Test Specification for HbbTV Version 1.2.1

Version 1.2
July 10, 2014

Test Specification
For HbbTV Version 1.2.1

Version 1.2
July 10, 2014

Conditions of Publication

CLASSIFICATION

The information contained in this document is public.

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: Ian Medland (HbbTV Testing Group Chair) testing-list@hbbtv.org
Web Site: http://www.hbbtv.org
E-mail: info@hbbtv.org

Table of 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 HbbTV Specification.
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.

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 1-1 are provided by the HbbTV Testing Group.

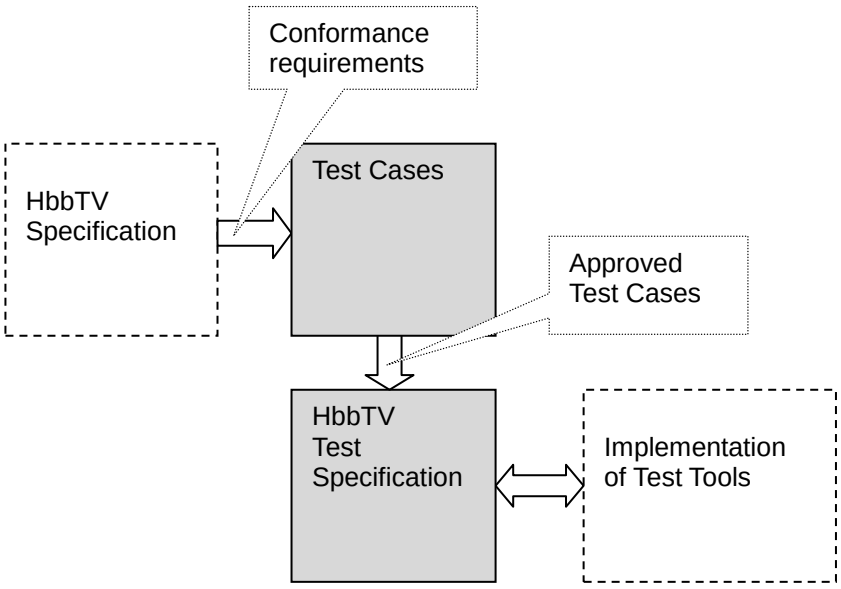

Figure 1-1 HbbTV Test Specification and implementation overview

1.2 Conformance and reference

HbbTV devices shall conform to the applicable parts of the HbbTV Technical Specification Version 1.2.1 [1].

This page is intentionally left blank

2 The terms of use

2.1 Ownership

This document is created and maintained by the HbbTV Test Group, agreed by the impacted work groups and approved by the HbbTV Steering Group.

This page is intentionally left blank

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”; V1.2.1 with errata 1

Available from http://hbbtv.org/pages/about_hbbtv/specification.php

[2] Open IPTV Forum Release 1 specification, volume 5 - “Declarative Application Environment” v1.2
Available from http://www.oipf.tv/specifications

[3] Open IPTV Forum Release 1 specification, volume 2 - “Media Formats” v1.2
Available from http://www.oipf.tv/specifications

[4] ETSI TS 102 809 “Digital Video Broadcasting (DVB); Signalling and carriage of interactive applications and services in hybrid broadcast / broadband environments", V 1.1.1

[5] Open IPTV Forum Release 1 specification, volume 4 - “Protocols” v1.2
Available from http://www.oipf.tv/specifications

[6] Open IPTV Forum Release 1 specification, volume 7 - “Authentication, Content Protection and Service Protection” v1.2
Available from http://www.oipf.tv/specifications

[7] Internet Streaming Media Alliance, “Implementation Specification Version 2.0, April 2005”
Available from http://www.isma.tv

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

[9] RFC2818, IETF: “HTTP over TLS”

[10] RFC5246, IETF: “The Transport Layer Security (TLS) Protocol Version 1.2”

[11] RFC5280, IETF: “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”

[12] ETSI TS 102 851 “Digital Video Broadcasting (DVB): Uniform Resource Identifiers (URI) for DVB Systems”, V1.3.1

[13] W3C, “XMLHTTPRequest” Working Draft 20 August 2009, http://www.w3.org/TR/XMLHttpRequest/

[14] CI Plus Forum, CI Plus Specification, “Content Security Extensions to the Common Interface”, V1.2 (2009-04).

[15] ISO/IEC 14496-3, “Information technology - Coding of audio-visual objects - Part 3: Audio”, 2009

[16] ETSI TS 101 154 “Digital Video Broadcasting (DVB);Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream”, V 1.9.1

[17] ETSI TS 102 366 “Digital Audio Compression (AC-3, Enhanced AC-3) Standard”, V1.2.1

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

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

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

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

[22] ECMAScript Language Specification (Third Edition), December 1999, ECMA-262.pdf, http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf

[23] ‘Document’ Names for specReference "name" attribute. Only accessible to HbbTV Members https://www.hbbtv.de/redmine/projects/hbbtv-ttg/wiki/DocumentNames

[24] Spec Names for appliesTo spec tags. Only accessible by HbbTV members https://www.hbbtv.de/redmine/projects/hbbtv-ttg/wiki/AppliesToSpecNames

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

4 Definitions and abbreviations

4.1 Definitions

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

Item	Definition
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 signaled 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 behavior 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.
Progressive download[2]	A variant of persistent download where playback of the content item can start before the download of the content item has completed

[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]
[2] Progressive download is referred to as playable download in the OIPF DAE specification [2].

Item	**Definition**
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. 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: • This definition does not include any test infrastructure (e.g. web server, DVB-Playout ...) required to execute the test case. • 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 a specific version of the HbbTV Specification, including test cases for all possible features. NOTE: The Testing Group specifies and approves which test cases are parts of the HbbTV Test Suite. Only Approved HbbTV Test Material shall be a part of the HbbTV Test Suite used for the Authorized Purpose. Upon approval of HbbTV, new HbbTV Test Material may be added from time to time at regular time-intervals. 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”

Item	Definition
	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
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
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 word, or the terms "***REQUIRED***" or "***SHALL***", mean that the definition is an absolute requirement of the specification.

MUST NOT This phrase, or the phrase "***SHALL NOT***", mean that the definition is an absolute prohibition of the specification.

SHOULD This word, or the adjective "***RECOMMENDED***", mean that there may exist 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 behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior 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 shall at least comprise of the following elements:

- A Device Under Test (DUT)
- A Test Harness
- An IP connection established between the Test Harness and the DUT
- A DVB Playout device
- A Test/Result Storage element

Figure 5-1 shows the interconnectivity between these required elements.
The DUT is connected to

- A DVB Playout device
- A Test Harness
- An audio surround system

The DVB playout device is also connected to the Test Harness.

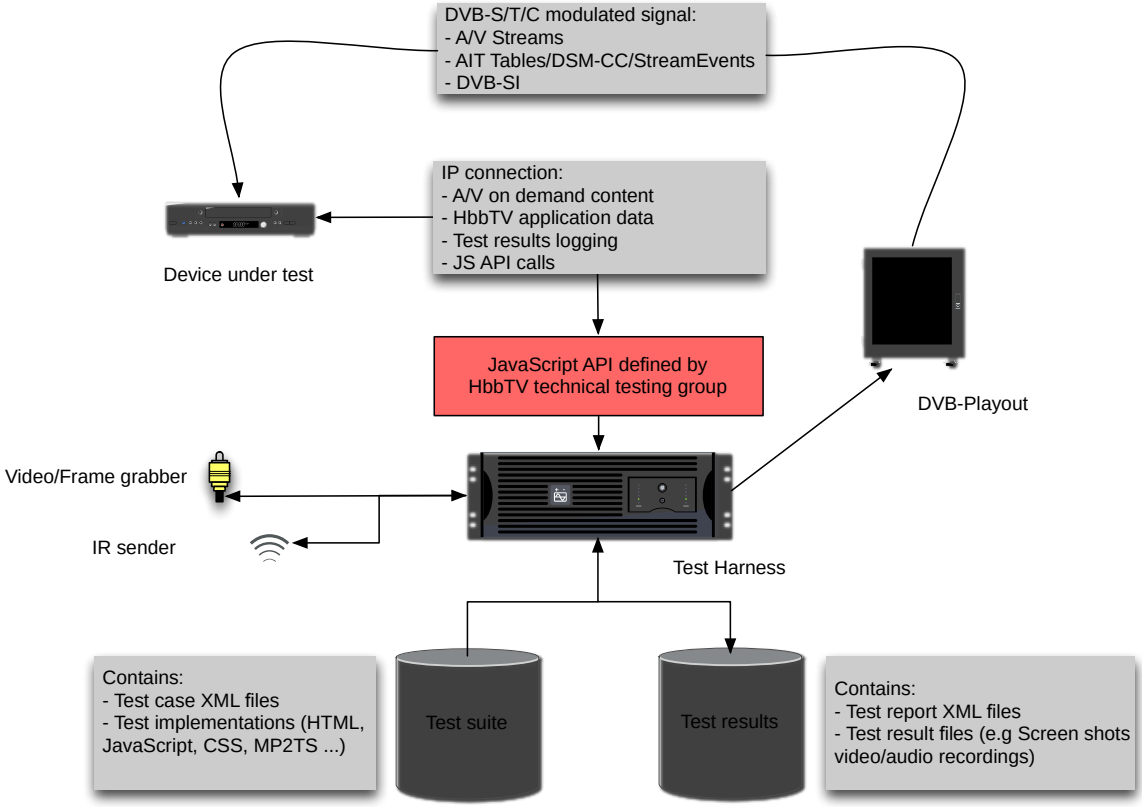

Figure 5-1 Outline of an example test system overview

5.1 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.

In detail this means that the Test Harness 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).

After that the Test Harness initiates the execution of the Test Case on the DUT.

During the runtime of the Test Case, the Test Harness collects the test results and responds to the JavaScript API calls from the Test Case running on the DUT.

5.1.1 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.

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

5.1.1.1 Web Server

The Standard Test Equipment shall include an HTTP web server serving the Test Material to the DUT.

The Web Server shall 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 this URL:
http://hbbtv1.test/_TESTSUITE/TESTS/00000010/index.html

The Web Server shall be equipped with a valid installation of the PHP Hypertext Preprocessor to allow dynamic generation of content. Any version of PHP 5 may be installed on the server, no special modules are required. Only files with the extension .php shall be processed by the PHP Hypertext Preprocessor.

Test authors shall ensure that content should be served with cache control headers when needed. To set headers, the test will need to use a PHP script to serve the content. For example, when serving a dynamic MPD file for DASH streaming, the PHP script should set the "Cache-Control: no-cache" header.

The following file extensions shall be served by the web server with their respective mime type:

- .html: application/vnd.hbbtv.xhtml+xml; 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: video/mp4
- .ts: video/mpeg
- .m4a: audio/mp4
- .mp4a: audio/mp4
- .mp3: audio/mpeg
- .bin: application/octet-stream
- .casd: application/vnd.oipf.contentaccessstreaming+xml
- .mpd: application/dash+xml, charset=UTF-8 [19]
- .xse: application/vnd.dvb.streamevent+xml

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.

5.1.1.1.1 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 [20] of HTTP requests matches the POSIX basic regular expression[3] [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:

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 of the Test Suite. 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 'video/mp4' [21].

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.

5.1.1.2 DNS Server

The Standard Test Equipment shall include a DNS (Domain Name System) server that is used by the DUT (either by manual configuration of the DUT, or by automatic configuration via DHCP).

The DNS server shall resolve the following domain names:

- hbbtv1.test, hbbtv2.test, hbbtv3.test, a.hbbtv1.test, b.hbbtv1.test, c.hbbtv1.test, shall be resolved to the IP address of the web server.
- server-na.hbbtv1.test shall be resolved to an IP address that is not reachable

[3] 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)/(.*)$

5.1.1.3 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 bitrate. This shall be controlled by the Test Harness, either directly or via instructions delivered to the operator. By default, this restriction shall be 8Mbps, but may be changed by the use of the setNetworkBandwidth() (see 7.4.6) function. At the beginning of each test case, the network throughput shall be returned to the default, regardless of any throttling that was applied during the previous test.

Throughput here is defined as the average number of bits per second passed in to the transmitting network interface at the application layer, i.e. the figure excludes all HTTP, TCP/UDP, IP, MAC, LLC, etc. overheads. The restriction only applies to data received by the device under test (i.e. data sent by the test environment). Transmission of data from the device under test shall not be restricted.

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

5.1.1.4 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.

Implementations 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.

NOTE: 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.

5.1.1.5 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.

5.1.1.6 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 signaled by the implementation.xml file for the current test. The application environment shall support ECMAScript version 3 [22] or greater 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.

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 [22]. 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.

5.1.2 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.

5.1.2.1 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 7.3.8. 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 HbbTV 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.

5.1.2.1.1 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.

5.1.3 Base Test Stream

This section describes the structure of the Transport Stream used by all tests as a basis.

5.1.3.1 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.
The table 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.

PID	Contents
0	PAT
16	NIT
17	SDT
18	EIT
20	TDT/TOT
100	PMT for service 10
101	Video
102	Audio
200	PMT for service 11
201	PMT for service 11 (DSM-CC signaled)
205	AIT for service 11*
300	PMT for service 12
305	AIT for service 12*
400	PMT for service 13
405	AIT for service 13*
500	PMT for service 14
505	AIT for service 14*

* 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).

5.1.3.2 Service configuration

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

PAT/SDT signaled 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
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
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
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
Service 14			
	Name	ATE Test14	
	PMT PID	500	
	AIT PID	505	
	Video PID	None (Radio service)	
	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
Non-signaled 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

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 6-1.
.

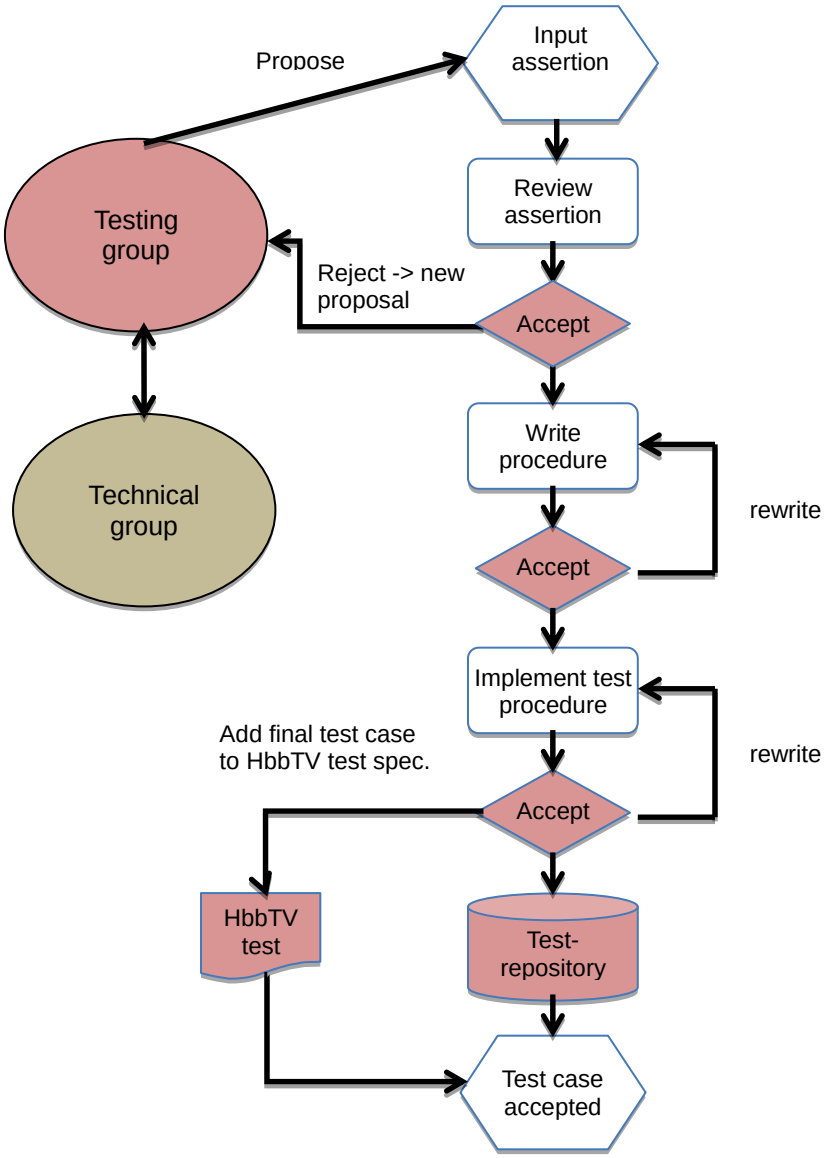

Figure 6-1 Test creation work flow

6.1.1 Details of the process for creating Test Cases

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 of the Test Suite.

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 that test 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

The part of the test case that this organization contributed to. 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

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 what 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" or "1.2.1". Tests that run on both HbbTV 1.1.1 and 1.2.1 will include both "1.1.1" and "1.2.1" tags. 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” [24]. For each spec element in the <appliesTo> tag, the testcase 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="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 of HbbTV and seek to use a particular test case - it is not required to include a spec element for that regime.

Every testcase shall have an appliesTo element with at least one spec element. It must at least include a spec element for HbbTV 1.2.1. It may also include spec elements for other specifications as listed on the “AppliesToSpecNames” Wiki page [24].

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 [23]. 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 textual description of the object under test (specification part or API part). The clause is optional. Where it is included the following values are possible:

- HbbTVClient/AVControlObject
- HbbTVClient/VideoBroadcastObject
- HbbTVClient/ApplicationLifecycle
- HbbTVClient/UserInput
- HbbTVClient/Signalling
- HbbTVClient/DSM-CC
- HbbTVClient/ApplicationManager
- HbbTVClient/Graphics
- HbbTVClient/Security
- HbbTVClient/Browser

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 HbbTV Specification, Version 1.1.1, section 10.2.4; and in HbbTV Specification, Version 1.2.1, 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. 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.

6.3.3.2 Optional Features

Terminal features which are required on the DUT, in addition to required terminal options, to run this test. Available features are:

Feature	Description
+CI_PLUS	Terminal supports CIPlus
+EAC3	Terminal supports E-AC3. This can be decoded and sent over analogue output(s), or sent over a digital output (possibly transcoded) or output over internal loudspeakers
+MULTI_CHANNEL_AUDIO	Terminal supports 5.1 or 7.1 channel audio output

+SPDIF	Terminal provides encoded digital audio via an SPDIF or TOSLink interface
+STEREO	Terminal supports 2.0 channel audio output
+VK_PLAY	Terminal can generate the VK_PLAY event
+VK_PAUSE	Terminal can generate the VK_PAUSE event
+VK_PLAY_PAUSE	Terminal can generate the combined VK_PLAY_PAUSE event

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 EAC3 support.

6.3.3.3 Text Condition

A textual description of a precondition. Whitespace is not significant.

6.3.3.3.1 Informative

This description is optional and considered informational for reviewers or implementers. 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 Testing

6.3.4.1 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:

6.3.4.1.1 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.

6.3.4.1.2 Procedure

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

6.3.4.1.3 Expected behavior

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

6.3.4.2 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.

6.3.4.3 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.

6.3.4.3.1 Name

The file name. This will be the file name on disk. The procedure can use this file name to refer to the media file.

This should just be a file name, not a path. So it shall not contain "/" or "\".

6.3.4.3.2 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.

6.3.4.4 History

Contains a history of additions and modifications to this test.

6.3.4.4.1 Date

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

6.3.4.4.2 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)

6.3.4.4.3 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.

6.3.5 Others

6.3.5.1 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 (eg. 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 HbbTV Test API and Playout Set definition for building Test Cases

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.

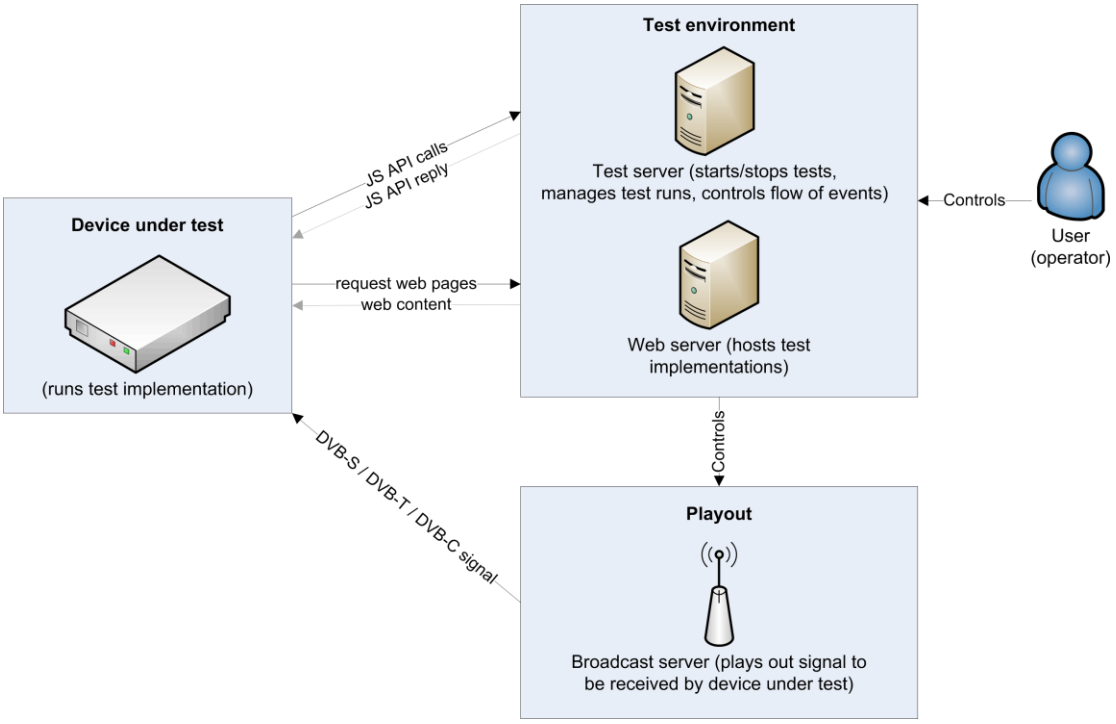

Figure 7-1 Overview of JS API communication between DUT and Test Harness.

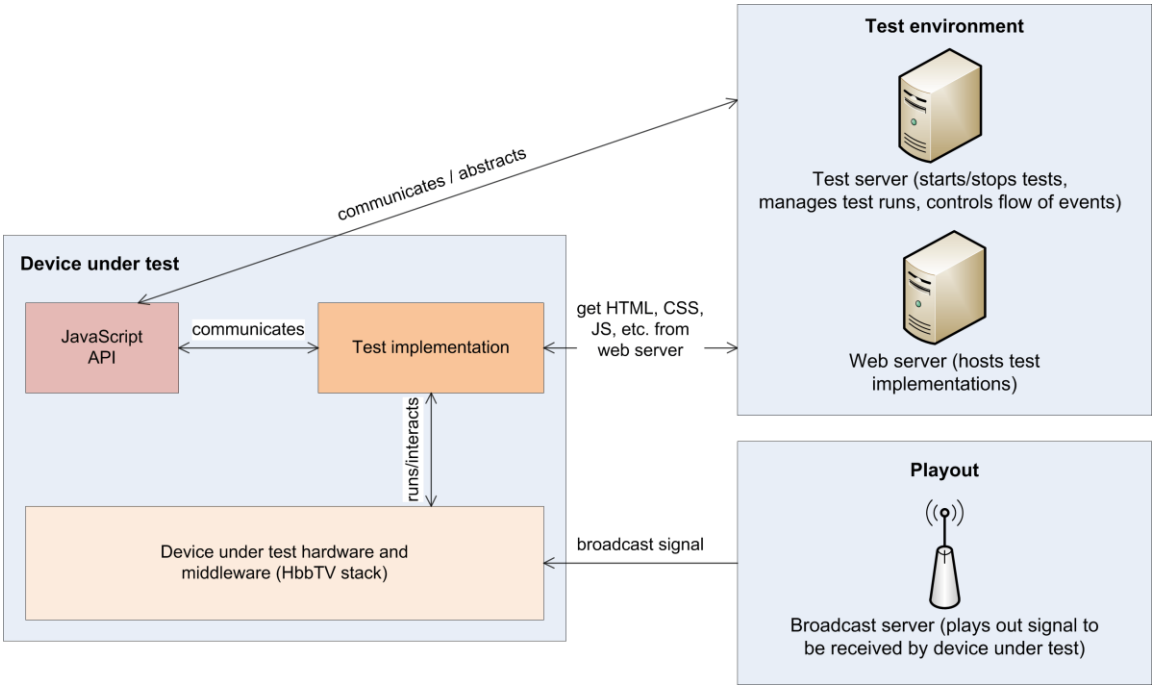

Figure 7-2 Detailed overview on JS-API abstraction of Test Harness communication.

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 three parts:

- APIs Interacting with Test Environment (see section 7.2):
  This part of the APIs communicate with the (automated) test environment. It informs the Test Environment about the current test's status.
- APIs Interacting with the Device Under Test (see section 7.3):
  This part of the APIs communicate 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).
- APIs Interacting with the Playout Environment (see section 7.4):
  This part of the APIs communicate 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.

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 must fail the test.

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

7.2 APIs Interacting with Test Environment

7.2.1 Starting the test

The device must be in a pre-defined state before the test is started. The pre-defined state is defined in 7.2.2. The test then is started automatically by the DUT as soon as the AIT is parsed and the test application (which is signaled as “AUTOSTART”) is detected. This will open the entry URL of the initial test page in the browser on the DUT.

The initial test page then includes the JavaScript file “../RES/testsuite.js” which contains the implementation of the JavaScript classes/functions defined in this document. Finally the testing API described in this document is 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>
```

- Add initial JavaScript to 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>
```

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(...);”

7.2.2 Pre-defined state

Before starting a test, the DUT must be in a special state which is described in this chapter. Setting the DUT to the pre-defined state is the only communication specific to the testing-environment between the Test Harness and the DUT (and therefore this is not described in this document). Obvious 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:

- known services to the DUT:
- transponder 1 (network id=99, original network id=99, transport stream id=1) with following services:
  - service “ATE test10” (TV service, id=10 – first entry in the service list. This service is not used by any test, but can be used by the test environment to return to a pre-defined state. The content of this service is not defined by the test suite. The content of this service may be changed to any possible content required by the test environment, as long as all other services are not affected).

  - service “ATE test11” (TV service, id=11 – second entry in the service list)
  - service “ATE test12” (TV service, id=12 – third entry in the service list)
  - service “ATE test13” (TV service, id=13 – fourth entry in the service list)
  - service “ATE test14” (Radio service, id=14 – fifth entry in the service list)
- tuned to transponder 1, service “ATE test11”
- no application is running, so the AUTOSTART application signaled on that service will be started automatically.

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

7.2.3 Callbacks

All API calls defined in this document are defined to be asynchronous, as the API will have to interact with the test harness. To find out when the API call actually comes into effect, the caller needs to provide a callback function. This function is called as soon as the API call was issued on the server. The first argument of the callback function is always the callback object also 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.

If the callback is irrelevant to the test, both callback function and callback object should be null or undefined. This will disable the callback.

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 is called in both cases. In a synchronous implementation, this would be done before the function has returned.

Specifying callback functions and callback objects is not required. These arguments may also be null if the caller is not interested in the callback.

7.2.4 JS-Function getPlayoutInformation

This function allows retrieving information on the current playout.

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

- 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 an associative array containing the following key/value pairs:
  - 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.

Notes:
- Implementation of this method may be asynchronous or synchronous.
- 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.
- In case this method calls fails, an empty object with no key/value pairs is returned.

7.2.5 JS-Function endTest

This function reports that the test case has completed.

```
void endTest();
```

Notes:
- 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. This is why a test implementer must make sure that the network connection is available at the end of the test. (see reportStepResult documentation).
- 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 store the stepId, the result, and the comment in order to be able to create a test report from this information. 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 reportStepResult(stepId : integer, result : boolean, comment : String);
```

- 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. Note: The step number is usually in ascending order, but this is not a fixed requirement (a test may actually skip test numbers and/or not report steps in ascending order. This is especially the case if multiple steps are executed in parallel).
  There is no direct correlation between the stepId in the implementation and 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. Note: if the result is false, the server will not stop the application immediately (this may take a few seconds or even minutes, depending on the server). Because of this, the testing application should ensure that no more reportStepResult calls are executed after this. The Test Harness shall ignore any reportStepResult calls received after a call with failed 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)

stepId values shall be unique throughout the execution of each test case. They shall not be repeated. For example: once a test has called reportStepResult (or one of the analyze functions) with a specific stepId value, it shall not call reportStepResult (or one of the analyze functions) with that stepId value again. Note that the test source code may have multiple calls to reportStepResult with a particular stepId value, as long as the test makes sure that 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:

- synchronous communication to the server (e.g. via a serial line connection)
- 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 down (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:

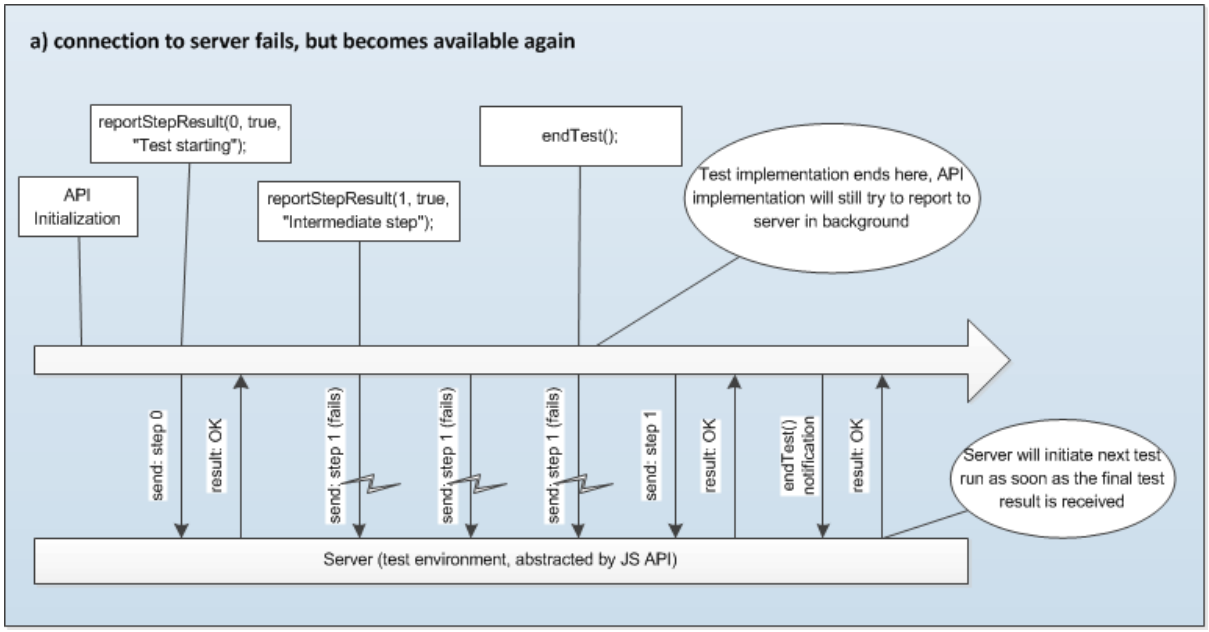

Figure 7-3 Communication between DUT and Test Harness when network is temporarily unavailable

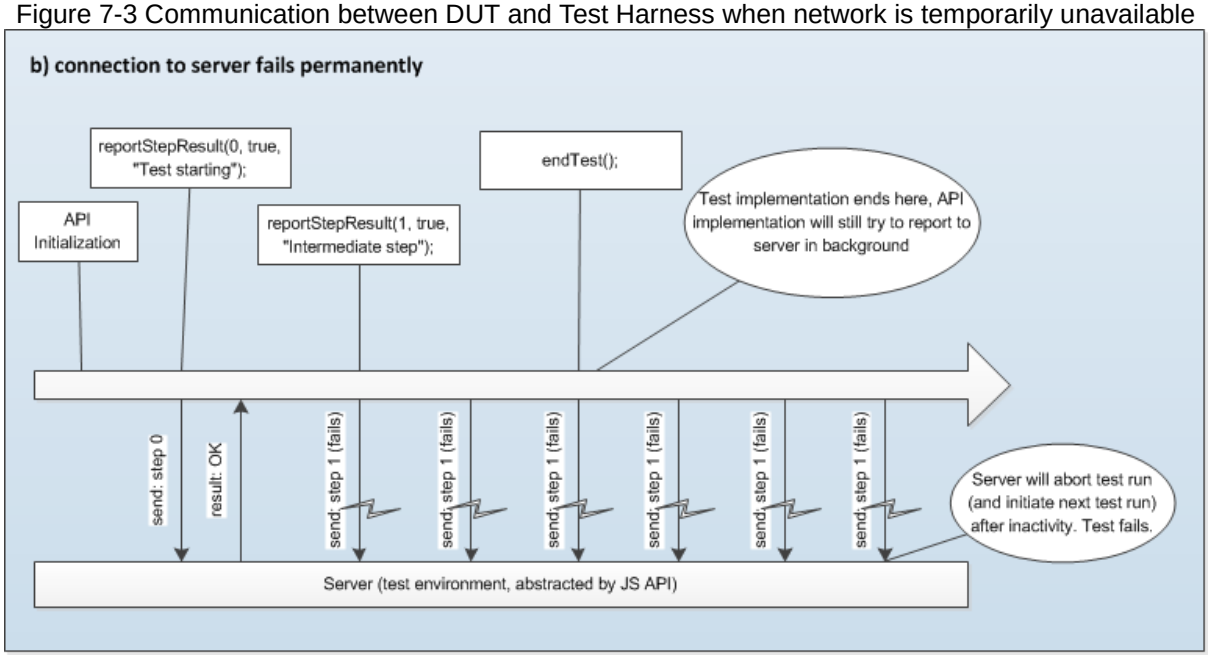

Figure 7-4 Communication between DUT and Test Harness when network is permanently unavailable

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 need to be stored: It is not specified how the Test Harness should handle these messages: They may be discarded, displayed, logged, etc.

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

- 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 may 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 could be 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 waitForCommunicationCompleted(callback : function, callbackObject :
object);
```

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

Notes:

- 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 arbitrary action. This function should only be used if the requested action cannot be achieved using other API methods (e.g. sendKeyCode).

```
void manualAction(check: String, callback : function, callbackObject: object);
```

- check: a textual description of the action required by 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 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.

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

- 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.
- 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 powercycle is defined by the device.
- callback/callbackObject: a callback function to invoke when the power cycle request was received (also see chapter 7.2.3 Callbacks).

Notes:

- Passing invalid parameters to this function (e.g. invalid type String) will cause the test to fail.
- 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 signaled on the initial service will be started automatically.

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 sendKeyCode(keyCode : String, durationSeconds : int, callback : function,
callbackObject : object);
```

- keyCode: a string describing the key to send: VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN, VK_ENTER, VK_BACK, VK_RED, VK_GREEN, VK_YELLOW, VK_BLUE, VK_0, VK_1, VK_2, VK_3, VK_4, VK_5, VK_6, VK_7, VK_8, VK_9, VK_PLAY, VK_PAUSE, VK_PLAY_PAUSE, VK_STOP, VK_FAST_FWD, VK_REWIND
- 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.
- callback/callbackObject: a callback function to invoke when the key code was actually sent to the receiver (also see chapter 7.2.3 Callbacks).

Notes:

- Passing invalid parameters to this function (e.g. invalid keyCode String) will cause the test to fail.
- 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 color.

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

- 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 color the pixel on the screenshot should match. The String must start with a hash (#) followed by a 2-digit hexadecimal representation of the red, green, and blue color (e.g. #ff0000 for red color).
- 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.

Notes:

- Pixel color matching is implementation dependent. For example, if the implementation uses a frame grabber, the pixel color matching probably will also compare pixels close to the specified position and may not be able to identify the exact color of the reference pixel(s). Inaccuracy of the frame grabber pixel position may be up to +/- 10 pixels in each direction of the specified pixel position, the pixel color value may be inaccurate by up to +/- 80/255 for each color component.
- 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 color of Pixel at Pos. "+posx+"/"+posy+" similar to reference color "+referenceColor+"?", callback, callbackObject).
- For some tests, analyzing the screen contents may require quick reaction (e.g. when checking broadcast video). For this reason, processing of a screen analysis shall not take longer than 2 seconds (delivering the analysis result to the callback, however, may take a lot longer).
- Passing invalid parameters to this function (e.g. invalid pixel position or reference color) will cause the test to fail.
- 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 analyzeScreenExtended(stepId : integer, comment : String, check : String,
callback : function, callbackObject : object);
```

- 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.

Notes:

- The same pixel and color inaccuracies apply as specified in the analyzeScreenPixel function.
- For some tests, analyzing the screen contents may require quick reaction (e.g. when checking broadcast video). For this reason, processing of a screen analysis shall not take longer than 2 seconds (delivering the analysis result to the callback, however, may take a lot longer).
- This function may be implemented by first taking the screenshot and then performing an offline analysis at a later point in time.
- Passing invalid parameters to this function (e.g. empty check String) will cause the test to fail.
- 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 analyzeAudioFrequency(stepId : integer, comment : String, channelId :
integer, referenceFrequency : int, callback : function, callbackObject :
object);
```

- 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 – center channel
  4 – rear left channel
  5 – rear right channel
- referenceFrequency: the reference frequency in Hz (allowed values are 500 Hz, 630 Hz, 800 Hz, 1000 Hz, 1250 Hz, 1600 Hz, 2000 Hz, 2500 Hz, 3150 Hz, 4000 Hz).
- 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.

Notes:

- Audio frequency matching is implementation dependent. E.g., if 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. Only frequencies should be detected that have a minimum signal level between -50dB and 0dB.

- 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).
- For some tests, analyzing audio may require quick reaction (e.g. when checking broadcast audio). For this reason, processing of an audio analysis shall not take longer than 2 seconds (delivering the analysis result to the callback, however, may take a lot longer).
- This function may be implemented by first making an audio capture and performing the analysis at a later point in time.
- Passing invalid parameters to this function (e.g. invalid channelId) will cause the test to fail.
- 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.

7.3.6 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 analyzeAudioExtended(stepId : integer, comment : String, check : String,
callback : function, callbackObject : object);
```

- 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 criteria that shall be used for the assessment of this analysis call. The described check may not require more audio data than the 10 seconds of audio after this function call was made. This allows the audio to be recorded (duration 10 seconds) and then processed later on.
- 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.

Notes:

- Audio frequency matching is implementation dependent. 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 some tests, analyzing audio may require quick reaction (e.g. when checking broadcast audio). For this reason, processing of an audio analysis shall not take longer than 2 seconds (delivering the analysis result to the callback, however, may take a lot longer).
- This function may be implemented by first making an audio capture and performing the analysis at a later point in time.
- Passing invalid parameters to this function (e.g. empty check String) will cause the test to fail.
- 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 analyzeVideoExtended(stepId : integer, comment : String, check : String,
callback : function, callbackObject : object);
```

- 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. The described check may not require more video data than the 10 seconds of video after this function call was made. This allows the video to be recorded (duration 10 seconds) and then processed later on.
- 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.

Notes:

- For some tests, analyzing video may require quick reaction (e.g. when checking broadcast video). For this reason, processing of an video analysis shall not take longer than 2 seconds (delivering the analysis result to the callback, however, may take a lot longer).
- This function may be implemented by first making an video capture and performing the analysis at a later point in time.
- Passing invalid parameters to this function (e.g. empty check String) will cause the test to fail.
- 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.

7.3.8 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 analyzeManual(stepId : integer, comment : String, check: String, callback :
function, callbackObject: object);
```

- 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 by 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 selectServiceByRemoteControl(serviceName: String, callback : function,
callbackObject : object);
```

- serviceName: the name of the service to select, e.g. "ATE Test11"
- callback/callbackObject: a callback function to invoke when the key codes were actually sent to the receiver (also see chapter 7.2.3 Callbacks).

Notes:

- Passing invalid parameters to this function (e.g. invalid serviceName String) will cause the test to fail.
- 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.

- VK_OK may be included in the sequence of key codes sent to the DUT.
- 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 must ensure no HbbTV applications are requesting the NUMERIC KeySet.

7.4 APIs Interacting 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” shall be active when the test starts. The referenced AIT should start the actual test application. Some tests may require multiple playout sets and a switching between those. 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.

Notes:

- The playout set with ID 1 is the initial playout set and must always be provided.
- 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.

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 to that, 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>
        <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="<file>" dst="<name>" />*
          <file src="<file>" dst="<name>" />*
          <streamEvent src="<file>" 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 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.

Notes:

- When writing 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.
- 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
- The playout set definition will always only affect the transponder 1. Currently, only transponder 1 is defined and required to run the tests of the test suite. In future test suite versions, there might be a transponder 2 containing a pre-defined transport stream file that is used for all tests. (used in very special test cases to test whether switching between two transponders is possible).
- All bitrates 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 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. The value of 16 is not permitted for dst (see 7.4.4.4), if it is present then the transport stream shall not be generated.

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 behavior 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 5.3 of [4]. 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], then the test harness shall not generate the transport stream.

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 not 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 behavior 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 behavior described in 7.4.4.5.1. If the value is outside the legal range then the transport stream shall not be generated.

7.4.4.4 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.

The test harness shall generate a NIT structured according to 5.2.1 of [18] and containing delivery descriptors as defined in 6.2.13 of [18] that shall correctly describe the delivery system that is in use. 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.

The generated transport stream should have a data rate matching that required by the parameters of the inserted delivery descriptors.

7.4.4.5 Optional behavior

The test harness may implement these behaviors.

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.

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.

7.4.5 JS-Function changePlayoutSet

This call changes the current playout (and may enable or disable the network connection).

```
void changePlayoutSet(playoutSetId : integer, callback : function,
callbackObject : object);
```

- playoutSetId: the ID of the playout set to start playing out
- callback/callbackObject: a callback function to invoke when the playout was changed (also see chapter 7.2.3

Notes:

- Passing invalid parameters to this function (e.g. invalid playoutSetId) will cause the test to fail.
- 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.
- To switch from no network to an available network connection, use the timeout parameter in the playout set.
- 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 artifacts, test case implementers should not perform any broadcast audio/video checks up to 5 seconds after the switch was confirmed (callback was called).

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 setNetworkBandwidth(bitrate : int, callback : function, callbackObject :
```

```
object);
```

- bitrate: the maximum throughput of application data into the device under test, in units of bits per second (bps). Values less than 1bps are not supported, and shall raise an exception.
- callback/callbackObject: a callback function to invoke when the restriction has been applied (also see chapter 6.2.3 Callbacks).

Notes:

- Throughput here is defined as the average number of bits per second passed in to the transmitting network interface at the application layer. i.e. the figure excludes all HTTP, TCP/UDP, IP, MAC, LLC, etc. overheads.
- 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.

7.5 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 an 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 channels (this avoids changing the playout set): Channel A contains the application 1 as AUTOSTART, which is responsible for performing the test. The application tunes to channel 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 color) 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.
    a. If it cannot be interpreted as a hex number, take the SHA-1 hex digest of the local part
    b. 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 only permits the 'strict' or 'transitional' XHTML doctypes, as defined in CEA-2014-A (Annex G)
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.

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.
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 8-1 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.

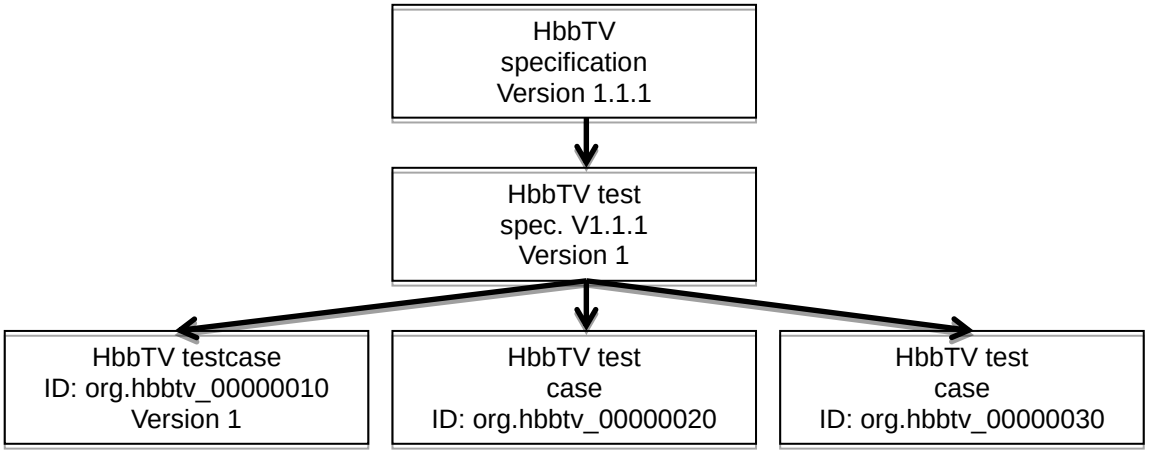

Figure 8-1 Version structure of the initial Test Specification release

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 8-2.

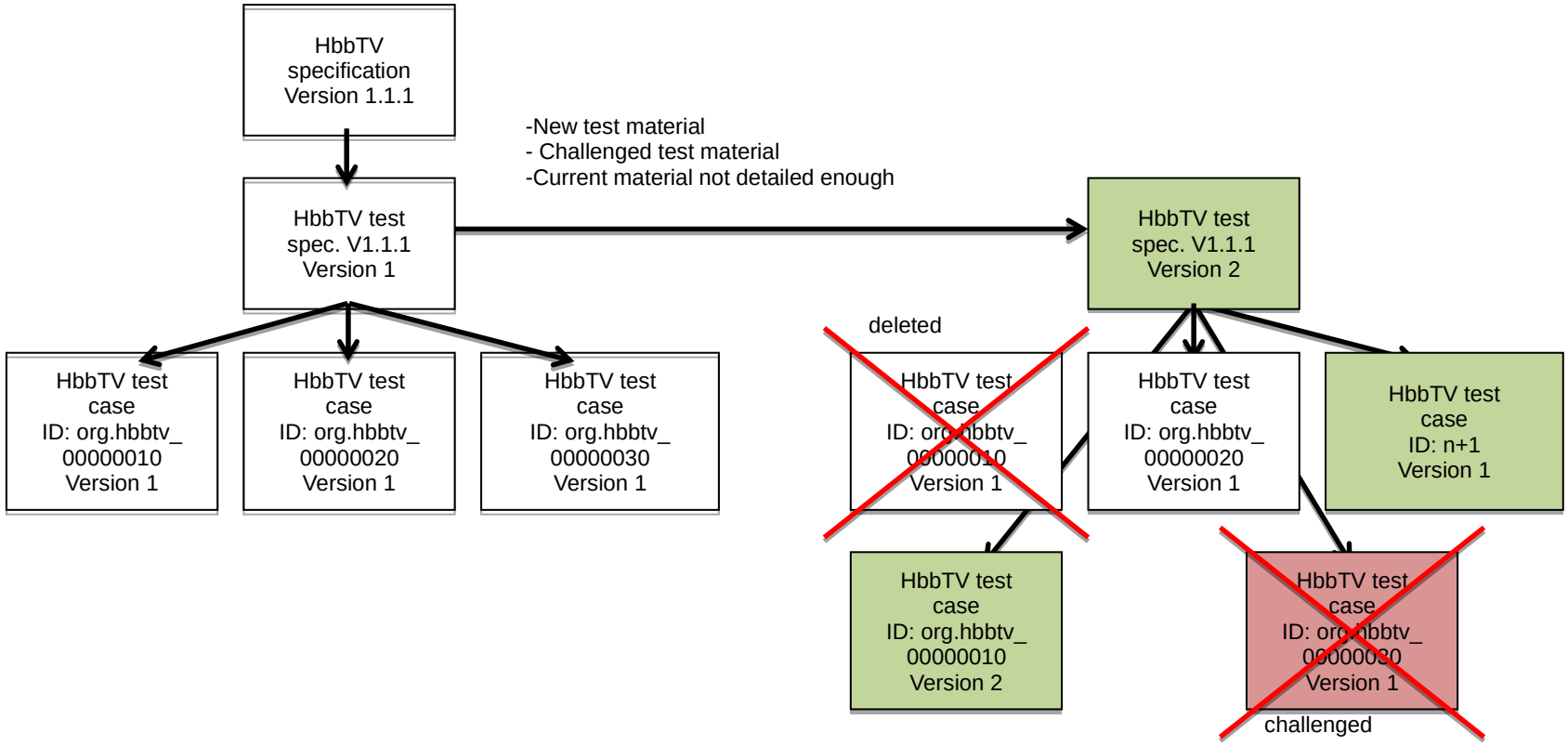

Figure 8-2 Example of creating a new version Test Specification while keeping the existing version of the Technical Specification

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 8-3.

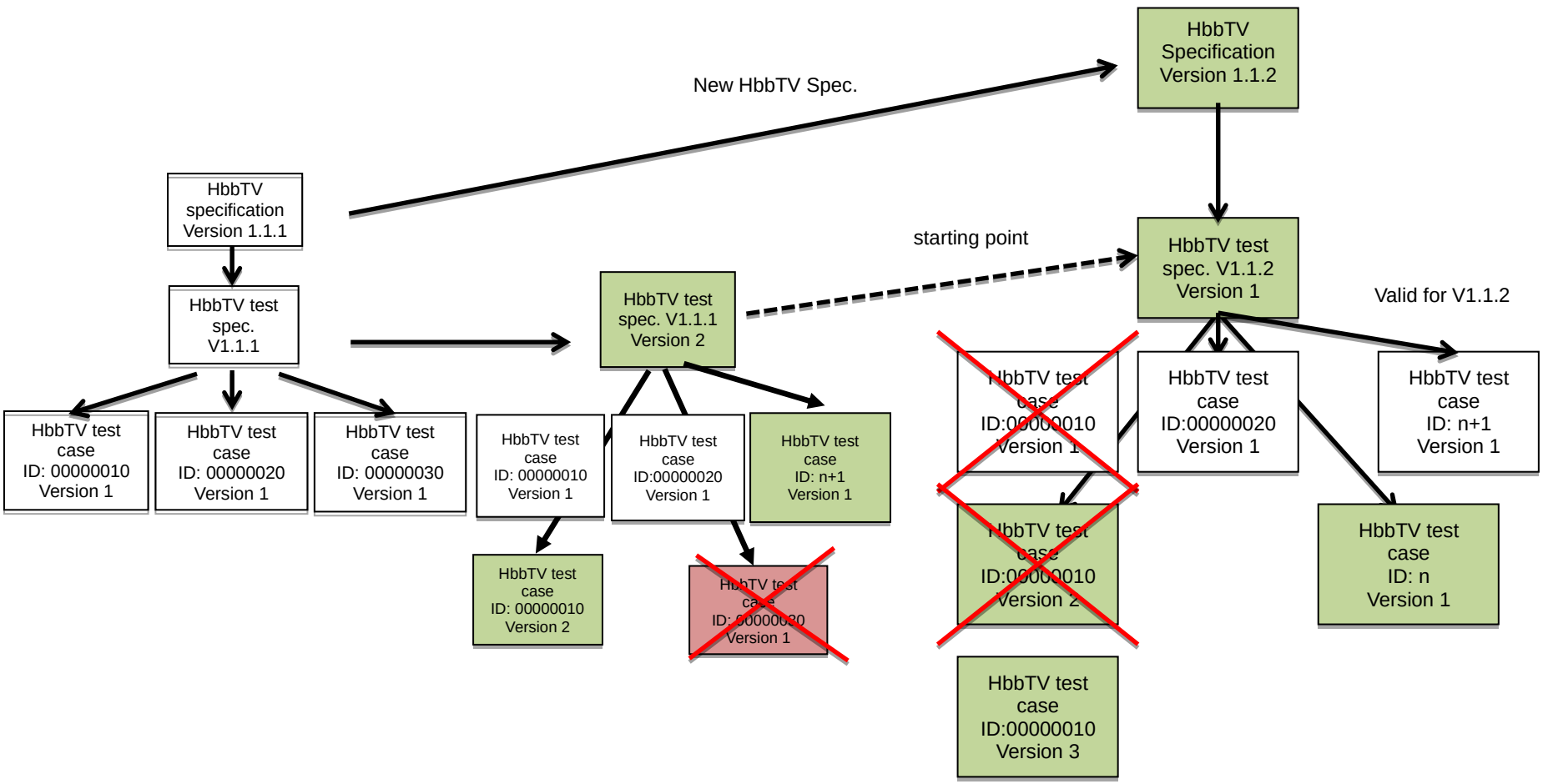

Figure 8-3 Example of creating a new version Test Specification after creating a new version of the Technical Specification

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, only the version is increased. 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 is changed whenever there is a functional change in the test material. This includes changes to Assertion, Procedure or any Test Assets (e.g. HTML-, JavaScript-, CSS and Media-Files etc.). The version need only be changed once per release. Minor typographical edits will not require a version change. It is at the discretion of the Testing Group to decide what constitutes a functional change of a Test Case.

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 numbers is administrated by the Test Group.

The Test Case Version number shall have an increased number from the previous version. The numerical increase is one (1) and is administered 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

HbbTV Specification, Version 1.1.1, section 10.2.4; and in HbbTV Specification, Version 1.2.1, section 10.2.4. 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. 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.

9.1.2 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
    - **[…]**

Document history

Draft nr	Date	Comments
draft 3	December 12th 2012	Mostly complete draft derived from the v1.1.1 document
draft 4	December 18th 2012	Changes from comments and formatting fixes
draft 6	March 27th 2013	Final proposals ahead of publication for RfP
draft 9	April 17th 2013	Removal of MTH and clarification of OpenCaster limitation
1.0	July 19th 2013	First publication
1.1d1	September 25th 2013	Clarification of Test Report format, minor clarifications
1.1d2	October 4th 2013	Note on StreamEvent naming and x-ref fixes.
1.1	October 11th 2013	Second Publication
1.2	July 10, 2014	Editorial improvements prior to wider publication