14.bbtest.net
https://ssltest19.bbtest.net
https://ssltest22.bbtest.net
https://ssltest21.bbtest.net
https://ssltest20.bbtest.net
https://ssltest34.bbtest.net
https://ssltest6.bbtest.net
https://ssltest8.bbtest.net
https://ssltest3.bbtest.net
https://ssltest1.bbtest.net
https://ssltest26.bbtest.net
https://comodoecccertificationauthority-ev.comodoca.com
https://usertrustecccertificationauthority-ev.comodoca.com
https://assured-id-root-g3.digicert.com
https://global-root-g3.digicert.com
https://2038r4.globalsign.com
https://2038r5.globalsign.com
https://ssltest42.ssl.symclab.com
https://ssltest40.ssl.symclab.com
https://ssltest48.ssl.symclab.com

ANNEX D Extensions for testing Operator Applications

D.1 General Approach

This appendix is about testing a terminal against the Operator Applications specification [37]. It is mandatory for test harnesses to implement the functionality described in this appendix if they support Operator Application testing, otherwise it is optional. Test cases shall not rely on the functionality in this appendix unless they test Operator Applications.

Initial testing will usually be carried out by the manufacturer (or a 3rd party test house on their behalf), because they are the people who can fix any problems found. Operators may also run tests themselves (or get a 3rd party test house to run the tests on their behalf) to verify compliance.

A lot of the OpApps specification depends on the choices that the operator and/or manufacturer have made. The operator and manufacturer agree a "bilateral agreement" which specifies many of the details that affect testing, such as keys and domain names and which options the manufacturer implements. So testing has to take place in the context of a specific bilateral agreement.

There are two approaches to certification for a specific operator: Testing can be carried out with real encryption keys and certificates, or with special test keys and certificates. Using real keys is more realistic, but using test keys may be more secure. It is up to the operator which approach is used. If the operator chooses to test with real keys and certificates, then the operator must arrange to provide such. If the operator chooses to test with test certificates, then the operator should provide them to ensure they are consistent with the structure of the live certificates.

A manufacturer may want to test their functionality before a bilateral agreement with a real operator exists. To do this, they can invent a fake operator just for testing purposes. A manufacturer may choose to do that to test their in-development terminal, or to test discovery methods that no currently-supported operator uses. This allows them to prove their software with the intent of updating their software later to add a real operator (or more real operators). In this case, since the operator is invented, the manufacturer can invent the corresponding "bilateral agreement" too, to ensure the features they want tested get tested.

A manufacturer can make a device that supports multiple operators. In this case, they may wish to repeat testing for each bilateral agreement they have.

Alternatively, a manufacturer may wish to do a single test run that exercises every feature used by every operator they wish to support, which may be much quicker. To do that they can invent a fake operator with an invented "bilateral agreement", and test against that fake operator. The invented "bilateral agreement" needs to include every feature needed by any real operator that they want to support.

This gives 3 approaches to testing:

1. Testing for real operator with real keys
    - Most realistic
2. Testing for real operator with special test-only keys
    - Required by some operators for security reasons
3. Testing for fake (invented just for testing) operator with test-only keys
    - Allows testing before agreement with real operator
    - Allows one pass of testing to cover all features needed by multiple operators
    - Allows testing features not used by any operator yet (e.g. alternate discovery modes)
    - NOTE: In this case the test keys may be created by the manufacturer, or shipped with a test harness, or shipped with the test suite.

NOTE: For security reasons test certificates should not be active by default in production devices, although the manufacturer may have a setting (usually hidden) to turn them on.

NOTE: A tester must be able to reset the device under test to its as-shipped condition in order to be able to run the test case org.hbbtv_OPAPP_INSTALL12.

D.2 The Bilateral Agreement

There will be a bilateral agreement between the operator that is being tested and the manufacturer. In the case where there is no real operator involved this may be a fake agreement with a notional operator. Where the operator does not want to use real encryption keys and certificates the bilateral agreement used in testing will contains the testing credentials. The tester must have:

- The ability to configure the test harness's TLS Server, which requires:
  - Knowing the domain name for the Test TLS Server (may be specified by the bilateral agreement)
  - the Test TLS Server certificate, corresponding private key, and corresponding certificate chain
  - Other TLS certificates, corresponding private keys, and corresponding certificate chains required for specific test cases:
    - Revoked Certificate (for org.hbbtv_OPAPP_INSTALL66)
    - Expired Certificate (for org.hbbtv_OPAPP_INSTALL65)
    - Potentially more to test additional failure conditions
  - If client certificates are specified in the bilateral agreement, then Client CA root certificate(s)
- The ability to create signed OpApps, which requires:
  - Operator Signing Certificate, corresponding private key, and corresponding certificate chain
  - Alternatively, if using real keys then signing the test OpApps will normally be done by the operator, so they can keep control of their private key.
  - The OpApps are:
    - Launcher OpApp
    - Stub OpApps, listed in section 0
- Terminal Packaging Certificate (specific to the terminal being tested, and specified by the bilateral agreement)
  - Note that the OpApps are signed first, then encrypted with this certificate, so one set of signed OpApps can be reused for all terminals that support that operator.
- A terminal that has been loaded with the relevant operator's:
  - discovery information
  - if applicable: operator-specific TLS server root CA certificate(s)
  - OpApp signing root CA certificate(s)
  - terminal packaging private key
  - if applicable: client certificate and private key, and any intermediate certificates
- The ability to configure the Test Harness with some of the information from the Bilateral Agreement in key/value pairs. These values shall be available to the test case through the JS APIs defined in "D.2.3 JS API to query discovery settings" and "D.2.5 Void".

NOTE: It may be useful to define the set of test cases to be executed based on the preconditions in the test cases and the bilateral agreement at the time that the bilateral agreement is developed.

D.2.1 Categories of information

The information from the bilateral agreement is grouped into three categories: "optional features", "discovery" and "extended":

- The "optional features" category is for things that can be handled with the existing optional features preconditions mechanism described in "6.3.3.2 Optional Features" and with the additional features defined in "D.3.5.1 Operator Application Optional Features".
- The "discovery" category covers everything needed to configure the test harness's TLS server and install and run the launcher application.
- The "extended" information is everything else.

D.2.2 Discovery settings

The tester must configure the test harness with the following information from the bilateral agreement:

- Default OpApp discovery method. Mandatory. The tester must choose 1 from following 8 options:

    - 1n) NIT with URI_linkage_descriptor with FQDN, DNS SRV lookup, XML AIT from broadband, app from broadband
    - 1b) BAT with URI_linkage_descriptor with FQDN, DNS SRV lookup, XML AIT from broadband, app from broadband
    - 2) Hardwired FQDN, DNS SRV lookup, XML AIT from broadband, app from broadband
    - 3) Hardwired XML AIT URI, XML AIT from broadband, app from broadband
    - 4) DNS SRV lookup to standardised address, XML AIT from broadband, app from broadband
    - 5) CICAM NIT with URI_linkage_descriptor with URI of XML AIT, XML AIT from broadband, app from broadband
    - 6n) NIT with URI_linkage_descriptor with URI of AIT, broadcast AIT, app from DSM-CC broadcast
    - 6b) BAT with URI_linkage_descriptor with URI of AIT, broadcast AIT, app from DSM-CC broadcast

  NOTE 1: The options and numbers here match table 5 in the OpApps specification [37]. Options 1 and 6 in that table have 2 sub-options, NIT or BAT, hence 1n/1b/6n/6b.

  NOTE 2: A particular test harness may not support all of these options. E.g. a test harness may decide not to support method 5. Such a harness is not suitable for certifying terminals that claim to support method 5, but can be used to test and certify terminals that do not support method 5.

- TLS server domain name. Mandatory. If the terminal supports discovery mode 3, this must match the domain name from the hardcoded URL.
- TLS server port number A (for most HTTPS usage). If the terminal supports discovery mode 3, this must match the port number from the hardcoded URL. Mandatory. Default is 443.
- TLS server port numbers B, C, D, E. (Only used by a small number of test cases). Mandatory. Defaults may be provided by the harness.
- The normal "TLS certificate bundle". (This is referred to in this appendix as the "good" configuration). For the purposes of this appendix, a "TLS certificate bundle" consists of:
    - TLS server certificate file. Mandatory.
    - TLS server private key file. Mandatory.
    - TLS server intermediate certificates file. Optional.
- The "TLS certificate bundle" (as described above) for the TLS server which will cause a TLS error because the host name does not match the certificate. (This is referred to in this appendix as the "host-name-mismatch" configuration).
- The "TLS certificate bundle" (as described above) for the TLS server which will cause a TLS error because the certificate is self signed. (This is referred to in this appendix as the "self-signed" configuration).
- The "TLS certificate bundle" (as described above) for the TLS server which will cause a TLS error because the certificate has a server IP address that is not the same as the IP address the harness is using for this server. (This is referred to in this appendix as the "ip-address-mismatch" configuration).
- The "TLS certificate bundle" (as described above) for the TLS server which will cause a TLS error because the certificate has expired. (This is referred to in this appendix as the "expired-certificate" configuration).
- The "TLS certificate bundle" (as described above) for the TLS server which will cause a TLS error because the certificate is on the CRL (i.e. it is revoked). (This is referred to in this appendix as the "revoked-certificate" configuration).
- Client CA(s) for TLS server client certificates (a single file that may contain multiple CA certificates). Optional. If not specified, then client certs are not expected to be used.
- Number of intermediate certificates allowed for TLS server client certificates. Cannot be set if Client CA(s) is not set, mandatory if Client CA(s) is set.
- Network ID (a default value [matching the pre existing base stream] will be used if not explicitly configured)
- Organization ID (a default value of 0x400 == decimal 1024 will be used if not explicitly configured). To be used for OpApps and for regular HbbTV applications that need to share the same organization ID as the OpApps. Must not match the normal Organization ID used by HbbTV tests as defined in "7.5.1.7 Choose the correct application and organization ID for your application".

NOTE: One reason this should not match the normal Organization ID used by HbbTV tests is because there is a risk of confusion if the same orgid/appid are used for normal HbbTV apps in existing HbbTV tests and by OpApps in the OpApp tests – e.g. consider running a pre-existing HbbTV test while an OpApp with the same orgid/appid is still installed.

NOTE: Another reason this should not match the normal Organization ID used by HbbTV tests is because some test cases need a normal HbbTV app with a different orgid, so it is natural to use the existing HbbTV testing Organization ID for the normal HbbTV apps and a different Organization ID for the operator apps.

- Application ID to be used for the stub OpApp var_app_id for use in test cases where the stub OpApp has to have a specific application ID to allow the terminal to launch it itself without direct input from the testcase. Legal values are any integer in the range 0-65535. Optional, defaulting to 16. If a bilateral agreement requires different values to be used for this app in different test cases, the tester should re-configure the harness before running the different test cases.
- Bouquet ID (optional unless discovery method 1b or 6b are supported by the terminal in which case it is mandatory)
- Hardcoded operator FQDN for discovery method 2. (Mandatory if discovery method 2 is supported by the terminal, otherwise not used. Note that a terminal may support multiple discovery methods, if the terminal supports discovery method 2 then this must be set even if a different discovery method is being used by default)
- Second hardcoded operator FQDN for discovery method 2. (Optional if discovery method 2 is supported by the terminal, otherwise not used)
- Hardcoded URL suffix for XML AIT for discovery method 3. This is a URL suffix, i.e. it does not include the "https://" bit, or the host name or port which will be taken from the harness's TLS server configuration. (Mandatory if discovery method 3 is supported by the terminal, otherwise not used. Note that a terminal may support multiple discovery methods, if the terminal supports discovery method 3 then this must be set even if a different discovery method is being used by default).
- Second hardcoded URL suffix for XML AIT for discovery method 3. (Optional if discovery method 3 is supported by the terminal, otherwise not used).
- Whether the operator apps should be "privileged" or "operator-specific" by default. For terminals supporting only one type of operator app, this shall be the supported type. For terminals supporting both types of operator app, this shall be "operator-specific". This information is configured by configuring the harness with the DUT optional feature +OPAPP_OPERATOR or -OPAPP_OPERATOR (see section D.3.5.1).
- Maximum uncompressed OpApp size in bytes.

D.2.3 JS API to query discovery settings

There is an JS API:

```
void getOpAppDiscoverySettings(callback, callbackObject)
```

This can only be called when the network is available. It calls the callback like this:

```
callback(callbackObject, opAppDiscoverySettings)
```

opAppDiscoverySettings is an object with the following methods, which all return immediately:

```
string OpAppDiscoverySettings.getTlsServerUri()
```

returns the URI to the TLS server used to serve OpApps. This shall be of the form "https://<domainname>/" or "https://<domainname>:<port>/" – note that the HTTPS protocol and trailing slash are always included.

```
String OpAppDiscoverySettings.getOrgIdHex()
```

Returns the organization ID for operator apps, as a string in hex format with lowercase letters and no prefix. (This matches the format used in DVB URIs).

```
int OpAppDiscoverySettings.getOrgIdInt()
```

Returns the organization ID for operator apps, as a number.

```
int OpAppDiscoverySetting.getVariableAppIdHex()
```

Returns the configured application ID used for the stub OpApp var_app_id, as a string in hex format with lowercase letters and no prefix. (This matches the format used in DVB URIs).

```
int OpAppDiscoverySetting.getVariableAppIdInt()
```

Returns the configured application ID used for the stub OpApp var_app_id, as a number.

```
String OpAppDiscoverySettings.getNidHex()
```

Returns the network ID used, as a string in hex format with lowercase letters and no prefix. (This matches the format used in DVB URIs).

```
int OpAppDiscoverySettings.getNidInt()
```

Returns the network ID used, as a number.

```
int OpAppDiscoverySettings.getDefaultOpAppKind()
```

Returns the configured default OpApp kind, as a string, either “urn:hbbtv:opapp:privileged:2017” or “urn:hbbtv:opapp:opspecific:2017”.

D.2.4 Void

The previous content of this section has been moved to 7.2.13 Extended Settings

D.2.5 Void

The previous content of this section has been moved to 7.2.14 JS-Function getExtendedSetting

D.2.6 Void

The previous content of this section relating to getDutOptions has been moved to sections 6.3.4, 7.2.10, 7.4.4.3.1, and 9.1.1.7.

D.3 Test Launching

D.3.1 Kinds of Test Case

For discussions about how tests are launched, we divide the HbbTV test cases into 3 kinds:

- “Regular HbbTV Tests”
- “Launcher-based OpApp Tests”
- “OpApp Discovery Tests”

“Regular HbbTV Tests” are launched using the AIT or harness-based tests, as defined in sections “5.2.1.6 ECMAScript Environment” and “7.4.4 Transport stream requirements” of the current document. All the HbbTV tests pre-existing the Operator Applications specification are Regular HbbTV Tests.

“Launcher-based OpApp Tests” are launched using a special launcher OpApp as described in the “Starting Launcher-based OpApp Tests” section below. A test case is a “Launcher-based OpApp Test” if and only if the implementation.xml file has the <runOpAppPage> tag (defined below).

“OpApp Discovery tests” are test cases that test how an operator application is discovered and run. They have special rules regarding launching, as described in the “Starting OpApp Discovery Tests” section below. They may or may not include a test Operator Application (therefore some test cases may be harness-based only).

D.3.2 Starting Launcher-based OpApp Tests

The Test Harness shall provide a TLS server. The domain name and TLS certificates used by that server shall be acceptable under the relevant bilateral agreement.

The Test Harness's DNS server shall respond to "A" lookups for the TLS server's DNS name, with the relevant server IP address.

The Test Harness shall provide a "Launcher OpApp". The Test Harness shall also provide appropriate OpApp discovery signalling to enable the terminal to find the Launcher OpApp. The details of this OpApp discovery signalling depends on the bilateral agreement. The tester shall install the Launcher OpApp on the terminal before running the Launcher-based OpApp Test, the details of how this is done depend on the bilateral agreement.

Each Launcher-based OpApp Test shall include this element in its implementation.xml file:

```
<runOpAppPage path="index.html5"/>
```

Where the specified path is relative to the directory containing the implementation.xml file. A test case is classified as a "Launcher-based OpApp Test" if and only if specifies such a HTML page.

When starting the test case, the Test Harness shall arrange for the Launcher OpApp to replace the HTML page with the one specified by the runOpAppPage tag (e.g. using "window.location = …"). This means that as far as the terminal is concerned there is a single OpApp that keeps running all the time, it just switches from the Launcher HTML page to the test case's HTML page (and back at the end of the test case). At the page switch, the state of the OpApp shall be:

- Foreground state
- Visible

NOTE: It is expected that the Launcher will display a UI on screen that indicates that the launcher is running. So the Launcher will be in Foreground state and visible. It is expected that many test cases will want to show progress information on screen, like other HbbTV test cases do, so will want to be in Foreground state and visible. Other test cases will need to switch to a different state, and they can do that. But Foreground & visible seems like the best state to default to, and it avoids unnecessary state transitions.

- Not replacing any terminal UI
- Not replacing the channel list
- Keyset is unset
- Running as the launcher app – i.e. organization ID matching the one configured in the harness, application ID 100 decimal, an "operator-specific" application if supported by the terminal else an "privileged" application.
- A channel scan has been done and has found the standard HbbTV base stream (with network IDs changed to the one configured in the harness)
- Service "ATE Test 11" is selected and is presenting audio and video, and the terminal is monitoring it for AITs.

The URL used to launch the test case's HTML page will be an "https:" URL, with the test suite mapped to "/_TESTSUITE/" on the specified server. The domain name of the URL depends on the bilateral agreement. For example, it might be "https://example.com:8888/_TESTSUITE/TESTS/org.hbbtv_EXAMPLE/index.html5".

The HTML page specified must load testsuite.js and create an HbbTVTestAPI object in the usual way. However, it shall pass an argument to the HbbTVTestAPI constructor, which shall be the string "OpAppPage". I.e:

```
var testApi = new HbbTVTestAPI("OpAppPage");
```

When a test case completes (whether pass, fail or error), or if a user cancels execution of a test case, then the test harness provided code in testsuite.js may replace the HTML page with another one. The test harness may do that to move back to the Launcher OpApp page.

There is a mechanism for a test case to specify steps to take after the test completes to undo any changes that would otherwise be preserved across page navigation, as follows:

A test case may optionally provide a global function called tearDown(). If tearDown() is provided then the test harness code in testsuite.js should call tearDown() immediately before replacing the HTML page. tearDown()

should undo any changes that the test case might have made that would otherwise be preserved across the page navigation, to make it possible to run multiple tests one after another.

If the harness chooses to implement this redirection back to the Launcher HTML page, it should make a good-faith effort to put the OpApp into the Foreground+Visible state, if it is not already in that state.

NOTE: A test case may be able to leave the terminal in a state such that the harness won't be able to recover to the Launcher page in the normal state. This is especially true on buggy terminals that don't comply with the specification. If getting back to the Launcher fails, the tester will have to manually relaunch the Launcher OpApp, perhaps by power cycling the terminal.

A launcher based test may launch a different OpApp with the createApplication(…, …, true) API. In this case, the OpApp that is launched shall initialise the HbbTV Test API with a special argument, like this:

```
var testApi = new HbbTVTestAPI("OpAppReplacedLauncher");
```

In this case, when the test case ends the harness may optionally use createApplication(<harness-specific URL to launcher app>, false, true) to return to the launcher application.

The launcher application behaviour is implementation-specific and the current document does not specify what will happen if it is killed and then relaunched during the test case. Test cases should not do that. If a test case needs an installed OpApp to be relaunched during the test case, the test case should use a stub application.

D.3.3 Starting OpApp Discovery Tests

The harness shall provide a specified set of stub OpApps, which can be launched as part of running the test case, using app discovery configuration mostly specified in the test case. Some details of the app discovery configuration, such as FQDNs and TLS server domain names, will be filled in by the harness, using configuration based on the bilateral agreement. The stub OpApps will immediately switch to a HTML page specified in the XML fileas defined below. Any parameters passed to the stub OpApp will be passed to the new HTML page. The XML syntax, in implementation.xml, is:

```
<stubOpApp ident="gen_a" target="index.html5" mode="html"/>
```

The stub OpApps will be served from the harness's test TLS server. It is also possible to deliver the stub OpApps via DSMCC.

All of the discovery tests will involve harness-based test code as well as OpApp(s). Most of those tests will have a manual step at or near the beginning instructing the tester to start OpApp discovery.

The test application should uninstall itself after running between tests, as on some terminals only a single operator application may be installed.

The test harness may be required to perform a factory reset after each test run.

If the test case has OPAPP_DISCOVERY_* preconditions (as defined in D.3.5.1 Operator Application Optional Features) and the default discovery mechanism configured in the harness is not one which matches those preconditions, then this is an error which the test harness shall detect and handle in a harness-dependent way

D.3.3.1 List of Stub OpApps

The stub OpApps are:

Name	AppID	Version	Comments
gen_a	1	0	Generic stub app. "privileged" or "operator-specific" as configured in harness
gen_b	2	0	Generic stub app for tests that need two or more operator apps. Also supports alternate entry point, see below for details.
gen_c	3	0	Generic stub app for tests that need many operator apps.

gen_d	4	0	Generic stub app for tests that need many operator apps
gen_e	5	0	Generic stub app for tests that need many operator apps
gen_f	6	0	Generic stub app for tests that need many operator apps
gen_g	7	0	Generic stub app for tests that need many operator apps
gen_h	8	0	Generic stub app for tests that need many operator apps
priv	11	0	Always signalled as "privileged"
upg_v1	13	1	Stub app for upgrade tests.
upg_v2	13	2	Same as "upg_v1" (including same appId) but signalled as version 2.
huge	13	2	Same as "upg_v2", except that the application package ZIP file contains additional data (as well as that normally included in the stub application) so that the total uncompressed size of the OpApp exceeds the configured maximum uncompressed OpApp size by 10 kibibytes
mav_a_v3	14	3	"privileged" or "operator-specific" as configured in harness. Version 3 and has minimum application version signalled as version 2.
mav_a_v2	14	2	Same as "mav_a_v3" (including same appId) but signalled as version 2 and has no minimum application version signalled in its XML AIT, and a minimum application version of 2 in the broadcast AIT.
mav_b_v1	15	1	privileged" or "operator-specific" as configured in harness. Version 1 and has minimum application version signalled as version 3.
mav_b_v2	15	2	Same as "mav_a_v3" (including same appId) but signalled as version 2 and has no minimum application version signalled in its XML AIT, and a minimum application version of 2 in the broadcast AIT.
var_app_id	16*	2147483647	For use in test cases where the stub OpApp has to have a specific application ID to allow the terminal to launch it directly. AppId will be replaced with the value configured in the harness discovery settings according to the bilateral agreement (see D.2.2 Discovery settings). Version is set as high as the highest possible MinimumApplicationVersion so that field never prevents this application from being installed.

D.3.3.2 Stub OpApp package locations

The configured stub OpApps shall be available on the harness's HTTPS test server. The URL that can be used to load them is:

`https://<server name>/StubOpApps/packages/<name>.pkg`

E.g. for OpApp "gen_a" if the harness is configured with the domain name "test.operator.com":

`https://test.operator.com/StubOpApps/packages/gen_a.pkg`

To embed the OpApp into DSMCC, use the <file> tag with a special path. To embed the OpApp's icons into DSMCC, use the <directory> tag with a special path. Typically the playoutset XML will contain something like this:

```
<generatedData>
  …
  <dsmcc …>
    <file dst="opapps/<name>.pkg"
      src="HARNESS:/StubOpApps/packages/<name>.pkg"/>
    <directory dst="icons/<name>"
```

```
      src="HARNESS:/StubOpApps/packages/icons/<name>"/>
  </dsmcc>
</generatedData>
```

D.3.3.3 Stub OpApp behaviour

The XML syntax, in implementation.xml, is typically:

```
<stubOpApp ident="gen_a" target="index.html5" mode="html"/>
```

When loaded, in “html” mode the stub OpApp will replace the HTML page with the one specified as the “target” in the “<stubOpApp>” tag. Any URL parameters passed to the stub OpApp will be passed to the target page in the same way. The OpApp should load the script /_TESTSUITE/RES/testsuite.js in the usual way to use the “HbbTVTestAPI” class.

In “html-visible” mode the stub OpApp will replace the HTML page as described for “html” mode but tries to make the OpApp visible first by checking the current state at startup. If the current state is Background or Transient or Overlaid Transient, then the stub OpApp shall call opAppRequestForeground() from the load event of its initial document and fail the test if that call fails.

In “script” mode, the specified JavaScript script is executed after parsing of the stub OpApp’s HTML is complete but before the DOMContentLoaded event has fired (i.e. in a <script defer src=””> block). The “HbbTVTestAPI” class will already be defined before the specified JavaScript script is executed, the test code must not try to load /_TESTSUITE/RES/testsuite.js. The stub app does not define any CSS rules, so the test code can insert its own CSS rules. The <body> contains a single with ID “stub-loading” which the test code will probably want to remove before it inserts its own HTML elements. Note that the test code cannot use document.write, it should use the DOM or innerHtml.

The behaviour described in this section shall apply each time the stub OpApp is launched during a test case.

D.3.3.4 Contents of Stub OpApps

Each stub app consists of standardised XML AIT, broadcast AIT, HTML page, and 4 icons (32, 64, 128 and 256 pixels square, for square pixel aspect ratio displays). The stub app identity (e.g., “gen_a”) and the HTTPS server used by the operator are encoded in the AIT, XML AIT and HTML page. The AIT and XML AIT also encode whether the stub app is “privileged” or “operator-specific”. The stub app loads its Javascript from the test harness. That Javascript will be harness-dependent. However, the stub app itself is harness-independent. The icons are harness-dependent and are not part of the app package (they are loaded directly from HTTPS or DSMCC).

The contents of the standardised XML AIT, broadcast AIT, and HTML page for the stub app with identity “gen_a” are defined below. For the other stub apps, throughout replace “gen_a” with the identity of the other stub app, and “GEN_A” with the identity of the other stub app converted to upper case and the app ID, app version, and MinimumApplicationVersion with the ones defined in section D.3.3.1 List of Stub OpApps. For the stub app with identity “priv” replace {auto} with “urn:hbbtv:opapp:privileged:2017”.

The standardized XML AIT included in a stub app shall have Unix-style LF line terminators, shall end with a single LF, and shall not have a UTF-8 BOM.

NOTE: Some terminals may do a binary comparison of the XML AIT used to discover the OpApp against the XML AIT inside the OpApp, so it is essential that they match precisely (i.e., the files are byte-for-byte identical).

The stub app with identity “gen_a” includes a file opapp.aitx which is the following XML AIT, with “{https}” replaced with the URL of the HTTPS server, the orgId replaced by the configured organization ID, and the ApplicationUsage replaced with the configured application kind:

```
<?xml version="1.0" encoding="UTF-8"?>
<mhp:ServiceDiscovery
    xmlns:mhp="urn:dvb:mhp:2009"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <mhp:ApplicationDiscovery DomainName="hbbtv.de">
        <mhp:ApplicationList>
            <mhp:Application>
                <mhp:appName Language="eng">Stub OpApp GEN_A</mhp:appName>
```

```
                <mhp:applicationIdentifier>
                    <mhp:orgId>0</mhp:orgId>
                    <mhp:appId>1</mhp:appId>
                </mhp:applicationIdentifier>
                <mhp:applicationDescriptor>
                    <mhp:type>
                        <mhp:OtherApp>application/vnd.hbbtv.opapp.pkg</mhp:OtherApp>
                    </mhp:type>
                    <mhp:controlCode>AUTOSTART</mhp:controlCode>
                    <mhp:visibility>VISIBLE_ALL</mhp:visibility>
                    <mhp:serviceBound>false</mhp:serviceBound>
                    <mhp:priority>1</mhp:priority>
                    <mhp:version>00</mhp:version>
                    <mhp:mhpVersion>
                        <mhp:profile>0</mhp:profile>
                        <mhp:versionMajor>1</mhp:versionMajor>
                        <mhp:versionMinor>4</mhp:versionMinor>
                        <mhp:versionMicro>1</mhp:versionMicro>
                    </mhp:mhpVersion>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0001"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0008"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0040"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0200"/>
                </mhp:applicationDescriptor>
                <mhp:applicationUsageDescriptor>
                    <mhp:ApplicationUsage>{auto}</mhp:ApplicationUsage>
                </mhp:applicationUsageDescriptor>
                <mhp:applicationTransport xsi:type="mhp:HTTPTransportType">
                    <mhp:URLBase>{https}/StubOpApps/packages/</mhp:URLBase>
                </mhp:applicationTransport>
                <mhp:applicationLocation>gen_a.pkg</mhp:applicationLocation>
            </mhp:Application>
        </mhp:ApplicationList>
    </mhp:ApplicationDiscovery>
</mhp:ServiceDiscovery>
```

Some stub applications require a minimum application version signalling in the broadband XML AIT. This is done as shown in the following example opapp.aitx template, which is for stub OpApp "mav_a_v3". The red text indicates the differences required to signal the minimum application version:

```
<?xml version="1.0" encoding="UTF-8"?>
<mhp:ServiceDiscovery
    xmlns:mhp="urn:dvb:mhp:2009"
    xmlns:hbb="urn:hbbtv:opapp_application_descriptor:2017"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <mhp:ApplicationDiscovery DomainName="hbbtv.de">
        <mhp:ApplicationList>
            <mhp:Application>
                <mhp:appName Language="eng">Stub OpApp MAV_A_V3</mhp:appName>
                <mhp:applicationIdentifier>
                    <mhp:orgId>0</mhp:orgId>
                    <mhp:appId>14</mhp:appId>
                </mhp:applicationIdentifier>
                <mhp:applicationDescriptor xsi:type="hbb:HbbTVOpAppApplicationDescriptor">
                    <mhp:type>
                        <mhp:OtherApp>application/vnd.hbbtv.opapp.pkg</mhp:OtherApp>
                    </mhp:type>
                    <mhp:controlCode>AUTOSTART</mhp:controlCode>
                    <mhp:visibility>VISIBLE_ALL</mhp:visibility>
                    <mhp:serviceBound>false</mhp:serviceBound>
                    <mhp:priority>1</mhp:priority>
                    <mhp:version>03</mhp:version>
                    <mhp:mhpVersion>
                        <mhp:profile>0</mhp:profile>
                        <mhp:versionMajor>1</mhp:versionMajor>
                        <mhp:versionMinor>4</mhp:versionMinor>
                        <mhp:versionMicro>1</mhp:versionMicro>
                    </mhp:mhpVersion>
                    <mhp:icon mhp:filename="icons/mav_a_v3/dvb.icon.0001"/>
                    <mhp:icon mhp:filename="icons/mav_a_v3/dvb.icon.0008"/>
                    <mhp:icon mhp:filename="icons/mav_a_v3/dvb.icon.0040"/>
                    <mhp:icon mhp:filename="icons/mav_a_v3/dvb.icon.0200"/>
                    <hbb:MinimumApplicationVersion>2</hbb:MinimumApplicationVersion>
                </mhp:applicationDescriptor>
                <mhp:applicationUsageDescriptor>
                    <mhp:ApplicationUsage>{auto}</mhp:ApplicationUsage>
```

```
                </mhp:applicationUsageDescriptor>
                <mhp:applicationTransport xsi:type="mhp:HTTPTransportType">
                    <mhp:URLBase>{https}/StubOpApps/packages/</mhp:URLBase>
                </mhp:applicationTransport>
                <mhp:applicationLocation>mav_a_v3.pkg</mhp:applicationLocation>
            </mhp:Application>
        </mhp:ApplicationList>
    </mhp:ApplicationDiscovery>
</mhp:ServiceDiscovery>
```

The stub app with identity "gen_a" includes a file opapp.ait. This is generated by taking the following XML AIT, generating the broadcast AIT as specified elsewhere in this document, then taking just the application loop entry from that AIT. The minimum application version is always signalled in the broadcast AIT; for stub applications that do not specify a value then the minimum application version is set to match the application version. The starting XML AIT is:

```
<?xml version="1.0" encoding="UTF-8"?>
<mhp:ServiceDiscovery
    xmlns:mhp="urn:dvb:mhp:2009"
    xmlns:hbb="urn:hbbtv:opapp_application_descriptor:2017"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:testait="http://www.hbbtv.org/2018/testAit">
    <mhp:ApplicationDiscovery DomainName="hbbtv.de">
        <mhp:ApplicationList>
            <mhp:Application>
                <mhp:appName Language="eng">Stub OpApp GEN_A</mhp:appName>
                <mhp:applicationIdentifier>
                    <mhp:orgId>0</mhp:orgId>
                    <mhp:appId>1</mhp:appId>
                </mhp:applicationIdentifier>
                <mhp:applicationDescriptor xsi:type="hbb:HbbTVOpAppApplicationDescriptor">
                    <mhp:type>
                        <mhp:OtherApp>application/vnd.hbbtv.opapp.pkg</mhp:OtherApp>
                    </mhp:type>
                    <mhp:controlCode>PRESENT</mhp:controlCode>
                    <mhp:visibility>VISIBLE_ALL</mhp:visibility>
                    <mhp:serviceBound>false</mhp:serviceBound>
                    <mhp:priority>1</mhp:priority>
                    <mhp:version>00</mhp:version>
                    <mhp:mhpVersion>
                        <mhp:profile>0</mhp:profile>
                        <mhp:versionMajor>1</mhp:versionMajor>
                        <mhp:versionMinor>4</mhp:versionMinor>
                        <mhp:versionMicro>1</mhp:versionMicro>
                    </mhp:mhpVersion>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0001"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0008"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0040"/>
                    <mhp:icon mhp:filename="icons/gen_a/dvb.icon.0200"/>
                    <hbb:MinimumApplicationVersion>0</hbb:MinimumApplicationVersion>
                </mhp:applicationDescriptor>
                <mhp:applicationUsageDescriptor>
                    <mhp:ApplicationUsage>{auto}</mhp:ApplicationUsage>
                </mhp:applicationUsageDescriptor>
                <mhp:applicationBoundary>
                    <mhp:BoundaryExtension>{https}</mhp:BoundaryExtension>
                </mhp:applicationBoundary>
                <mhp:applicationTransport xsi:type="mhp:OCTransportType">
                    <mhp:ComponentTag mhp:ComponentTag="C8" />
                </mhp:applicationTransport>
                <mhp:applicationLocation>opapps/gen_a.pkg</mhp:applicationLocation>
            </mhp:Application>
        </mhp:ApplicationList>
    </mhp:ApplicationDiscovery>
</mhp:ServiceDiscovery>
```

The HTML for the stub app with identity "gen_a", which will be in "index.html" inside the app package, is generated by taking the following and replacing "https://hbbtv1.test" with the URL of the HTTPS server:

```
<!DOCTYPE html>
<html>
    <meta charset="utf-8">
    <head>
        <title>Stub OpApp GEN_A</title>
        <script>
            (function () {
```

```
                var target, elem;
                // HTTPS server location, from bilateral agreement
                // Must be customized.
                var origin = "https://hbbtv1.test";

                // Identification of this stub application
                var stubName = "gen_a";

                // Get URL to the startup Javascript, ensuring that the
                // browser does not try to cache it.
                var cacheBuster = Date.now() + "_" + Math.random();
                var url = (origin + "/StubOpApps/startup/" + stubName +
                    ".js?cacheBuster=" + cacheBuster);

                // Load the script.  Defer execution so it runs after
                // parsing the document but before the DOMContentLoaded and
                // onload events.
                // (Do not use document.write because of
                // https://www.chromestatus.com/feature/5718547946799104 )
                elem = document.createElement("script");
                elem.async = false;
                elem.defer = true;
                elem.src = url;
                target = document.getElementsByTagName('script')[0];
                target.parentNode.insertBefore(elem, target);
            })();
        </script>
    </head>
    <body>
        <div id="stub-loading">Stub application GEN_A loading...</div>
    </body>
</html>
```

D.3.3.5 Alternate entry point

For stub OpApp “gen_b” only, there is an additional HTML file inside the app package at “alternate/entry_point.html”. This is similar to the main HTML page, and can be configured by specifying a stub OpApp with identity “gen_b” and the @alt=true attribute in the XML, like this:

```
<stubOpApp ident="gen_b" target="started_normally.html5" mode="html"/>
<stubOpApp ident="gen_b"target="alternate_entry_point.html5" mode="html" alt="true"/>
```

This allows tests to use the “hbbtv-package://2.71/alternate/entry_point.html” URI syntax to launch the app, and check that the app launches and the alternate entry point is used.

If not using this feature, there is no need to configure it (even if you are using OpApp “gen_b” in the normal way).

D.3.4 App ID ranges

A range of OpApp application IDs needs to be reserved for use by the test harness, e.g. for the launcher OpApp.

The launcher OpApp shall have application ID 100 (decimal)

Values 101-199 inclusive are reserved for use by the test harness for other purposes.

The rest of the application IDs may be used by test cases.

D.3.5 Testcase XML extensions

D.3.5.1 Operator Application Optional Features

There are some additional features specific to testing Operator Applications as well as those defined in 6.3.3.2 Optional Features. The master list of available feature strings is on the “OptionalFeatures” wiki page [42].

D.3.5.2 Extended settings information

The previous content of this section has been moved to 6.3.4.2 Extended settings information.

D.3.6 Implementation XML extensions

There are several extensions to the implementation.xml file syntax.

D.3.6.1 Launcher App

As discussed above, this tag indicates an OpApp test that uses the launcher app, and indicates the initial page to launch:

```
<runOpAppPage path="index.html5"/>
```

The path is relative to the test case XML file. It must refer to a HTML or XHTML page, or to a PHP script that generates a HTML or XHTML page. This must be a path that exists. Note it is a path, not a URL, so URL parameters & fragment references are not allowed. The specified page will be loaded from the harness's HTTPS server.

If you use the <runOpAppPage> tag then you must not specify an <opAppDiscovery> block in the implementation.xml file.

D.3.6.2 Stub OpApps

If the test uses stub OpApp(s) then it must include configuration for each stub OpApp it uses. For each stub OpApp the implementation.xml shall have a tag like this:

```
<stubOpApp ident="gen_a" target="index.html5" mode="html"/>
```

For certain test cases it is necessary to intercept requests to download the OpApp package file to enable special handling. Those test cases specify an extra attribute:

```
<stubOpApp ident="gen_a" target="index.html5" mode="html"
           interceptDownload="get_package.php"/>
```

"ident" identifies which stub OpApp is used, and must be unique in a test case. Valid values are listed in the table in section 0.

"mode" configures what the OpApp does when it's loaded. In "html" mode, it replaces the HTML page with the one specified, e.g. using "window.location=…". In "html-visible" mode the stub OpApp will replace the HTML page but tries to make the OpApp visible first as described in D.3.3.3 Stub OpApp behaviour. In "script" mode, it loads the specified JavaScript script into the stub OpApp's initial HTML page.

The "target" is a path relative to the test case XML file. If "mode" is "html", then "target" must refer to a HTML or XHTML page, or to a PHP script that generates a HTML or XHTML page. If "mode" is "javascript", then "target" must refer to a Javascript script, or to a PHP script that generates a Javascript script. This must be a path that exists. Note it is a path, not a URL, so URL parameters & fragment references are not allowed. The specified page or script will be loaded from the harness's HTTPS server.

If "target" ends ".html", ".html5" or ".cehtml" then "mode" must be "html" and may be omitted. If "target" ends ".js" then "mode" must be "script" and may be omitted. If "target" ends ".php" then "mode" must be specified explicitly.

If interceptDownload is used, then the harness shall configure the HTTPS server so that when the stub OpApp package is requested from https://***<server name>***/StubOpApps/packages/***<name>***.pkg, the HTTPS server shall do an internal redirect (i.e. not a HTTP redirect, so not visible to the HTTP client) and return the results of the PHP script. This allows the test case to control the package returned, including simulating errors and capturing details of the request. The value specified for the interceptDownload attribute must be a relative URI. It is relative to the location of the testcase XML file. It may include query strings but not fragment identifiers. With the query string (if any) removed, it must specify a file that actually exists in the test suite. The harness shall add an extra parameter to the URL, realPackageFilePath, which shall be the path on disk that PHP can use to read the actual package file. E.g. if test com.example_123 has interceptDownload set to "get_package.php" then the URL redirected to might be "/_TESTSUITE/TESTS/get_package.php?realPackageFilePath=/tmp/test_harness/stub_opapps/gen_a.pkg". (Note that you cannot just specify the URL of a PHP script in the XML AIT because that would cause the package to fail the XML AIT check). The harness shall add an extra parameter to the URL,

packageContentType, which shall be the content type of the package to be served, e.g., application/vnd.hbbtv.opapp.pkg.

The <stubOpApp> tag must not be used if you use <runOpAppPage>

This XML tag goes inside the <opAppDiscovery> block in the implementation.xml file. Tests that use stub apps must also specify the following tags: <opAppDnsDiscovery>, <opAppAitDiscovery>, and <cicamOperatorProfile>.

NOTE: This requirement ensures test case authors explicitly specify discovery for their stub apps, or explicitly choose to turn off the discovery features.

D.3.6.3 DNS SRV configuration

The harness will support DNS SRV lookup on a configurable address, to support discovery modes 1n, 1b, 2 and 4.

There are XML tag(s) for configuring the DNS server to respond to a SRV request. Many test cases will find this sufficient:

```
<opAppDnsDiscovery enabled="auto"/>
```

The full syntax is:

```
<opAppDnsDiscovery enabled="true">
    <dnsRecord fqdn="!default">
        <aitServer/>
    </dnsRecord>
    <dnsRecord fqdn="somefqdn.example.org">
        <aitServer host="a.example.com" port="443"
                   priority="0" weight="1"/>
        <aitServer host="a.example.org" port="8888"
                   priority="2" weight="1"/>
    </dnsRecord>
    <dnsRecord fqdn="someotherfqdn.example.org">
        <aitServer host="b.example.com"/>
    </dnsRecord>
</opAppDnsDiscovery>
```

The attributes are:

- “enabled” controls whether non-empty DNS SRV responses are generated at all, it can be “true”, “false” or “auto”. “auto” means it will be enabled if the harness is configured to use default discovery mode 1n, 1b, 2 or 4, i.e. the discovery modes that need DNS response, and disabled if the default discovery mode is set to something else. If “false”, no <dnsRecord> tags may be specified.
- "fqdn” is the domain name that the terminal will look up. The harness’s DNS server will respond to SRV requests for “_hbbtv-ait._tcp.<fqdn>”. For international domain names, this must be specified in Punycode-encoded format. There are some special values allowed:
    - Specify the special value “!hardcoded” to use the first FQDN for discovery mode 2 as configured in the harness. It is an error if the terminal does not support discovery mode 2.
    - Specify the special value “!hardcoded-optional” to use the first FQDN for discovery mode 2 as configured in the harness. If the terminal does not support discovery mode 2 then this <dnsRecord> is ignored.
    - Specify the special value “!hardcoded-2” to use the second FQDN for discovery mode 2 as configured in the harness. It is an error if the terminal does not support discovery mode 2 or does not have a second FQDN.
    - Specify the special value “!hardcoded-2-optional” to use the second FQDN for discovery mode 2 as configured in the harness. If the terminal does not support discovery mode 2 or does not have a second FQDN then this <dnsRecord> is ignored.

- Specify the special values “!nit” or “!nit-2” to use the same values that the harness will use by default when generating the URI_linkage_descriptor in the NIT. It is legal to specify this whatever discovery mode is actually being used.
- Specify the special values “!bat” or “!bat-2” to use the same values that the harness will use by default when generating the URI_linkage_descriptor in the BAT. It is legal to specify this whatever discovery mode is actually being used.
- Specify the special value “!default” to use a value that depends on the configured default discovery mode, for mode 1n this is the same as “!nit”, for mode 1b this is the same as “!bat”, for mode 2 this is the same as “!hardcoded”, and for mode 4 this is the same as “hbbtvopapps.org” (as specified in the HbbTV OpApps standard). For other configured default discovery modes this <dnsRecord> entry will be ignored.
- Specify the special value “!default-2” to use a value that depends on the configured default discovery mode, for mode 1n this is the same as “!nit-2”, for mode 1b this is the same as “!bat-2”, for mode 2 this is the same as “!hardcoded-2”, and for mode 4 this is an error (since discovering multiple OpApps is not possible in that mode). For other configured default discovery modes this <dnsRecord> entry will be ignored.

The default value for this attribute is “!default”. It is an error to specify multiple dnsRecord entries with the same FQDNs (after resolving the special values into the actual FQDNs).

- “host” is the HTTPS server’s DNS name to send in the DNS SRV response. Defaults to the DNS name of the configured TLS server.
- “port” is the HTTPS server’s port number to send in the DNS SRV response. Can be a numeric port number, or “!default”, “!default-a”, “!default-b”, “!default-c”, “!default-d” or “!default-e” to respond with one of the port numbers configured for the TLS server. “!default” means the same as “!default-a”. Defaults to the port number “A” of the configured TLS server.
- “priority” is the numeric priority for this AIT server to send in the SRV response. Lower numbers should be tried first by the terminal. Range is 0-65535. Default is 1.
- “weight” is the numeric weight for this AIT server to send in the SRV response. When comparing results of the same priority, servers with a higher weight should be more likely to be tried first by the terminal. Range is 0-65535. Default is 1.

Each <dnsRecord> tag specifies a single FQDN to respond to DNS SRV requests for.

Each <aitServer> tag specifies a host to respond with in the DNS SRV response. Zero or more hosts are allowed.

If no <dnsRecord> tags are specified, then a default configuration is used which is the same as:

```
<dnsRecord>
    <aitServer/>
</dnsRecord>
```

The harness will always respond to DNS SRV requests for the FQDN hbbtvopapps.org, for the FQDN(s) for mode 2 as configured in the harness (if any), for the FQDN that the harness will use by default when generating the URI_linkage_descriptor in the NIT,, and for the FQDNs that the harness will use by default when generating the URI_linkage_descriptor in the BAT. For each of those FQDNs, if no dnsRecord entry is specified for it or if DNS SRV responses are disabled via the “enabled” attribute then the harness will respond with an empty SRV response.

This XML tag goes inside the <opAppDiscovery> block in the implementation.xml file.

This XML tag must not be used if you use <runOpAppPage>, in that case the harness will configure the DNS server automatically.

D.3.6.4 XML AIT serving

For discovery modes 1n, 1b, 2 or 4, the AIT is requested using the path "/opapp.aitx" on the harness's HTTPS server. For discovery mode 3, the AIT is requested using an operator-defined path on the harness's HTTPS server.

To support these modes, it shall be possible to configure the HTTPS server to respond to these AIT requests by either:

- Returning the contents of a specified XML AIT file, with certain fields optionally filled in by the harness. Or
- do an internal redirect (i.e. not a HTTP redirect, so not visible to the HTTP client) to a PHP script and return the results of that script. This allows the test case to control the AIT returned, including generating dynamic AIT using PHP, and to capture details of the request using a PHP script.

To do this, two XML tags are defined. Most test cases will be able to use this simple tag:

```
<opAppAitDiscovery enabled="auto"
    target="my_opapp_ait.aitx"/>
```

Some test cases will need to specify more parameters:

```
<opAppAitDiscovery enabled="true"
    from="auto"
    target="generate_ait.php"/>
```

Some test case(s) will need to use this more complex form, to specify multiple different redirects:

```
<opAppAitDiscoveryCustom enabled="true">
    <redirect
        from="dns-a"
        target="generate_ait.php?source=a"/>
    <redirect
        from="dns-b"
        target="generate_ait.php?source=b"/>
    <redirect
        from="dns-c"
        target="ait3.aitx"/>
    <redirect
        from="hardcoded"
        target="generate_ait.php?source=hardcoded"/>
</opAppAitDiscoveryCustom>
```

The attributes are:

- "enabled" is required, and controls whether AIT redirection happens at all, it can be "true", "false" or "auto". "auto" means it will be enabled if the harness is configured to use default discovery mode 1n, 1b, 2, 3, 4 or 5 i.e. the discovery modes that need AIT redirection, and disabled if the default discovery mode is set to something else. If "false", none of the other attributes may be specified. <opAppAitDiscoveryCustom enabled=false> is not allowed.
- "from" is optional on <opAppAitDiscovery> and mandatory on <redirect>. It can be "hardcoded", "hardcoded-2", "dns", "dns-a", "dns-b", "dns-c", "dns-d", "dns-e", "cicam", "cicam-2", "auto" or "auto-2". If not specified on <opAppAitDiscovery> it defaults to "auto". It sets which URL is redirected. For discovery mode 3, it should be "hardcoded" or "hardcoded-2" to redirect from the first or second URL (respectively) that is hardcoded in the terminal and configured in the harness, on the TLS server on port A. For modes 1n, 1b, 2 or 4 it should be one of "dns", "dns-a", "dns-b", "dns-c", "dns-d" or "dns-e" to redirect from the standard URI suffix "/opapp.aitx" on the TLS server on the specified port. ("dns" is an alias for "dns-a", provided for convenience since most tests will only use TLS server port A). For discovery mode 5 it should be "cicam" or "cicam-2" to redirect from the first or second AIT URLs used by the CICAM. If set to "auto" then it uses the value appropriate for the default discovery mode configured in the harness, either "hardcoded", "dns-a" or "cicam", and it is an error if that default discovery mode is not 1n, 1b, 2, 3, 4 or 5. If set to "auto-2" then it uses the second value appropriate for the default discovery mode configured in the harness, either "hardcoded-2", "dns-b" or "cicam-2", and it is an error if that default discovery mode is not 1n, 1b, 2, 3 or 5.
- "target" is mandatory and must be a relative URI. It is relative to the location of the testcase XML file. It may include query strings but not fragment identifiers. With the query string (if any) removed, it

must specify a file that actually exists in the test suite, and that file must have a file extension of ".aitx", ".xml" or ".php". If it specifies a ".aitx" or ".xml" file then it must not include a query string.

If the target is an ".aitx" or ".xml" file then it must be well-formed XML. It will be served with the XML AIT Content-Type, with the following changes made automatically:

- If the value of an <orgId> element is "0" then it will be set to the configured organisation_id (in decimal)
- If the value of an <orgId> element is "0" and the corresponding <appId> element is "16" then the <appId> value will be set to the configured application_id (in decimal) for the stub app var_app_id (see D.2.2 Discovery settings).
- If the value of an <ApplicationUsage> element is "{auto}" then it will be set to the configured default OpApp kind
- If the value of an <URLBase> or <BoundaryExtension> element contains the string "{https}" then "{https}" will be replaced with the configured URL of the HTTPS server without a trailing slash, for example "https://hbbtv1.test" or "https://operator.example.com:1234".

The above changes are made without making any other changes to the XML document, not even whitespace changes. The harness shall not do XML schema validation. The harness shall make the above changes whether the specified elements have XML namespace prefixes or not, regardless of the namespace the element is in.

If the target is a ".php" file then the harness shall do an internal redirect to that PHP script, with the following parameters in the query string (or added to the end of any query string specified in the "target" attribute):

- "orgId" – the configured organisation_id, in decimal.
- "varAppID" – the application_id configured, in decimal, for the stub app var_app_id (see D.2.2 Discovery settings).
- "ApplicationUsage" – the configured default OpApp kind, either "urn:hbbtv:opapp:privileged:2017" or "urn:hbbtv:opapp:opspecific:2017", URL encoded.
- "https" – the URL of the HTTPS server without a trailing slash, for example "https://hbbtv1.test" or "https://operator.example.com:1234", URL encoded.

These XML tags go inside the <opAppDiscovery> block in the implementation.xml file.

These XML tags must not be used if you use <runOpAppPage>, in that case the harness will configure redirection automatically.

D.3.6.5 CICAM configuration

New tags are defined so that test cases can configure the XML AIT URL signalled in the CICAM NIT:

<cicamOperatorProfile enabled="auto"/> The attributes are:

- "enabled" controls whether CICAM NIT is used at all, it can be "true", "false" or "auto". "auto" means it will be enabled if the harness is configured to use default discovery mode 5, i.e. the discovery modes that uses the CICAM, and disabled if the default discovery mode is set to something else. If "false", the other attribute must be omitted.
- "numOpApps" is the number of OpApps to signal (i.e. the number of AITs to signal). Valid values are 1 (the default) or 2.

The harness will use a built-in CICAM NIT, which will list all the services from the standard HbbTV base stream. The harness will configure the AIT URL(s) to point at the configured HTTPS server, with harness-chosen AIT path(s). The test case may use the XML AIT serving mechanism from the previous section of this appendix to configure the contents of the AIT(s).

This XML tag goes inside the <opAppDiscovery> block in the implementation.xml file.

This XML tag must not be used if you use <runOpAppPage>, in that case the harness will configure the CICAM NIT automatically if it is necessary.

D.3.6.6 HTTPS server configuration

A tag is defined that configures the harness HTTPS server. The tag is:

```
<httpsServerConfig mode="host-name-mismatch"/>
```

or:

```
<httpsServerConfig mode="good" clientCertForOpAppAit="true" clientCertForOpAppPackage="true"/>
```

The “mode” attribute allows the harness to be configured with special invalid certificates for testing. The valid modes are: “good” (the default), “host-name-mismatch”, “self-signed”, “ip-address-mismatch”, “expired-certificate” or “revoked-certificate”. See section 0 for details of the certificates.

For “host-name-mismatch”, “self-signed”, “expired-certificate” or “revoked-certificate” the HTTPS server domain name used shall be as configured for that certificate bundle. For “ip-address-mismatch” all references to the HTTPS server “domain name” shall actually refer to the IPv4 address of the server in dotted quad format.

When the clientCertForOpAppAit attribute is specified as true, then the OpApp AIT URL(s) shall request but not require a client certificate as described in section D.5. For the purposes of this paragraph, the OpApp AIT URL(s) for a particular test case are the URLs that are used as "from" URLs on an <opAppAitDiscoveryCustom> or <opAppAitDiscoveryCustom><redirect> tag in that test case. When this flag is true, a test harness may choose to request client certificates for all possible OpApp AIT URLs, even URLs not used by that particular test case, if that make it easier for the test harness implementer.

When the clientCertForOpAppPackage attribute is specified as true, then the stub OpApp package URLs shall request but not require a client certificate as described in section D.5. For the purposes of this paragraph, the stub OpApp package URLs for a test case are the URLs of the form https://<server name>/StubOpApps/packages/<name>.pkg, as described in section D.3.3.2 Stub OpApp package locations, for the stub OpApps that are configured using <stubOpApp> tags in that particular test case. When this flag is true, a test harness may choose to request client certificates for all possible OpApp package URLs, even URLs not used by that particular test case, if that make it easier for the test harness implementer.

When this tag is used and specifies any non-default values, TLS server port A is configured as specified, and test cases shall not use the other ports. The configuration of TLS on those other ports (or even whether there’s a server running there at all) is deliberately not specified.

This XML tag goes inside the <opAppDiscovery> block in the implementation.xml file.

This XML tag must not be used if you use <runOpAppPage>.

D.3.6.7 Minimum timeout

The content of this section has been moved to 7.4.1.1 Minimum timeout.

D.3.6.8 Examples (informative)

This is a complete <opAppDiscovery> block for a test with a single OpApp that uses the default discovery mode and supports all discovery modes:

```
<opAppDiscovery>
    <stubOpApp ident="gen_a" mode="html"
        target="index_opApp.html5"/>
    <opAppDnsDiscovery enabled="auto"/>
    <opAppAitDiscovery enabled="auto"
        target="gen_a_https.aitx"/>
    <cicamOperatorProfile enabled="auto"/>
</opAppDiscovery>
```

This is a complete <opAppDiscovery> block for a test with two OpApps that uses the default discovery mode and supports all discovery modes except mode 4:

```
<opAppDiscovery>
    <stubOpApp ident="gen_a" mode="html"
        target="index_opApp_a.html5"/>
    <stubOpApp ident="gen_b" mode="html"
        target="index_opApp_b.html5"/>
```

```
        <opAppDnsDiscovery enabled="auto">
            <dnsRecord fqdn="!default">
                <aitServer port="default-a">
            </dnsRecord>
            <dnsRecord fqdn="!default-2">
                <aitServer port="default-b">
            </dnsRecord>
        </opAppDnsDiscovery>
        <opAppAitDiscoveryCustom enabled="auto">
            <redirect from="auto" target="gen_a_https.aitx"/>
            <redirect from="auto-2" target="gen_b_https.aitx"/>
        </opAppAitDiscoveryCustom>
        <cicamOperatorProfile enabled="auto" numOpApps="2"/>
    </opAppDiscovery>
```

(Note that a terminal that only supports discovery mode 4 can only discover a single OpApp).

D.4 DVB TS Generation extensions

D.4.1 Playoutset extensions

Note: It is assumed that the tester has configured the harness with the preferred OpApp discovery mechanism from the bilateral agreement. Where the below talks about "if using an OpApp discovery mechanism that …", that refers to the OpApp discovery mechanism configured there. This feature is intended to allow certain OpApp discovery tests to be written in a way that allows them to run on all terminals without having to write 8 variants of each test case for the 8 different discovery methods specified in the OpApps specification [37].

There are extensions to the playoutset XML schema to:

- Include an XML syntax to indicate that a BAT should be conditionally included, i.e. only include it if using an OpApp discovery mechanism that uses it, or only include it if the terminal supports an OpApp discovery mechanism that uses it. If included, the contents of the BAT are specified by a separate XML file. This is the "enabled" attribute on the <bat> element, which can be "true" (default) to unconditionally include the BAT or "auto" to include it if the selected default discovery mode uses it, or "if-possible" to include it if any supported discovery mode uses it.

D.4.2 NIT extensions

There is an extension to the NIT XML schema to allow the URI_linkage_descriptor to be generated. This is the <hbbtvUriLinkageDescriptor> element.

In the uri attribute of the <hbbtvUriLinkageDescriptor> element the following substitutions are made:

- "{nid}" is replaced by the network ID, as a lowercase hex number with no leading zeroes. This is the Network ID configured in the harness if running OpApp tests or the standard Network ID used for the base stream if running regular HbbTV tests.
- "{opapp-fqdn}" is replaced by the harness-chosen default FQDN as used for discovery mode 1n. (Note This is NOT the FQDN configured in the harness by the tester for discovery mode 2).
- "{opapp-fqdn-2}" is replaced by the second harness-chosen default FQDN as used for discovery mode 1n. (Note This is NOT the FQDN configured in the harness by the tester for discovery mode 2).

If the <hbbtvUriLinkageDescriptor> element has enabled="auto" then the URI_linkage_descriptor will only be included if the selected default OpApp discovery mechanism uses it , as follows:

- If "enabled" is "auto" and the "uri" attribute starts "dns:", then the URI_linkage_descriptor will only be included if the selected default OpApp discovery mechanism is 1n.
- If "enabled" is "auto" and the "uri" attribute starts "dvb:", then the URI_linkage_descriptor will only be included if the selected default OpApp discovery mechanism is 6n.
- If "enabled" is "auto" and the "uri" attribute does not start "dns:" or "dvb:", then the NIT XML file is invalid.

Launcher OpApp tests should not specify an explicit URI_linkage_descriptor in the broadcast NIT or BAT.

Here is an example NIT:

```
<?xml version="1.0" encoding="utf-8"?>
<nit xmlns="http://www.hbbtv.org/2016/nit" version="0">
```

```
    <network>
        <networkNameDescriptor>HBBTV_A</networkNameDescriptor>
        <hbbtvUriLinkageDescriptor enabled="auto" uri="dns:{opapp-fqdn}"/>
        <hbbtvUriLinkageDescriptor enabled="auto" uri="dvb://63.1.b"/>
    </network>
    <transportStream onid="99" tsid="1">
        <autoDeliverySystemDescriptor />
        <serviceListDescriptor>
            <service sid="10" type="mpeg2-sd-tv"/>
            <service sid="11" type="mpeg2-sd-tv"/>
            <service sid="12" type="mpeg2-sd-tv"/>
            <service sid="13" type="mpeg2-sd-tv"/>
            <service sid="14" type="radio"/>
        </serviceListDescriptor>
        <privateDataSpecifierDescriptor value="40"/>
    </transportStream>
</nit>
```

D.4.3 BAT generation

For the BAT, interpretation of <hbbtvUriLinkageDescriptor> with enabled=”auto” is the same as the NIT except the relevant discovery modes are 1b and 6b (instead of 1n and 6n).

For the BAT, in the <hbbtvUriLinkageDescriptor>, “{opapp-fqdn}” and “{opapp-fqdn-2}” shall expand to the harness-chosen default FQDNs as used for discovery mode 1b, which will be different from the value it expands to in the NIT.

Here is an example BAT for an OpApps test:

```
<?xml version="1.0" encoding="utf-8"?>
<bat xmlns="http://www.hbbtv.org/2016/nit" version="0">
    <bouquet>
        <bouquetNameDescriptor>HBBTV_A</bouquetNameDescriptor>
        <hbbtvUriLinkageDescriptor enabled="auto" uri="dns:{opapp-fqdn}"/>
        <hbbtvUriLinkageDescriptor enabled="auto" uri="dvb://63.1.b"/>
    </bouquet>
    <transportStream onid="99" tsid="1">
        <autoDeliverySystemDescriptor />
        <serviceListDescriptor>
            <service sid="10" type="mpeg2-sd-tv"/>
            <service sid="11" type="mpeg2-sd-tv"/>
            <service sid="12" type="mpeg2-sd-tv"/>
            <service sid="13" type="mpeg2-sd-tv"/>
            <service sid="14" type="radio"/>
        </serviceListDescriptor>
        <privateDataSpecifierDescriptor value="40"/>
    </transportStream>
</bat>
```

D.4.4 AIT extensions

D.4.4.1 Organization ID

If orgId is specified as 0 in the AIT XML file, then the harness shall use the tester-selected organisation_id (as specified in the bilateral agreement).

Note: this does not require an XML schema change, the existing XML AIT schema allows 0 even though that value is invalid according to the specification.

D.4.4.2 Application ID

If orgId is specified as 0 and appId is specified as 16 in the AIT XML file, then the harness shall use the tester-selected application_id for the stub app var_app_id (as specified in the bilateral agreement). See D.2.2 Discovery settings.

D.4.4.3 Application Usage

Three additional values are supported for the existing XML <ApplicationUsage> tag:

- “urn:hbbtv:opapp:privileged:2017” causes the application usage to be signalled as 0x80

- “urn:hbbtv:opapp:opspecific:2017” causes the application usage to be signalled as 0x81
- “{auto}” means that the harness shall use the OpApp kind selected in the harness configuration.

Note: this does not require an XML schema change, the existing XML AIT schema specifies a string here, this is just defining 3 additional values.

D.4.4.4 Void

The previous content of this section has been moved to 7.4.4.3.1 AIT.

D.4.4.5 Application overlay descriptor

The harness shall generate the application_overlay_descriptor as defined in section 7.2.1 of the OpApps specification, if the XML AIT specifies the <SuppressOverlays> tag as defined in testingAit.xsd. To do this, the root element of the XML AIT file needs to be <testait:ServiceDiscovery>.

In addition, it is possible to generate a DVB AIT that does not list any applications. This will be necessary when testing this descriptor. If the root element of the XML AIT file is <testait:ServiceDiscovery> then the ApplicationList is allowed to be empty.

If the value of an < testait:orgId> element is “0” then the harness shall use the tester-selected organisation_id (as specified in the bilateral agreement).

D.4.4.6 Application version descriptor

The harness shall generate the application_version_descriptor as defined in section 7.2.2 of the OpApps specification, if the XML AIT specifies the <MinimumApplicationVersion> as defined in section 7.3.1 of the OpApps specification.

Note that the interpretation of the version field in the XML AIT was modified by errata to the OpApp specification. See HbbTV OpApp Spec Redmine #9749. For TS generation, test cases and test harnesses shall use the new interpretation.

Note that this means that whenever an OpApp is signalled by a broadcast AIT, then the test case's XML AIT used to generate the broadcast AIT must include the <MinimumApplicationVersion> tag to comply with the OpApp specification. If you have no particular minimum application version to signal, then the <MinimumApplicationVersion> tag should specify the same version as the application version.

D.4.4.7 Conditional Include of applications in AIT

There is a way to indicate that an application should only be included in the AIT if using an OpApp discovery mechanism that uses the AIT. The <testait:Application> element can be marked with the testait:enabled=”auto” attribute to do this.

D.4.4.8 Void

The previous content of this section has been moved to 7.4.4.3.1 AIT.

D.4.4.9 Injecting error in AIT

There is a way to indicate that a malformed AIT should be generated, with the transport_protocol_descriptor for application(s) placed in the common loop. To enable this the <testait:Application> element can set the injectError=”transportProtocolDescriptorInCommonLoop” attribute.

D.4.5 OpApp Launcher Stream

A test harness may play a stream in between OpApp Launcher-based test cases. The OpApp Launcher Stream differs with the Base Test Stream described at 5.2.3 Base Test Stream, as it may use version number 6 for the PID 200 (PMT for service 11) and/or PID 16 (NIT), and it signals the launcher OpApp if such signalling is required. OpApp test cases shall therefore avoid using a version number of 6 for their PID 200 (PMT for service 11) and PID 16 (NIT).

D.5 TLS client certificate extensions

The test harness's test TLS server shall always request (but not require) a client certificate to access URLs matching this pattern:

```
https://[^/]+/_TESTSUITE/TESTS/[^/]+/TLSClientCertRequired/.*
```

Test cases that use such URLs must contain appropriate preconditions so that they are only run if the bilateral agreement requires the terminal to have a client certificate.

If the terminal sends a client certificate, then the server shall automatically verify that the certificate is valid using the usual TLS rules (e.g. certificate chains back to a known client CA, client and any intermediate certificates are not revoked)

The test harness's test TLS server shall make details of the client certificate available to PHP via environment variables. The set of environment variables to be made available are:

`HTTPS`	If set, HTTPS is being used.
`SSL_CLIENT_S_DN_CN`	The CN= part of the subject of the client certificate
`SSL_CLIENT_S_DN_O`	The O= part of the subject of the client certificate
`SSL_CLIENT_CERT`	The client certificate. PEM-encoded
`SSL_CLIENT_CERT_CHAIN_0`, `SSL_CLIENT_CERT_CHAIN_1`, `SSL_CLIENT_CERT_CHAIN_2`, …	The client certificate chain. PEM-encoded. One certificate in each environment variable, as many environment variables as needed.

D.6 Running OpApp and Regular HbbTV apps at the same time

D.6.1 Mode identification

The HbbTVTestAPI constructor now optionally takes a parameter. For tests invoking the HbbTVTestAPI from an OpApp, this parameter shall be a mode string – either "OpApp" for operator apps not started from the Launcher OpApp, or "OpAppPage" for a page started from the Launcher OpApp, or "OpAppReplacedLauncher" for a separate OpApp that was launched from a launcher based test with the createApplication(…, …, true). E.g:

```
var testApi = new HbbTVTestAPI("OpApp");
```

Normal HbbTV apps, and harness-based test code, shall not pass this parameter.

D.6.2 Void

The previous content of this section has been moved to 7.2.1.3 Multiple clients.

D.6.3 Void

The previous content of this section has been moved to 7.2.11 JS-Function endTestApp().

D.6.4 Void

The previous content of this section has been moved to 7.2.12 JS-Functions for multiple-client communication.

D.7 Void

D.7.1 Void

The previous content of this section has been moved to 7.4.8 JS-Function setSignalLevel().

ANNEX E Informative Guidance to Testers

E.1 Introduction

This appendix gives guidance to testers to help them understand and use the information in the current document to execute a programme of tests.

E.2 Is a test case mandatory?

This section gives guidance on how to determine whether a test case is mandatory to run on a particular device under test.

A test case will have a set of specification versions to which it is applicable. See section 6.3.2.1 Test Applicability.

A test case may have preconditions that restrict the devices for which it is mandatory to run the test. The current section gives guidance on understanding and interpreting these preconditions. Section 6.3.3 Preconditions in the current document gives the definition of preconditions.

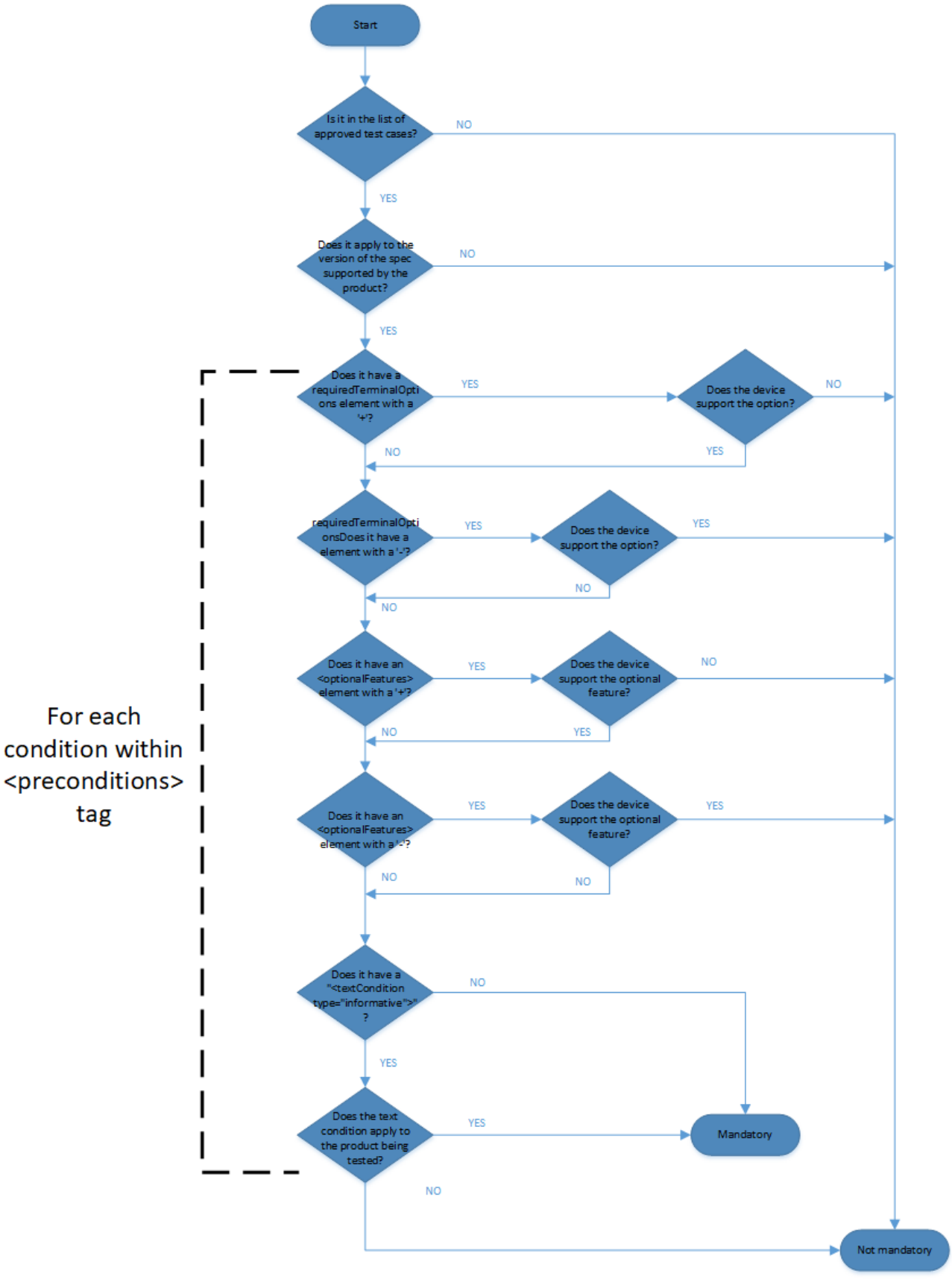
The following describes the decision rules for whether a particular test case is mandatory on a device:

- The test case is mandatory if the <appliesTo> tag *matches* as defined below, AND the <preconditions> tag is *satisfied* as defined below. Otherwise the test case is not mandatory.
- The <appliesTo> tag *matches* if **any** of the listed specifications are supported by the terminal. Note that a terminal can only support one version of the HbbTV specification, since it has to report a single HbbTV version number in its user agent string, but might also support OpApps and/or one or more regional or operator-defined specifications.
- The <preconditions> tag is *satisfied* if **every** XML tag that is directly inside it (i.e. if there are nested tags, only check the outer one here) is *satisfied*, except for any <textCondition type="procedural"> tags and <testRun> tags which shall be ignored.
- A <requiredTerminalOptions> tag is *satisfied* if **every** feature it lists with a "+" sign is supported by the terminal **and every** feature it lists with a "-" sign is **not** supported by the terminal.
- A <optionalFeatures> tag is *satisfied* if **every** feature it lists with a "+" sign is supported by the terminal **and every** feature it lists with a "-" sign is **not** supported by the terminal.
- A <textCondition type="informative"> tag is *satisfied* if the described condition is true.
- An <or> tag is *satisfied* if **at least one** of the XML tags that are directly inside it (i.e. if there are nested tags, only check the outer one here) is *satisfied*. Note that this tag can contain <requiredTerminalOptions>, <optionalFeatures> or <and> tags.
- An <and> tag is *satisfied* if **both** the <requiredTerminalOptions> tag **and** the <optionalFeatures> tag inside the <and> are *satisfied*. Note that this tag can only appear inside an <or> tag.

Note that <textCondition type="procedural"> tags, <testRun> tags and the contents of <adaptsTo> tags are ignored when deciding if a test case is mandatory or not.

- <textCondition type="procedural"> tags are things you should do before running the test case, they do not affect the decision about whether or not to run it.
- <testRun> specifies the order the tests have to be run in, it doesn't affect the decision about whether or not to run a test, and since HbbTV requires test cases to be self contained any use of <testRun> is always a test case bug.
- <adaptsTo> does not affect whether a test is mandatory or not, it identifies that a test behaves differently based on certain capabilities of the terminal.

The following diagram shows a flowchart that can be used to determine if a test case is mandatory to run on a device or not.



ANNEX F Example of JSON returned by TLS Server


```
{
"length": 16,
"client_version": "3.3",
"server version": "3.3",
"supported_versions": ["3.3","3.4"] //supported_versions extension
"random": {
"gmt_unix_time": "A65891BB",
"bytes": "17B760AA285FB652D27D2D6BED0CF8D7657932EA36A6FF46E032B6DE"
},
"session_id": "896C5DD3AFB8230D01919A99A80A38C0BD08454E63B81DD9F2D1D50B00D33161",
"renegotiation_info": {
"renegotiated_connection": ""
},
"handshake_integrity_failure": {
"decrypt_error alert_sent": false
},
"server_name_indication": {
"serverNameList": [
"158kcdz704k0qq24cln0f4p1z5t6qfz4-tls1010.w.hbbtvtest.org"
]
},
"supported_groups": [ "0x0017", "0x0018"], // NamedGroupList from supported_groups extension
,
"application_layer_protocol_negotiation":["h2","http/1.1"], // the order on the list is important
"elliptic_curves": {
"EllipticCurveList": [
"001D",
"0017",
"0018",
"0019"
]
},
"signature_algorithms": {
"supported_signature_algorithms": [
"ecdsa",
"UNKNOWN"
]
},
"signature_algorithms_certificate": {
"supported_signature_algorithms": [
"ecdsa",
"UNKNOWN"
]
},


"compression_methods": [
"00"
],
"cipher_suites_count": 4,
"cipher_suites": {
"1301": { // cipher suite value {0x13,0x01}
"cipher": "UNKNOWN",
"protocol": "",
"keyExchange": "",
```

```
"cryptosystem": "",
"algorithms": "",
"bits": "",
"algorithm": ""
},
"C02B": {
"cipher": "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
"protocol": "",
"keyExchange": "",
"cryptosystem": "",
"algorithms": "",
"bits": "",
"algorithm": ""
},
"C014": {
"cipher": "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA",
"protocol": "",
"keyExchange": "",
"cryptosystem": "",
"algorithms": "",
"bits": "",
"algorithm": ""
},
"00FF": {
"cipher": "TLS_EMPTY_RENEGOTIATION_INFO_SCSV",
"protocol": "",
"keyExchange": "",
"cryptosystem": "",
"algorithms": "",
"bits": "",
"algorithm": ""
}
},
"alert_code": 0,
"jsonOrigin": "script"
}
```