IPTV STANDARD

HTML5 Browser Specification

IPTVFJ STD-0011 Version 2.1

Created on March 22, 2013 (Version 1.0)
Revised on June 29, 2014 (Version 2.0)
Revised on December 15, 2014 (Version 2.1)

IPTV Forum Japan

<Blank>

Table of Contents

Chapter 1 Overview

This chapter describes reference specifications and guidelines on applying HTML5 to TV.

1.1. Reference specifications

All or parts of the specifications and recommendations listed below constitute, through references in this text, provisions of this specification. If a version is specified for any of the specifications and recommendations listed, that version shall be referred to. If no version is specified, the latest version shall be referred to. It should be noted that draft specifications or recommendations may be changed extensively in the future.

1) W3C Candidate Recommendation “HTML5 A vocabulary and associated APIs for HTML and XHTML” http://www.w3.org/TR/html5/
2) W3C Recommendation “Media Queries” http://www.w3.org/TR/css3-mediaqueries/
3) W3C Working Draft “CSS Device Adaptation” http://www.w3.org/TR/css-device-adapt/
4) W3C Candidate Recommendation “CSS Fonts Module Level 3” http://www.w3.org/TR/css3-fonts/
5) W3C Working Draft “Document Object Model (DOM) Level 3 Events Specification” http://www.w3.org/TR/DOM-Level-3-Events/
6) “Data Broadcast Encoding and Transmission Schemes in Digital Broadcasting,” Standard ARIB STD-B24
7) "Digital Terrestrial Television Broadcasting Operation Specification," Technical Document ARIB TR-B14.
8) ECMA-262 (ISO/IEC 16262), ECMAScript 5th Edition
9) W3C Candidate Recommendation, “WebIDL” http://www.w3.org/TR/WebIDL/
10) IPTVFJ STD-0006 “IPTV Specification: CDN-scope Service Approach Specifications Version 1.3”
11) IPTVFJ STD-0007 “IPTV Specification: Internet-scope Service Approach Specifications Version 1.2”
12) Networked Digital Television, ”Functional Specifications for Networked Digital Television: Streaming Function Specifications – Browsers Part Version 1.2”Terminology

1.2. Terminology

Term	Description
HTML	HyperText Markup Language
HTML5	In a limited sense, it refers to the fifth revision of HTML. In a broader sense, it refers to a web application platform that includes

	functions defined in separate specifications, such as CSS3 and various APIs.
CSS	Cascading Style Sheets
JavaScript	Script language used in web pages
DOM	Document Object Model
ARIB	Association of Radio Industries and Businesses: Entity that standardizes technologies concerning the use of radio waves in Japan, with the participation of broadcasters, telecom carriers and equipment manufacturers
W3C	World Wide Web Consortium
HTTP	HyperText Transfer Protocol [RFC2616]: Protocol in the application layer, used for data transfer for the worldwide web
NPT	Normal Play Time: Absolute coordinate on a timeline, indicating the position of the occurrence of an event in a stream
Aspect ratio	Ratio of the width to the height of the image display area
Font	A set of characters. Distinguished by font type and size
DTD	Document Type Definition
General application	Refer to the relevant definition in IPTVFJ STD-0010 Chapter 3
Unmanaged state	The state in which a generation application is being executed
Application boundary	Refer to the relevant definition in IPTVFJ STD-0010 Chapter 3
Broadcast resource	Image, audio, metadata, SI information, etc. used in broadcasting
HTML application	Application that comprises one or more HTML files
HTML file	File that contains content written in HTML
Page	Web page. A screen built and displayed based on an HTML file
Application transition	Transition in which the page of the currently loaded HTML application is released, and a page of another HTML application is loaded and displayed
Page transition	Transition in which the current HTML page is released, and another HTML page is loaded and displayed. This transition occurs within an HTML application.
URL	Uniform Resource Locator
PC	Personal computer
Mobile terminal	Mobile information terminal equipped with communication capability, such as a smartphone or a tablet terminal

API	Application Programming Interface
Browser	Web browser
Application control information	Refer the relevant definition in IPTVFJ STD-0010 Chapter 3
AIT	Application Information Table. If there is no risk of confusion, it may also refer to application control information
Overscan	Operating state in which the peripheral part surrounding a video screen or a browser image screen is not displayed.
Companion application	Software that runs on a terminal that is associated with a TV receiver. When it operates, it communicates with applications running on the receiver's application engine in order to realize a service that uses the receiver and the terminal in an integrated manner.

1.3. Basic policy

In establishing the specifications and guidelines in this document, it is intended to allow services and receivers to develop and evolve in a step-by-step manner while ensuring that ongoing services are not uninterrupted. This section describes basic principles for achieving the above.

1.3.1. How to apply HTML5 to TV

The basic policy is to use the HTML5 recommendations (including peripheral specifications, such as CSS3 and JavaScript APIs) specified by W3C (World Wide Web Consortium) without modification as far as possible. When this standard was established, W3C HTML5 recommendations still included working drafts and thus were subject to change. Some functions may be removed from the final recommendations before they are implemented in browsers, and some may be modified in the final recommendations in order to reflect what is learned from their implementation in browsers. It should be noted that provisions that may be modified are so indicated in the draft W3C recommendations.

If functional requirements for TV can be satisfied by functions specified in W3C recommendations, no new requirement will be added in this Specification. This Specification also assumes the possibility that some functional requirements may be satisfied by using libraries and frameworks.

1.3.2. Service evolution and diversity of devices

This Specification does not explicitly limit TV receiver functions (except for functions that contradict the service requirements). In other words, a receiver must have the minimum functions needed to provide a service (the specific scope of the required functionality shall be determined for each service), but it can have additional functions. A receiver may be implemented with additional functions in accordance with whatever product plan the particular manufacturer may have. A service provider will be able to provide a better service by using such additional functions as appropriate. It is generally expected that a receiver's functions will continue to become more sophisticated with a result that functions provided by receivers and the use of those functions by service providers will evolve step by step, leading to continuous development and evolution of services.

However, under the conditions assumed above, confusion may arise because browsers with additional functions and those without them may coexist in the market. Therefore, it is recommended that any application developed based on this Specification either use such additional functions only after confirming that the browsers being used support them, or provide a fallback capability for any browser that does not support them in order to avoid cases where the user becomes unable to operate the service or is baffled because a particular service feature is meaningless.

Table 1-1 Analysis of risks arising from differences in the level of implementation of functions in browsers

Risk level	Typical case	Typical remedies
Level 1: The necessary functions are absent, and there is no alternative means available. The user is unable to continue to use the application or service.	Absence of JavaScript API	Avoid developing such an application
Level 2: The necessary functions are absent, but the user can continue to use the application or service by avoiding these functions or by replacing them with alternative methods.	HTML5 elements and absent functions are substituted for by JavaScript	Use functions selectively or replace the functions concerned with alternative methods in order to avoid the use of the absent functions
Level 3: The necessary functions are present, but their behavior varies significantly from browser to browser.	Differences in the visual effect processing with CSS transitions and animations	No action will be taken if the differences in behavior are tolerable. If intolerable, take actions similar to those in Level 2.
Level 4: The necessary functions are	Differences in decoration displays by CSS3	In principle, no action is required

sufficient but their behavior varies slightly from browser to browser.		

Based on the risk classification above, this Specification provides examples of how to check the presence of functions supported by a browser and how to replace a process with an alternative method.

1.3.3. Significance of recommended methods and basic policy

When any method is recommended in these Specifications (when “is recommended” is explicitly written), this is done because that method has taken application compatibility and differences in browser execution environments into consideration. These aspects must be taken into consideration when writing an application and selecting a means of implementation. There may be methods other than the recommended ones that can satisfy particular requirements. This Specification does not prohibit the use of such methods. However, the operation of such methods will not always be guaranteed and may depend on the particular implementation or on the particular machine on which they run.

1.3.4. Considerations in applying HTML5 to TV

Applying HTML5 to TV is not the same as applying it to PCs, tablet terminals or smartphones because there are certain features that are specific to TV.

1. Pointing device
   TVs use remote control units, but a different type of device shall be used to point to a particular position on the screen. It is necessary to consider the use of a mouse, a trackball or other pointing device.
2. 10-foot UI
   It is generally assumed that, unlike a PC or a tablet terminal, a TV is operated from a distance. This should be taken into consideration in selecting the font size and the number of characters to be displayed.
3. Execution of applications to match the sequential progress of the broadcast audio and video.
   It may be necessary to update application control information to match the progress of the broadcast audio and video, and to load or terminate particular applications. The capability of switching the displays of HTML applications while continuing to show the broadcast video is not included in the W3C recommendations. It is necessary to consider this feature in this Specification.

1.3.5. HTML application model

An HTML application consists of HTML, CSS, and JavaScript codes as well as the external resources (images, audios, videos, etc.) that they may refer to (Fig. 1-1 Elements that make up an HTML application).

HTML is a markup language used to indicate the document structure. An HTML file written in HTML provides the basis on which the browser (the application processing part) that loads this file determines its action. A data structure based on DOM (Document Object Model) is created within the browser during runtime in accordance with what is written in the HTML file, and this object model provides the basis for the browser's action.

CSS is used to describe the behavior of each element described in the HTML file, such as how it looks, how it is laid out and how it is animated.

JavaScript is a programming language. It is used to perform input and output, to implement the processing for waiting for events, to perform control through an API and to obtain various values. In addition, it is used to change the overall structure, operation, look and layout of a webpage by dynamically overwriting HTML and CSS codes (based on DOM).

CSS and JavaScript codes can be included in an HTML file. Alternatively, the HTML file can include references to the associated CSS files and JavaScript files. Therefore, what is written in the HTML file entirely determines how the browser displays the webpage concerned, such as how it operates and how it is laid out. In addition, the HTML codes and associated CSS and JavaScript codes specify the images and videos to be shown, their positions on the screen and their display sizes. Therefore, the HTML file developer can have complete control on how a webpage is displayed and how it behaves (Fig. 1-2 HTML and screen layout control).

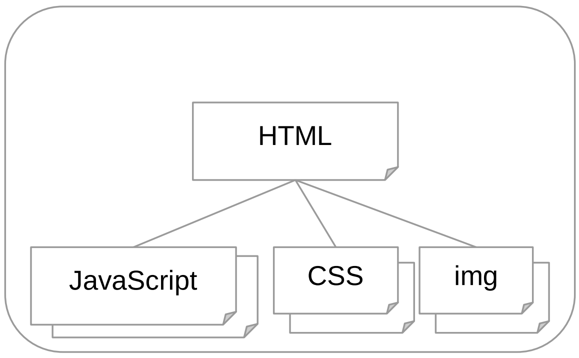


Fig. 1-1 Elements that make up an HTML application

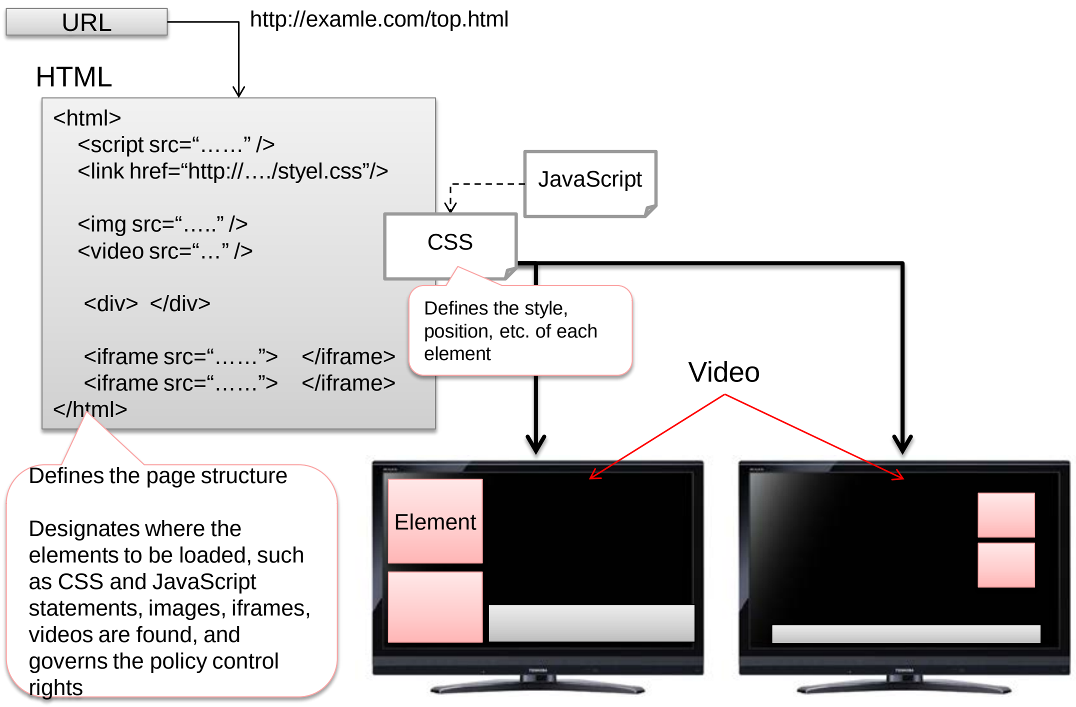

URL
http://examle.com/top.html
HTML
<html>
<script src="……" />
<link href="http://…./styel.css"/>
<img src="….." />
<video src="…" />
<div> </div>
<iframe src="……"> </iframe>
<iframe src="……"> </iframe>
</html>
JavaScript
CSS
Defines the style, position, etc. of each element
Video
Defines the page structure
Designates where the elements to be loaded, such as CSS and JavaScript statements, images, iframes, videos are found, and governs the policy control rights
Element

Fig. 1-2 HTML and screen layout control

An HTML application consists of one or more HTML files (hereinafter referred to as “pages”) (Fig. 1-3 Structure of an HTML application). The structure of each page is determined by the HTML file, which is designated by a URL (Uniform Resource Locator). A page can behave dynamically according to the behavior of the page’s elements and CSS codes and through dynamic overwriting by JavaScript codes. Dynamic behavior can also be created by using multiple pages, loading pages through appropriate links to cause screen transitions.

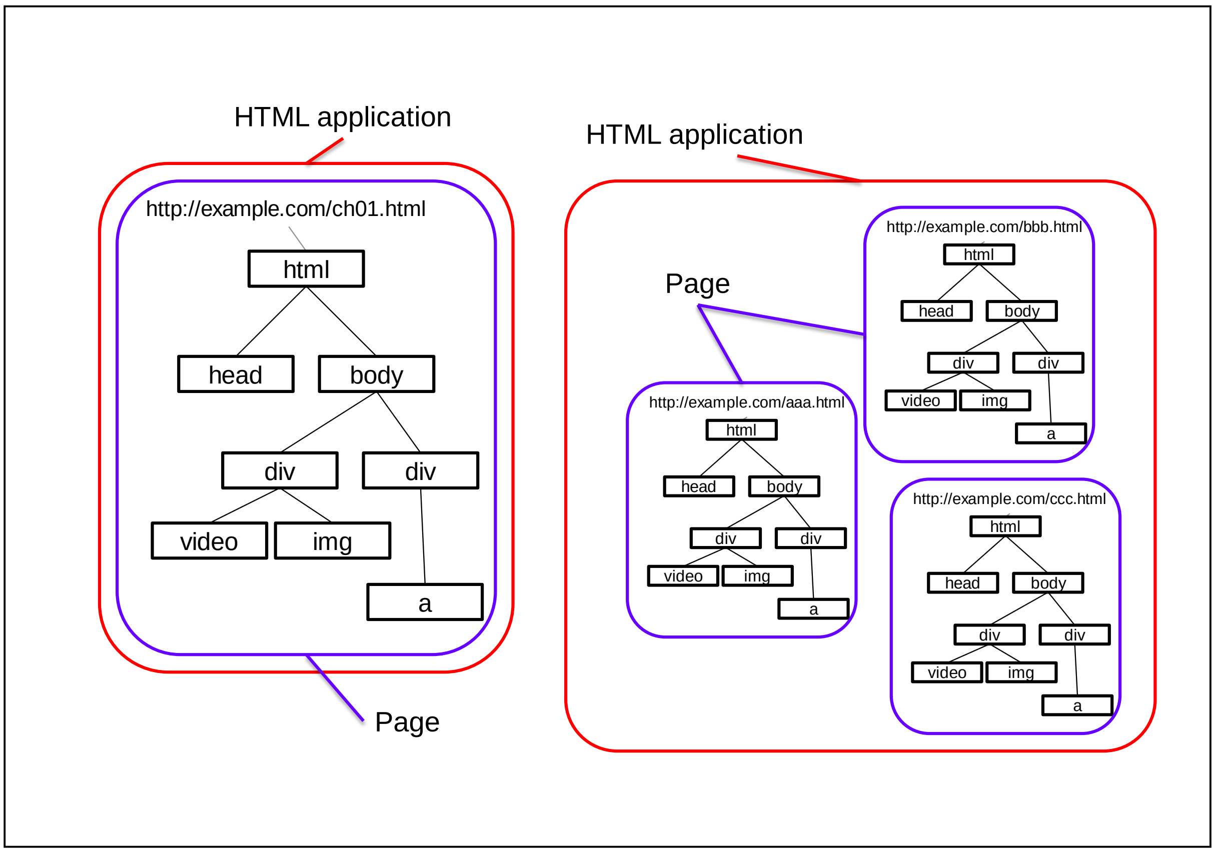

HTML application
http://example.com/ch01.html
html
head
body
div
div
video
img
a
Page
HTML application
Page
http://example.com/bbb.html
http://example.com/aaa.html
http://example.com/ccc.html

Fig. 1-3 Structure of an HTML application

When an HTML application consists of multiple pages, a page transition is caused by a user clicking a link specified by an “a” (anchor) element, or by the operation of JavaScript codes. Each time a transition occurs, the browser loads the HTML file of the designated URL, builds the page and displays it ().

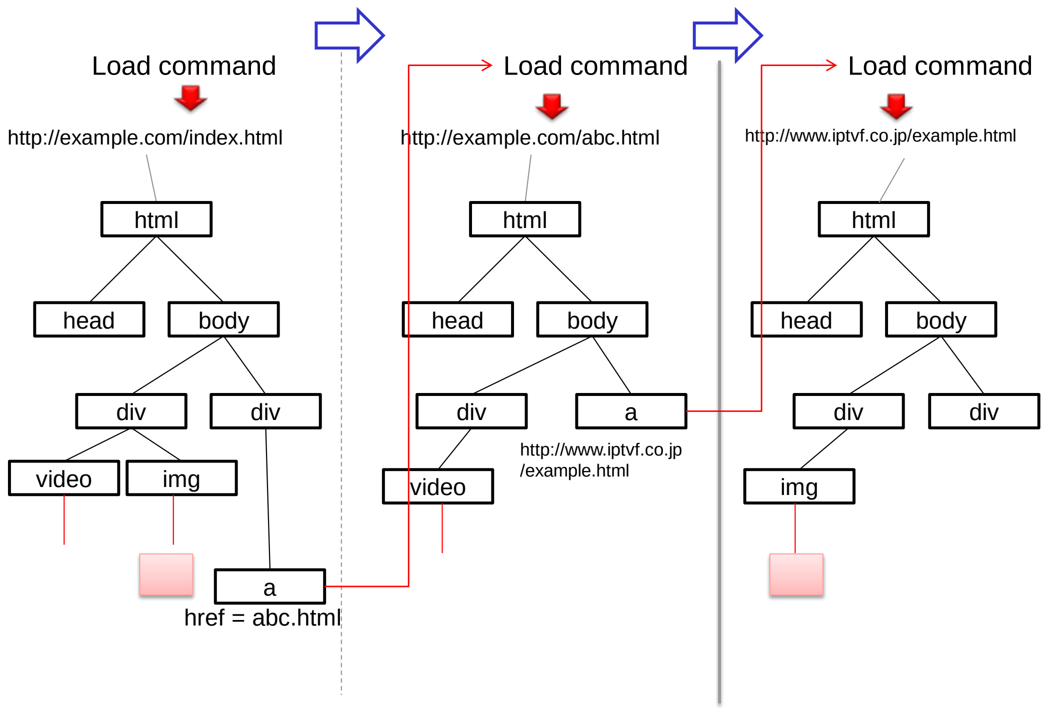


Fig. 1-4 Page transition

1.3.6. Data handover model

A loaded HTML application can call an API with JavaScript codes and send data to an external entity through event processing. This section describes the model used for this purpose.

1.3.6.1. Sender and receiver of data

The following can be the sender or receiver of data.

(1) Web server

Web server that provides an HTML application

(2) Device function

Function provided by the HTML application execution environment. It obtains or changes the device's state.

(3) HTML application

When multiple HTML applications are to operate, they exchange messages to cause coordinated actions or to share data.

(4) Coordinated operation with a terminal

A TV exchanges messages with a terminal, such as a PC or a mobile terminal, with which the user operates the TV, in order to cause coordinated actions.

1.3.6.2. PULL model

In the PULL model, JavaScript codes of an HTML application issue a data acquisition request to a function and the application receives response data from the called function. There can be two types of call for invoking this action: synchronous and asynchronous calls. In the synchronous call, the caller receives data only after the called function has completed its processing. In the asynchronous call, the caller receives data as soon as it has called a function. Since the application calls an entity that is outside its execution environment, it is assumed to take a relatively long time for the caller to receive response data. Asynchronous calls are preferred because it is recommended to avoid JavaScript processing from being blocked. To receive response data, a callback function shall be specified. The specified function receives response data and performs the necessary processing.

1.3.6.3. PUSH model

In the PUSH model, an entity outside an HTML application sends data to the HTML application.

The JavaScript code concerned specifies a callback function and registers it with an event handler. When data arrives from an external entity, the specified callback function is called. It receives data and performs the necessary processing.

Chapter 2 Application of HTML5 to TV

This chapter describes how W3C HTML5 recommendations (HTML5, CSS3, and JavaScript functions) shall be applied to TV.

2.1. HTML5 syntax

For HTML5 syntax, refer to W3C Recommendation HTML5 – Section 8 “The HTML syntax” (http://www.w3.org/TR/html5/syntax.html#syntax).

2.2. The DOCTYPE

The DOCTYPE shows the type and version of HTML used in the document. It shall come at the head of the document. To show that the document conforms to W3C Recommendation HTML5, the DOCTYPE is written as follows.

```
<!DOCTYPE html>
```

If the document type is declared in a way inconsistent with HTML5, such as omission of DOCTYPE and inclusion of a DTD statement in a parameter, the browser may draw a webpage in a way different from what is expected with HTML5, possibly resulting in a layout not intended by the author. Therefore, it is recommended to write a DOCTYPE that explicitly specifies conformance to HTML5.

For details, refer to W3C Recommendation HTML5 - Section 8.1.1 “The DOCTYPE” (http://www.w3.org/TR/html5/syntax.html#the-doctype).

2.3. The HTML element

The HTML element represents the root of an HTML document.

```
<html>
```

The language used in the document can be specified using the *lang* attribute.

In the case where English is used in the document

```
<html lang="en">
```

In the case where Japanese is used in the document

```
<html lang="ja">
```

For details, refer to W3C Recommendation HTML5 - Section 4.1.1 “The html element” (http://www.w3.org/TR/html5/semantics.html#the-html-element)

2.3.1. HTML application cache

When an HTML application is executed, it loads files that make up the application. In order to be prepared for temporary disconnection of the file acquisition paths or to reuse files obtained previously to reduce the processing load on the server and network, it is recommended to use an application cache in accordance with the W3C Offline Web Applications Specification.

For details, refer to W3C Recommendation HTML5 - Section 5.7.2 “Application cache” (http://www.w3.org/TR/html5/browsers.html#appcache).

2.4. Head element

The head element represents a set of information (metadata) about the HTML document.

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Metadata   -->
    </head>
</html>
```

For details, refer to W3C Recommendation HTML5 - Section 4.2.1 “The head element” (http://www.w3.org/TR/html5/document-metadata.html#the-head-element).

2.5. The title element

This element shows the title of the HTML document.

```
<title> IPTV Forum HTML5 Specifications </title>
```

This element shall not appear more than once in one document.

For details, refer to W3C Recommendation HTML5 - Section 4.2.2 “The title element” (http://www.w3.org/TR/html5/document-metadata.html#the-title-element).

2.6. The meta element

This element provides information about the document using meta data.
For details, refer to W3C Recommendation HTML5 - Section 4.2.5 “The meta element” (www.w3.org/TR/html5/document-metadata.html#the-meta-element).

2.6.1. Character set (charset)

An example of specifying the character set used is as follows.

```
<meta charset="UTF-8" >
```

It is recommended to specify the character set used by the HTML document.

2.7. The link element

For the link element, refer to W3C Recommendation HTML5 - Section 4.2.4 “The link element” (http://www.w3.org/TR/html5/document-metadata.html#the-link-element).

It is recommended to specify the styles of the HTML application in separate files, and provide links to these files using link elements, rather than specifying the styles within the HTML file.

```
<link rel="stylesheet" href="example.css">
```

The style sheets appropriate for each user device environment can be applied using conditional expressions in Media Queries.

An example of switching links to style sheets depending on given conditional expressions is shown below.

```
<link rel="stylesheet" media="device-width:   1280px" href="720p.css">
<link rel="stylesheet" media="device-width:   1920px" href="default.css">
```

It is also possible to apply conditions within CSS codes. For details, refer to W3C Recommendation “Media Queries” (http://www.w3.org/TR/html5/infrastructure.html#mq).

Although “TV” is defined as a media type in the Media Queries specification, whether “TV” is actually identified in Media Query depends on the implementation of the browser used. Therefore, it is necessary to pay due consideration to this when using media type “TV.” It is recommended to change the style depending on the screen size of the device used.

2.8. The viewport meta element

It is expected that TV receivers of different resolutions will coexist. It is necessary to address cases where the drawing resolution handled by the browser is different from the resolution of the receiver’s display. How to use a viewport meta element to write an HTML application that supports different resolutions is described below.

2.8.1. When specifying a drawing resolution that is identical to the default resolution assumed in operational rules

When the width of the drawing resolution is a default value (e.g. 1920)

```
<meta name="viewport" content="width=1920”>
```

Even when the drawing resolution handled by the browser and specified in the viewport is different from the resolution of the display, it will be adjusted by means of scaling. The HTML application needs only to specify the layout based on the default size. It is recommended to specify the default width and height as operational rules to be prepared for cases where viewport is not specified or the browser in use does not support the viewport meta element. It should be noted that, when a receiver supports resolutions other than the default one, it shall support scaling of this viewport (Fig. 2-1 Browser drawing resolution and display resolution).

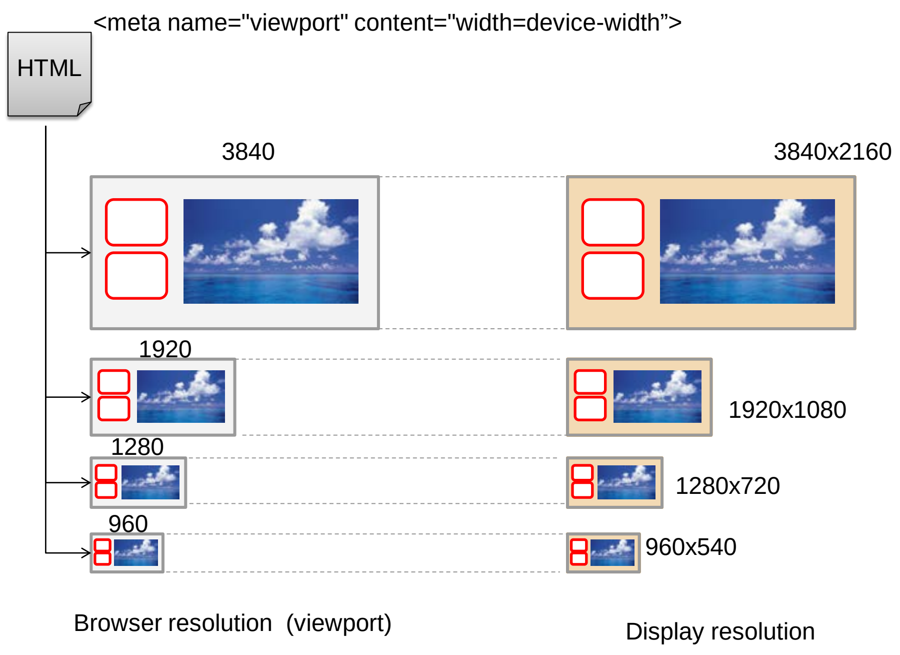

<meta name="viewport" content="width=device-width”>
HTML
3840
3840x2160
1920
1920x1080
1280
1280x720
960
960x540
Browser resolution (viewport)
Display resolution

Fig. 2-1 Browser resolution and display resolution

2.8.2. When specifying a drawing resolution different from the default resolution assumed in operational rules

An HTML application can set a layout by specifying a browser resolution using a viewport meta element. However, when a drawing resolution different from the default resolution is specified, it is recommended to support the default resolution in cases where the receiver does not support the viewport meta element.

For example, it is possible to use two layouts (one that suits the specified resolution and the other that suits the default resolution), and to switch between the two depending on the width value obtained by Media Query.

```
<meta name="viewport" content="width=3840”>
<link rel="stylesheet" media="(max-device-width:   1920px)" href="default.css">
<link rel="stylesheet" media="(device-width:   3840px)" href="4k2k.css">
```

It is should be noted that, while the viewport meta element is implemented in browsers for mobile terminals and embedded browsers, it is still at a draft stage in W3C, and so the specification and the implementation may change in the future.

For details of the viewport meta element and the handling of device resolution, refer to W3C Recommendation “CSS Device Adaptation”
(http://www.w3.org/TR/css-device-adapt/#viewport-meta-element).

2.9. The body element

This element represents the body of the document. Only one body element shall be placed in an HTML element.

For details, refer to W3C Recommendation HTML5 - Section 4.3.1 “The body element”
(http://www.w3.org/TR/html5/sections.html#the-body-element).

When you want to display a background image, specify the background image in the body element using a CSS code.

```
body {
      background-image:    url(background.png);
}
```

2.10. The script element

This element is used to embed a JavaScript script in the document or to load external scripts.

When loading an external file

```
<script src="example.js"></script>
```

When embedding script element contents within the document directly

```
<script>
   function example() {
      /* do sometihng */
   }
</script>
```

For details, refer to W3C Recommendation HTML5 - Section 4.11.1 “The script element”
(http://www.w3.org/TR/html5/scripting-1.html#the-script-element).

It should be noted that depending on the location of a particular script element, the loading of the script and the timing of its evaluation can vary. This can affect the application of CSS codes, and consequently affect the layout and initial event processing.

2.11. Layout

An area is created for each element based on the CSS box model. The size, line width, color, position and the order of overlapping can be specified using a stylesheet (CSS). For details, refer to W3C Recommendation CSS specifications (http://www.w3.org/Style/CSS/).

2.12. Presentation of a video

2.12.1. How to present a broadcast video

An object element is used to present a broadcast video.

For details of the object element, refer to W3C Recommendation HTML5 - Section 4.8.4 “The object element” (http://www.w3.org/TR/html5/embedded-content-0.html#the-object-element).

By specifying “video/x-iptvf-broadcast” in the type attribute of an object element, the default stream of the broadcast service of the currently selected channel can be presented. The initial operation parameter of the object element can be specified using a param element. For details of the parameters of the object element, refer to Section 3.2 “Broadcast audio/video object.”
Below is an example of presenting a broadcast audio and video.

(Code example) Description for presenting the audio/video stream of the broadcast service of the currently selected channel

```
<object type=”video/x-iptvf-broadcast”>
```

It is recommended to specify the broadcast audio/video object (object element that refers to the relevant broadcast audio/video) statically in an HTML application. It should be noted that, if the object is added dynamically, continuity of the broadcast audio/video stream is not guaranteed at the time of page transition. When the broadcast audio/video stream is the same after a page transition, it is expected that the stream will be shown without interruption at the time of page transition. However, if the object is added dynamically by a script, the browser may not be able to determine correctly whether the audio/video stream should be continued or not. This may result in the broadcast audio/video stream being interrupted.

2.12.2. How to present a network-delivered video

VOD will be specified in Section 2.23. Other types of video will be specified in the future.

2.12.3. Presentation example: Displaying elements over a full-screen broadcast video

An example of displaying text and an image over a full-screen broadcast video is shown in Fig. 2-2.

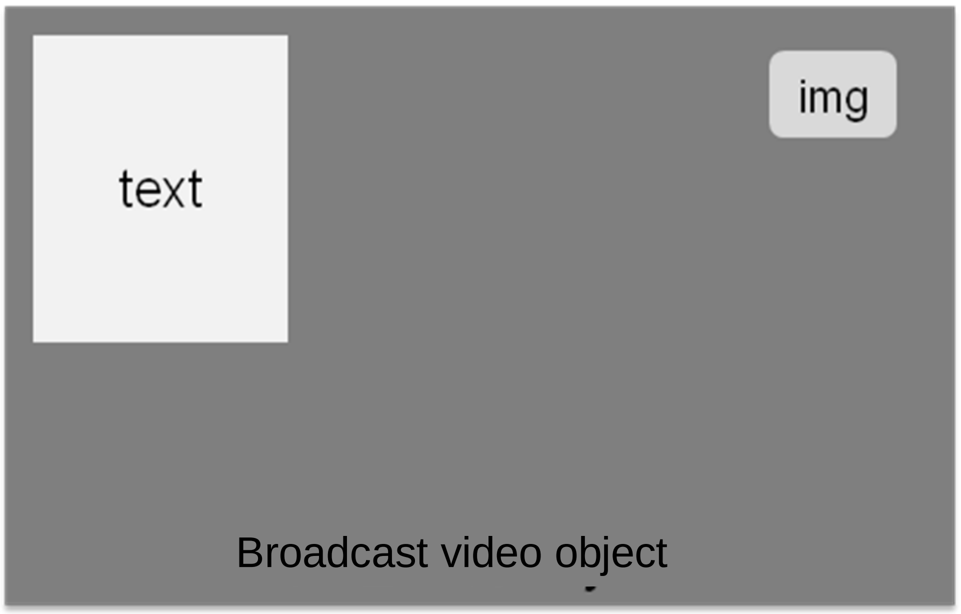


Fig. 2-2 Example of presentation: Displaying elements over a full-screen broadcast video

HTML code example

```
<!DOCTYPE html>
  <html lang="ja">
      <head>
      <title> Example service </title>
      <meta name="viewport" content="width=1920”>
      <link rel="stylesheet" href="example.css">
</head>

      <body>
            <object id=”video” type=”x-iptvf-broadcast”>

            <img id="logo" src="logo.png">
            <div id="text"> ............ </div>
      </body>
</html>
```

CSS code example

```
body {
      overflow:   hidden;
      margin:   0px;
}
```

```
#video {
        position:   absolute;
        z-index: 0;
        width:   100%;
        height:   100%;
        top:   0;
        left:   0;
}

#logo {
        position:   absolute;
        z-index: 2;
        left:   90%;
        top:   10%;
}

#text {
        position:   absolute;
        z-index: 1;
        width:   25%;
        height:   50%;
        left:   10%;
        top:   20%;
}
```

2.12.4. Example of presentation: L-shaped display

An example in which the size of the broadcast video is reduced, text information is shown outside the video, and an image is displayed over the video is shown in Fig. 2-3. The manner of displaying elements as shown in Fig. 2-3 is called an L-shaped display from its shape.

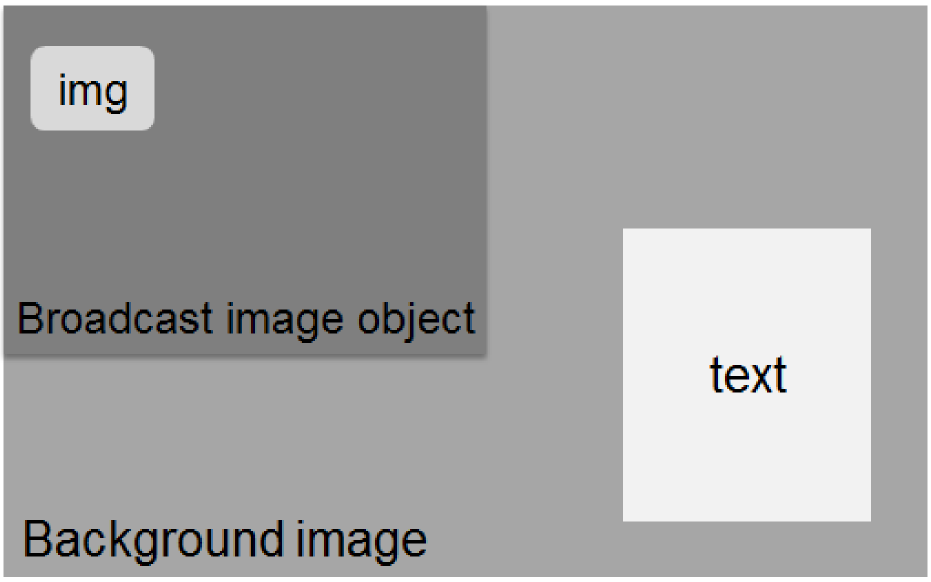


Fig. 2-3 Example of L-shaped display

HTML code example

```
<!DOCTYPE html>
  <html lang="ja">
      <head>
            <title> Example service </title>
            <meta name="viewport" content="width=1920”>
            <link rel="stylesheet" href="example.css">
      </head>

      <body>
            <object id=”video” type=”x-iptvf-broadcast”>

            <img id="logo" src="logo.png">
            <div id="text"> ............ </div>
      </body>
</html>
```

CSS code example

```
body {
      overflow:    hidden;
      margin:    0px;
      background-image:    url(background.png);
}

#video {
      position:    absolute;
```

```
    z-index: 0;
    width:   50%;
    height:   50%;
    top:   0;
    left:   0;
}

#logo {
    position:   absolute;
    z-index: 2;
    left:   90%;
    top:   10%;
}

#text {
    position:   absolute;
    z-index: 1;
    width:   25%;
    height:   50%;
    left:   0;
    top:   0;
}
```

2.13. Full-screen display of a video

This section specifies how to hide the display of an HTML application temporarily while the application is running, and to display the broadcast video on a full screen.

To display a video on a full screen from the time when a page is loaded (time when the broadcast audio/video object is generated), use a param element. Specify “fullscreen” for the name attribute and “enable” for the value attribute. This will cause the broadcast video to be displayed on a full screen from the time when the HTML application is launched. For details of the broadcast audio/video object, refer to Section 3.2. To switch to full-screen display by a key operation or the user’s UI operation, use a function that comes with the broadcast audio/video object. To switch to full-screen display, call the enableFullscreen() function. To disable full-screen display, call the disableFullscreen() function. To know the display mode, use the isFullscreen() function. For details of these functions, refer to Section 3.2.2.

Even during full-screen display, the browser continues to operate in the background. Since the processing associated with the pointer and key operations continues to operate, unexpected operations may occur. Pay due attention to event processing that is associated with user operations during full-screen display.

HTML code example (Full-screen display when an application is launched)

```
<object id=”video” type=”x-iptvf-broadcast” >
            <param name=”fullscreen” value=”enable”>
</object>
```

JavaScript code example for full-screen display

```
var video = document.getElementById('video');

if ('function' === typeof(video.enableFullscreen)) {
   video.enableFullScreen();
} else {
   /* Width, height and z-index are overwritten so that the element is displayed at the forefront on a full screen */
}
```

JavaScript code example for disabling full-screen display

```
var video = document.getElementById('video');

if ('function' === typeof(video.disableFullscreen)) {
   video.disableFullScreen();
} else {
/* Width, height and z-index are overwritten so that the element reverts to the previous size */
}
```

JavaScript code example for obtaining fullscreen state information

```
var video = document.getElementById('video');

if ('function' === typeof(video.isFullscreen)) {
   if (video.isFullScreen()) {
        /* In case of full screen display */
   } else {
         /* In case of normal display    */
   }
}
```

2.14. Text drawing

The font used to draw text can be specified in CSS.

```
#text {
      font-family:   Verdana, "Times new Roman";
      ......
}
```

However, the font used for drawing text may be different from that specified in the HTML application for reasons such as the unavailability of the specified font to the browser in the receiver. In such a case, the display layout and the quality of font display may differ from what is intended.

It should be noted that it will be easy to unify the actual layouts resulting from what is specified in the HTML application by specifying the fonts that receivers should have as operation rules.

Measures to be taken when the fonts available to the browser in the receiver are unknown are described below. These measures should be applied in accordance with the aim of the particular HTML application.

2.14.1. Use of monospaced fonts

If monospaced fonts are available to the browser in the receiver, it is expected that specifying “monospace” for the property of the font-family in CSS can ensure that the layout will not significantly vary depending on the types of font available in the execution environment. However, due attention should be paid to the operation and legibility when both alphabetic and numerical characters are present.

```
#text {
    font-family:   monospace;
    ......
}
```

2.14.2. Web fonts

Web fonts can be downloaded. It should be noted that use of Web fonts increases the volume of data, can cause a time lag between the loading and the display of the HTML application, and can cause memory shortage, resulting in degradation in performance.

It is recommended to pay due attention to cases where the particular browser used does not support the Web fonts specification.

Code example of using Web fonts

```
@font-face {
    font-family:   example-font-name;
    src:   url("example.woff"); format("woff");
}

#text {
    font-family:   example-font-name;
}
```

For details, refer to W3C recommendation CSS Fonts Module Level 3 (http://www.w3.org/TR/css3-fonts/).

2.14.3. Solutions through wise planning of a layout

When fonts used are different from those specified, the height or width of text boxes may differ from what is intended, or text may overflow from the boxes specified in CSS codes, resulting in overlapping of text with other elements. This can undermine legibility.

The following measures can be taken against this problem.

・Make sufficient room in the layout to absorb differences in font size.

・Specify “hidden” for the property of overflow in the style sheet. When this is specified, any text that overflows from a text box is hidden and thus does not interfere with the display of other elements. However, it should be noted that the viewer may not be able to receive the intended information because part of the information is hidden.

2.15. Display of graphics

There are several methods available for an HTML application to display graphic images

(1) The img element

```
<img src="example.png">
```

The position where an image is displayed can be specified in a style sheet. Representation of an image can be enriched by transforming or rotating the image. This can be done by using CSS’s decoration functions (text-shadow, etc.), or by combining them with other functions, such as CSS3 Transform.

Consider setting appropriate alternative text in the alt attribute as necessary.

For details of the img element, refer to W3C Recommendation HTML5 - Section 4.7.1 “The img element” (http://www.w3.org/TR/html5/embedded-content-0.html#the-img-element).

(2) The canvas element

This element is suitable for drawing and displaying an image dynamically. An image can be drawn using JavaScript codes. Since images can be generated within a receiver, the volume of communication data can be reduced. However, there are browser implementations that do not support the canvas element. Therefore, it is recommended to provide an alternative image to cater for such a case. The speed of generating and drawing images depends on the processing power of the receiver. When a particular drawing requires excessive computing power, operability may suffer. Ensure that the drawing processing load is such that it will not affect operability.

```
<canvas id="canvas" width="100" height="100">
```

```
  <!-- Provide an item that can be displayed if the browser does not support the canvas element -->
</canvas>
```

For details of the canvas element, refer to W3C Recommendation HTML5 - Section 4.11.4 “The canvas element” (www.w3.org/TR/html5/scripting-1.html#the-canvas-element).

(3) SVG

Images can be drawn using scalable vector graphics (SVG). By doing this, it is often possible to generate a smooth image with a small volume of data, a benefit from using vector graphics. The image quality hardly deteriorates even when the image is scaled up or down. The receiver needs to have enough computing power to draw SVG. Due attention should be paid to avoid imposing excessive processing on the receiver, which may cause operability to be undermined.

For details of the svg element, refer to W3C Recommendation HTML5 - Section 4.8.15 “SVG” (http://www.w3.org/TR/html5/svg-0.html#svg-0). For details of SVG, refer to W3C Recommendation “Scalable Vector Graphics (SVG) Tiny 1.2 Specification” (http://www.w3.org/TR/SVGTiny12/).

(4) WebGL

WebGL processes 3D graphics to render 2D graphics. It should be noted that some receivers may not support WebGL.

For details of WebGL, refer to WebGL - OpenGL ES 2.0 for the Web (http://www.khronos.org/webgl/).

2.16. Display of a moving image (movement or transformation of an element)

(1) CSS3 transitions

To display a moving image, such as an element that moves or is transformed, each element can be decorated with CSS3 transitions, etc.

For details, refer to W3C Recommendation CSS Transitions (http://www.w3.org/TR/css3-transitions/).

(2) Overwriting attributes of the target element by JavaScript codes

In addition to the above, a moving image can be displayed by overwriting and controlling attributes of the target element by JavaScript codes.

2.17. Display of a moving image (animation)

What is called animation, another way of displaying an element with movement, can be realized through the following methods.

(1) CSS3 animations

Movements can be decorated using CSS3 animations.

For details, refer to W3C Recommendation CSS3 Animations.

(http://www.w3.org/TR/css3-animations/).

(2) Overwriting the attributes of the target element by JavaScript codes

As in Section 2.16, it is also possible to display an element with movement by overwriting and controlling the attribute of the target element by JavaScript codes.

2.18. Page transition

2.18.1. Unload event

When a page transition occurs, an event shall be issued in accordance with the event model specification. An HTML application can obtain information about the event of a page being loaded or unloaded. These Specifications assume that load/unload events are issued. In particular, when the HTML application specified is to be replaced as a result of switching of broadcast signals, the operating HTML application is released. Before it is released, the event of unloading that application is implemented. Broadcast services require the running HTML application to be terminated quickly. Therefore, if the handling of the unload event takes some time, the processing of the running HTML application may be forced to terminate irrespective of whether or not the event handling has been completed and irrespective of the result of the event handling. The author of an HTML application should pay due attention to how and when to store data.

For details of event definition, refer to W3C Recommendation 6.1.5.2 Event handlers on elements, Document objects, and Window objects (http://www.w3.org/TR/html5/webappapis.html#event-handlers-on-elements,-document-objects,-and-window-objects).

2.18.2. The href attribute

When a link is clicked, the browser refers to the href attribute of an “a” element, and loads the HTML file of the specified link and causes page transition. The way the new page opens can be specified by specifying the target attribute.

To maintain the consistency in the page display of an HTML application, this Specification recommends that no target attribute be used.

It should be noted that how the browser handles a target attribute depends on the browser implementation, and so the use of the target attribute may result in inconsistency in page display of an HTML application.

For details, refer to W3C Recommendation HTML5 - Section 4.5.1 “The a element” (http://www.w3.org/TR/html5/text-level-semantics.html#the-a-element).

2.19. Key event processing

The KeyboardEvent interface defined in W3C can be used. Specifically, it is defined in W3C Working Draft “Document Object Model (DOM) Level 3 Events Specification” - Section 5.2.5 and B.1 (http://www.w3.org/TR/2012/WD-DOM-Level-3-Events-20120614/).
The definition of key codes added in this Specification is given in Section 3.1.6.

2.20. Focus navigation

Focus navigation is defined in W3C Recommendation HTML5 - Section 7.3.1 Sequential focus navigation and the tabindex attribute. It is possible to specify the style of the focus navigation, such as whether or not focus highlight display is used, the line width, and color. Specifically, the style can be specified using the outline property of a CSS pseudo focus class.
Example of CSS to remove focus display by the browser

```
: focus   { outline:   none }
```

It is possible to focus on an element explicitly by calling the focus() method that each element has. An HTML application can also move a focus by event handling through key operations or by an appropriate algorithm. It is recommended to achieve common user operation by developing a library of scripts.
For details, refer to W3C Recommendation HTML5 - Section 7.4 “Focus” (http://www.w3.org/TR/html5/editing.html#focus) and W3C Recommendation “CSS Basic User Interface Module Level 3” (CSS3 UI) – Section 7 “Outline properties” (http://www.w3.org/TR/css3-ui/#outline-properties).

2.21. Captions

2.21.1. How to present captions in a broadcast channel

The receiver presents the captions in the existing stream of the currently selected broadcast channel. The initial operation parameters of the ES and language identification (the value of Language_tag in the caption management data based on ARIB STD-B24 Volume Section 19.3.1) that should be referred to shall be specified using a param element in the broadcast audio/video object described in 2.12.1. For details of this parameter, refer to the broadcast object specification described in 3.2. However, if only a broadcast audio/video object element and a param element are set, the application engine does not control the presentation of captions. Captions shall be controlled via an API that is used to give instructions on caption presentation, described in 3.2.

2.21.2. Other types of captions

Other types of captions are to be specified later.

2.22. The iframe element

An HTML application can incorporate other HTML documents by using iframe elements. For details of the iframe element, refer to W3C Recommendation HTML5 - Section 4.7.2 “The iframe element” (http://www.w3.org/TR/html5/embedded-content-0.html#the-iframe-element).

The security model regarding basic sharing of elements and events between the parent and child HTML documents shall follow W3C recommendations. However, the use of broadcast resources using the extended APIs specified in Section 3.1 and the broadcast audio/video object specified in Section 3.2 shall follow the access control specified in Integrated Broadcast-Broadband Integration System Specifications IPTVFJ STD-0010 - Section 7.6 “Application boundary and broadcast resource access control.” When the obtained URL domain of the parent HTML document is different from that of the child document, there may be restriction on access to broadcast resources. Service operators and HTML application authors should pay due attention to this problem.

When an HTML application implements focus navigation using a focus() method and a key event handler, and when the focus is on an iframe element, the key event is handled by the DOM within the iframe, resulting in the parent DOM being unable to obtain the key event. It is recommended that the HTML application execution environment (browser) provide a means by which the user/operator of the application can return the focus from the DOM within the iframe to the DOM of the parent document.

2.23. VOD

2.23.1. Support of services that are based on the CDN-scope service approach specifications

To be specified later.

2.23.2. Support of services that are based on the Internet-scope service approach specifications

It is assumed in this document that an application based on this Specification refers to and presents a VOD service in which a video is delivered pursuant to IPTV Specification IPTVFJ STD-0007 “Internet Scope Service Approach Specification.” The application that presents this service is described in accordance with Chapters 3 and 4 of “Digital TV Network Functional Specifications - Streaming Functional Specifications - Browser Ver.1.2” (Networked Digital Television). The scope of the functions specified in these specifications that are covered by the browser shall be determined as operational rules.

2.23.3. Handling of services that use MPEG-DASH or HLS

If an application based on this Specification is to refer to and present a VOD service that is to be delivered using MPEG-DASH or HLS, Media Source Extensions (http://www.w3.org/TR/media-source/) and Encrypted Media Extensions (http://www.w3.org/TR/encrypted-media/) shall be applied to a video element as specified in W3C Recommendation HTML5 Section 4.7.6 “The video element” (http://www.w3.org/TR/html5/embedded-content-0.html#the-video-element). Details of this operation shall be specified in operational rules.

2.23.4. Support of other VOD services

To be specified later.

Chapter 3 Extended Technical Specification

3.1. Extended API specification

This section specifies the extended APIs that can be used by scripts in HTML documents that make up an application.

The functions provided by the extended APIs are a means of controlling application execution, a means of accessing information included in the broadcast signals being received, and a means of accessing functions provided by the receiver.

In general, extended API specifications have a strong impact on the realizable behavior of applications and on the kinds of services that can be provided. Therefore, the final scope and format of extended APIs should be carefully defined in separate operational rules after detailed study taking the actual service operations into consideration. The specifications described below are based on the receiver configurations and service scenarios assumed at the time when this Specification was compiled. When this Specification is to be updated or expanded, due attention should be paid to compatibility issues, in particular, backwards compatibility of receiver implementations.

3.1.1. ISDB resource reference object

The ISDB resource reference object is used in the interfaces specified in Section 3.1 as an argument or a return value to identify the relevant service, component or event.

```
dictionary ISDBResourceReference {
	attribute unsigned short original_network_id;
	attribute unsigned short transport_stream_id;
	attribute unsigned short service_id;
	attribute unsigned short content_id;
	attribute unsigned short event_id;
	attribute octet component_tag;
	attribute octet channel_id;
	attribute unsigned short module_id;
	attribute DOMString? module_name;
	attribute DOMString? resource_name;
};
```

The properties of this object mean the following.

original_network_id	Original network identifier
transport_stream_id	Transport stream identifier
service_id	Service identifier

content_id	Content identifier
event_id	Event identifier
component_tag	Component tag
channel_id	Value indicating a channel or a combination of channels within the audio component
module_id	Identifier of the module sent via DSM-CC data carousel
module_name	Name given to the module sent via DSM-CC data carousel
resource_name	Name to identify an entity included in the module sent via DSM-CC data carousel

It is not mandatory that all the properties are present in an object. Which properties should be present is determined by the context in which the object is used.

3.1.2. Application manager object

The application manager object provides an interface for application execution control.

It is provided as the applicationManager property of the navigator object built into the receiver. The constructor of the application manager object is not provided.

3.1.2.1. Interface definition

```
[NoInterfaceObject]
interface NavigatorApplicationManager {
        readonly attribute ApplicationManager applicationManager;
};

Navigator implements NavigatorApplicationManager;

[NoInterfaceObject]
interface ApplicationManager {
        Application? getOwnerApplication(optional Document document);
        void launchDataBroadcastingBrowser(ISDBResourceReference startup);
};
```

3.1.2.2. Method

getOwnerApplication		
Description	Returns the application to which the document identified by the argument *document* belongs	
Arguments	document	Object identifying the HTML document that wants to obtain information about the application it belongs to. When this argument is omitted, it is deemed that the object that identifies the HTML document that has executed this method is specified.
Return value	Application object that identifies the application whose information has been obtained. “Null” if no applicable application exists.	

launchDataBroadcastingBrowser		
Description	Launches the data broadcast browser. Control of the state of the application that has executed this method shall be determined for each service operation	
Arguments	startup	Specifies one of the following: ・Object identifying the document to be presented when the data broadcast browser is launched (startup document) ・Object identifying the component that includes the startup document ・Object identifying the service that includes the startup document When an object identifying a component is specified, it is deemed that the startup document specified in ARIB STD-B24 Part II Section 9.2.2 is specified. When an object identifying a service is specified, it is deemed that the object identifying the entry component of that service is specified.

How to handle each property in the argument *startup* of the launchDataBroadcastingBrowser method is described below.

1. When the application specifies a service

original_network_id	The Application must specify or omit all of these. When all are omitted, the application engine assumes that the current service is specified.
transport_stream_id	
service_id	
content_id	I*
event_id	I
component_tag	The application must always omit this property.
channel_id	I
module_id	I
module_name	I
resource_name	I

*I: It is recommended that the application omit this property. The application engine ignores this property even if this property is present.

2. When the application specifies a component

original_network_id	The application must specify or omit all of these. When all are omitted, the application engine assumes that the current service is specified.
transport_stream_id	
service_id	
content_id	I*
event_id	I
component_tag	The application must always specify this property.
channel_id	I
module_id	The application must always omit this property.
module_name	The application must always omit this property.
resource_name	I

*I: It is recommended that the application omit this property. The application engine ignores this property even if this property is present.

3. When the application specifies a document

original_network_id transport_stream_id service_id	The application must specify or omit all of these. When all are omitted, the application engine assumes that the current service is specified.
content_id	I*
event_id	I
component_tag	The application must always specify this property.
channel_id	I
module_id module_name	The application must always specify either one of these.
resource_name	The application may omit this property. When this property is omitted, it is deemed that the startup document specified in ARIB STD-B24 Part II Section 9.2.2 is specified. (However, if the module specified by module_id or module_name is not of entity format, this property is ignored.)

*I: It is recommended that the application omit this property. The application engine ignores this property even if this property is present.

3.1.3. Application object

The application object identifies an application. It is returned by the method getOwnerApplication of the application manager object, or by the method getApplications of the ApplicationInformationTable object. The constructor of the application object shall not be provided.

3.1.3.1. Interface definition

```
[NoInterfaceObject]
interface Application {
        readonly attribute DOMString type;
        readonly attribute unsigned long long organization_id;
        readonly attribute unsigned long long application_id;
        readonly attribute DOMString control_code;
        readonly attribute octet autostart_priority;
        void replaceApplication(
                unsigned long organization_id,
                unsigned long application_id,
                DOMString? uri);
        void destroyApplication();
        void exitFromManagedState(DOMString uri);
        ApplicationInformationTable getOwnerAIT();
        ApplicationBoundaryAndPermissionDescriptor
                getApplicationBoundaryAndPermissionDescriptor();
};

[NoInterfaceObject]
interface ApplicationBoundaryAndPermissionDescriptor {
        sequence<PermissionManagedArea>? getCurrentBoundary();
        void addPermissionManagedArea(PermissionManagedArea pma);
};

dictionary PermissionManagedArea {
        sequence<unsigned short>? permission;
        sequence<DOMString>? urls;
};
```

3.1.3.2. Properties

type	
Description	Application type of the given application. The value of this property is a character string specified as the value of the applicationType element in ARIB STD-B24 Volume 4 Section 6.3.

organization_id	
Description	Organization ID of the given application. For details of organization IDs, refer to ARIB STD-B24 Volume 4 Section 5.2.

application_id	
Description	Application ID of the given application. For details of application IDs, refer to ARIB STD-B24 Volume 4 Section 5.2.

control_code	
Description	Application control code of the given application. The value of this property is one of the character strings specified as the value of the controlCode element in ARIB STD-B24 Volume 4 Section 6.3.

autostart_priority	
Description	Autostart priority of the given application. For details of autostart priority, refer to ARIB STD-B24 Volume 4 Section 5.3.5.

3.1.3.3. Method

replaceApplication		
Description	Terminates the executed application, and launches the application identified by the argument	
Arguments	Organization_id	Organization ID of the application to be launched. For details of organization IDs, refer to ARIB STD-B24 Volume 4 Section 5.2.
	application_id	Application ID of the application to be launched. For details of application IDs, refer to ARIB STD-B24 Volume 4 Section 5.2.
	uri	URI identifying the location of the AIT. Operations that will be executed when “null” is specified shall be specified in operational

		rules.

destroyApplication	
Description	Terminates the executed application. The behavior of the application engine after the termination of the application follows the Integrated Broadcast-Broadband System Specifications IPTVFJ STD-0010 7.4.3.

exitFromManagedState		
Description	Makes a transition to a general application. Terminates the application that has executed this method, and makes a transition to the document specified by the argument in unmanaged state.	
Arguments	uri	Entry URL of the general application to which the transition is to be made

getOwnerAIT	
Description	Obtains the application information table (AIT) that controls the given application
Return value	Object identifying the obtained AIT

getApplicationBoundaryAndPermissionDescriptor	
Description	Obtains an object that identifies the application boundary and permission descriptor included in the given AIT. For details of the application boundary and permission descriptor, refer to ARIB STD-B24 Volume 4 Section 5.3.4
Return value	Object identifying the obtained application boundary and permission descriptor

getCurrentBoundary		
Description	Obtains the current boundary of the application	
Return value	Array describing the access permission managed area, which indicates the obtained boundary.The meaning of the property of each element is as follows. The array has no element if no access permission descripter is set for the given application and, additionally, no access permission managed area is added using the addPermissionManagedArea method.	
	permission	Array whose elements are a bit map indicating the given application's access permission to the access permission managed area identified in the urls property. This is "null" if the maximum permission is given.
	urls	Array whose elements are URL character strings indicating the access permission managed area. This is "null" if any location is included in this access permission managed area.

addPermissionManagedArea	
Description	Adds an application's access permission managed area. For details on access permission managed areas, refer to IPTVFJ STD-0010 Section 7.6. The execution of this method is equivalent to adding one loop of an access boundary and permission descriptor. The access permission managed area added using this method is initialized when the application that has made this addition is terminated.

Arguments	pma	permission	Array whose elements are a bit map indicating the given application's access permission to the access permission managed area that is to be added. This is "null" if the maximum permission is given. When an array with no element is specified, it is deemed that "null" is specified.
		urls	Array whose elements are URL character strings indicating an access permission managed area to be added. This is "null" if all other locations are to be specified.

3.1.4. ApplicationInformationTable object

The ApplicationInformationTable object identifies the application information table (AIT). It is returned by the getOwnerAIT method of the application object. The constructor of the ApplicationInformationTable object is not provided.

3.1.4.1. Interface definition

```
[NoInterfaceObject]
interface ApplicationInformationTable {
      sequence<Application> getApplications();
};
```

3.1.4.2. Method

getApplications	
Description	Obtains all applications that have been set in the given AIT
Return value	Array that contains application objects that indicate the applications set in the given AIT. The application objects are stored in the sequence in which the the applications appear in the AIT.

3.1.5. Capabilities object

The Capabilities object provides information about the scope of functions provided by the application engine and the receiver platform.

The capabilities object is provided as the capabilities property of the Navigator object that is built into the receiver. The constructor of the capabilities object is not provided.

3.1.5.1. Interface definition

```
[NoInterfaceObject]
interface NavigatorCapabilities {
      readonly attribute Capabilities capabilities;
};

Navigator implements NavigatorCapabilities;
```

```
[NoInterfaceObject]
interface Capabilities {
      boolean hasCapability(DOMString query, DOMString ... params);
};
```

3.1.5.2. Method

hasCapability		
Description	Obtains information about whether the application engine or the receiver platform has the function identified by the argument.	
Arguments	query	Character string identifying the function that is subject to query. The strings that can be specified and their meanings shall be defined as operational rules.
	params	Character string identifying supplementary information about the query. The information is determined in accordance with the string specified in the query. The string that is handed over and its meaning shall be defined as operational rules.
Return value	True if the function specified by the argument is present, or false if not.	

3.1.6. ReceiverDevice object

The ReceiverDevice object provides a means of accessing the function provided by the device in which the application engine is operating, or of accessing information managed by the device.

The ReceiverDevice object is provided as the receiverDevice property of the Navigator object built into the receiver. The constructor of the ReceiverDevice object is not provided.

3.1.6.1. Interface definition

```
[NoInterfaceObject]
interface NavigatorReceiverDevice {
      readonly attribute ReceiverDevice receiverDevice;
};

Navigator implements NavigatorReceiverDevice;

[NoInterfaceObject]
interface ReceiverDevice {
};
```

3.1.6.2. Methods

This section describes the specification of the methods provided by the ReceiverDevice object.

There may be some restrictions on how to call a method of the ReceiverDevice object depending on the way the receiver is implemented (such as the maximum number of parallel executions, and combinations of methods that cannot be executed simultaneously). If restrictions are present, their details shall be defined as operational rules.

3.1.6.2.1. Obtaining the receiver-unique identifier

```
partial interface ReceiverDevice {
        void getDeviceIdentifier(long type,
                        DeviceIdentifierCallback resultCallback);
};
callback DeviceIdentifierCallback = void (DOMString? identifier);
```

getDeviceIdentifier		
Description	Returns a receiver-unique identifier of the type approriate for the value of the argument *type*. Although details of the values that can be specified in the argument, and the string to be returned should be defined for each service operation, it is expected that they will be the same as those in getIRDID defined in ARIB TR-B14 Part III. Since the identifier returned by this function can be associated with personal information of the receiver user, the application creator should tpay due consideration to handling the identifier returned by this function.	
Arguments	type	Value identifying the type of the receiver-unique identifier to be obtained
	resultCallback	Function to be called when the processing is completed

callback DeviceIdentifierCallback		
Arguments	identifier	Obtained receiver-unique identifier, or “null,” which indicates that the attempt to obtain the identifier has failed.

3.1.6.2.2. Channel selection

```
partial interface ReceiverDevice {
        void tuneTo(ISDBResourceReference service_ref,
                        TuneToResultCallback? resultCallback,
                        optional TuneToOptions options);
};
callback TuneToResultCallback = void (ISDBResourceReference service_ref);

dictionary TuneToOptions {
        attribute boolean unbound = false;
};
```

tuneTo	
Description	Changes the service being received. Even if the specified service is the same as the service being received (except for a case where “true” is set in the argument *unbound*, which makes these regarded to be the same), the channel selection operation is executed. If there is any broadcast video and/or audio present, they continue to be broadcast without interruption. Except for a case where true is specified for the argument *unbound* and the application

	continues to be executed, the application engine may terminate the application before this funciton sends back a return value.		
Arguments	service_ref	Object identifying the service to be changed	
	resultCallback	Function to be called when the processing is completed. “Null” if this is not required.	
	options		
		unbound	If the service identified by service_ref is not the currently received service, and if this property is true, the service identified by service_ref is regarded as the same as the currently received service, and continued execution of the application is attempted. After the service is changed, the operation continues if signals that allow the application to continue to be executed are received in this service. Otherwise, the operation is terminated.

callback TuneToResultCallback		
Arguments	service_ref	Object identifying the service that has become the current service after channel selection. Or “null,” which indicates a failure in channel selection.

Handling of each property of the argument *service_ref* of the tuneTo method

original_network_id	The application must either specify or omit all of these. If all of them are omitted, the application engine assumes that the current service has been specified.
transport_stream_id	
service_id	
content_id	I*
event_id	I
component_tag	I
channel_id	I
module_id	I
module_name	I
resource_name	I

*I: It is recommended that the application omit this property. The application engine ignores this property even if this property is present.

Handling of each property of the argument *service_ref* of TuneToResultCallback

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	—**
event_id	—
component_tag	—
channel_id	—
module_id	—

module_name	—
resource_name	—

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present.

3.1.6.2.3. Obtaining information about the event information table (EIT) [current/following]

```
partial interface ReceiverDevice {
        void getCurrentEventInformation(
                        CurrentEventInformationCallback resultCallback);
};
callback CurrentEventInformationCallback = void (CurrentEventInformation info);

dictionary CurrentEventInformation :   ISDBResourceReference {
        attribute Date start_time;
        attribute   long long duration;
        attribute   DOMString name;
        attribute   DOMString desc;
        attribute   unsigned short f_event_id;
        attribute Date     f_start_time;
        attribute   long long f_duration;
        attribute   DOMString f_name;
        attribute   DOMString f_desc;
};
```

getCurrentEventInformation		
Description	Returns information about the EIT [current/following]	
Arguments	resultCallback	Function to be called when the processing is completed

callback CurrentEventInformationCallback			
Arguments	info	Information about the current event obtained as a result of the processing	
		start_time	Value representing start_time in the event information section of the EIT [current] in Date format
		duration	Value representing duration in the event information section of the EIT [current] in milliseconds
		name	Character string representing the value of the program name (event_name_char) in the short event descriptor of the EIT [current]
		desc	Character string representing the program description (text_char) in the short event descriptor of the EIT [current]
		f_event_id	Value representing event_id in the event information section of the EIT [following]
		f_start_time	Value representing start_time in the event information section of the EIT [following] in Date format
		f_duration	Value representing duration in the event information section of

			the EIT [following] in milliseconds
		f_name	Character string representing the value of the program name (event_name_char) in the short event descriptor of the EIT [following]
		f_desc	Character string representing the program description (text_char) in the short event descriptor of the EIT [following]

Handling of each property in the argument *info* of CurrentEventInformationCallback

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	—**
event_id	M
component_tag	—
channel_id	—
module_id	—
module_name	—
resource_name	—

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present.

3.1.6.3. Determination of event sharing

```
partial interface ReceiverDevice {
	boolean isCommonEvent(
			ISDBResourceReference service_ref);
};
```

isCommonEvent		
Description	Determines whether the event is shared between the specified service and the current service	
Arguments	service_ref	Object identifying the service that is subject to this determination
Return value	True if the event is shared. False if not.	

Handling of each property in the argument *service_ref* of isCommonEvent

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	—**
event_id	—
component_tag	—
channel_id	—
module_id	—
module_name	—

resource_name	—

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present.

3.1.7. The EIT search manager object

The EIT search manager object provides a means of accessing the EIT schedule information managed by the device on which the application engine is running. The EIT schedule information is schedule information about the own or other transport stream and event both specified in ARIB STD-B10. To obtain information about the current/following event, use the getCurrentEventInformation() method.

The EITsearch manager object is provided as the eitSearchManager property of the Navigator object built into the receiver. The constructor of the EITsearch manager object is not provided.

3.1.7.1. Interface definition

```
[NoInterfaceObject]
interface NavigatorEITSearchManager {
        readonly attribute EITSearchManager eitSearchManager;
};

Navigator implements NavigatorEITSeachManager;

[NoInterfaceObject]
interface EITSearchManager {
        function onEITScheduleUpdate();
        function onEITSearch(EITSeach search, Integer status);
        EITSearch createSearch();
};
```

3.1.7.2. Properties

This section describes the specifications of the properties provided by the EIT search manager object.

onEITScheduleUpdate()	
Description	Event handler that is executed when schedule information about the own and other transport stream and event specified by ARIB STD-B10 is updated. It should be noted that, since the timing and frequency of this update depends on the implementation of the receiver, they may not be the same as those of broadcast signals.

onEITSearch()		
Description	Event handler associated with the search processing of the EIT	
Arguments	search	Applicable EITSearch object
	status	Value indicating search status: 0: Search has been completed. Indicates that search has been completed and the result can be obtained.

		3: Search has been terminated. This appears when the SearchResults.abort() method is called or when search conditions are changed. 4: Indicates that search has not been completed due to lack of resources (e.g. No search results can be displayed due to memory shortage)

3.1.7.3. Method

This section shows the specification of the method provided by the EIT search manager object.

createSearch	
Description	Generates the EITSearch object
Return value	Generated EITSearch object. “Null” if the generation has failed.

3.1.8. EITSearch object

EITSearch object represents search for EIT schedule information. It is returned by the createSearch method of the EIT search manager object. The constructor of the EITSearch object is not provided.

The relation and state transition between the EITSearch object and the associated SearchResults object are described in Fig. 3-1 and Table 3-1.

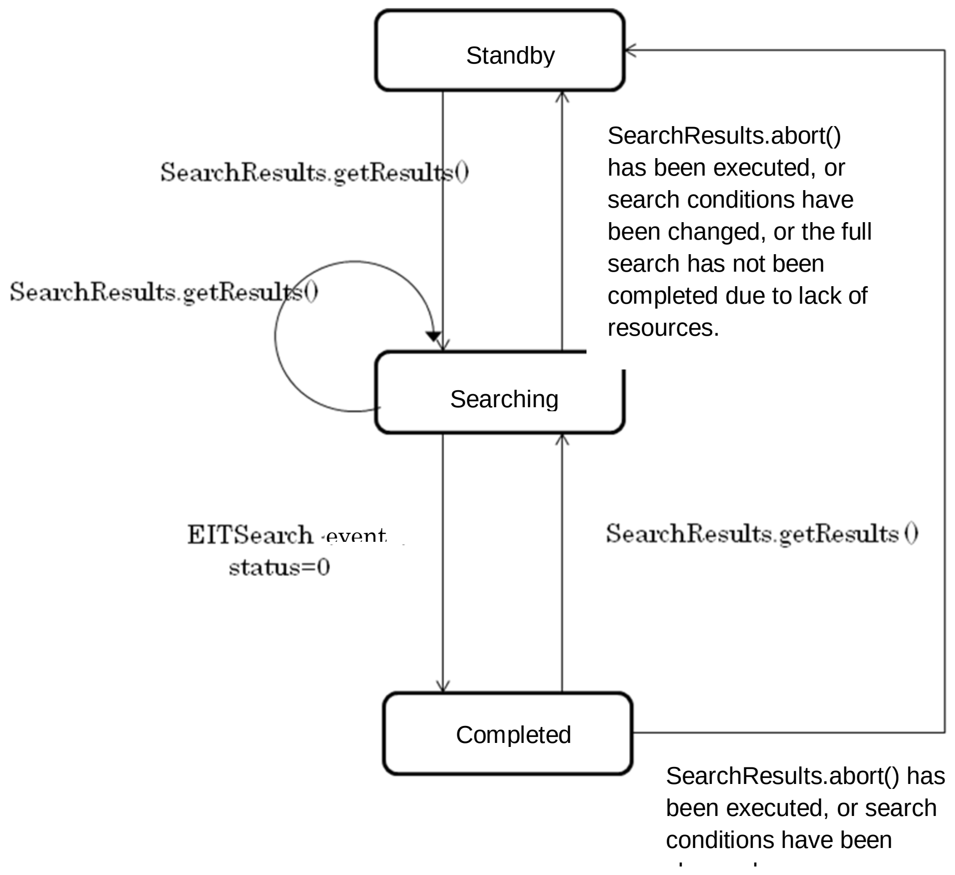

Standby
SearchResults.getResults()
SearchResults.abort() has been executed, or search conditions have been changed, or the full search has not been completed due to lack of resources.
SearchResults.getResults()
Searching
EITSearch event status=0
SearchResults.getResults ()
Completed
SearchResults.abort() has been executed, or search conditions have been

Fig. 3-1 State transition diagram for the EITSearch object (reference)

Table 3-1 States of the EITSearch object

State	Specification
Standby	Search is in standby mode. No result can be obtained from the application. This is the initial state. The application can set or modify search conditions in this state. The value “undefined” must be returned as a result of SearchResults.item(). “0” must be returned as the values of the length and totalSize properties of the SearchResults object. When put in standby mode, the receiver must discard the search result. When the SearchResults.getResults() method is called, the state must change to “searching.”
Searching	State in which the receiver has obtained the result but the application cannot access it. When the receiver has not yet obtained all search results, it is continuing to

	obtain the requested search result. When the EIT schedule is updated and a new version is detected, and when the object is in this state, the receiving entity can keep either the previous or the new version but should not mix them up. The value “undefined” must be returned as a result when SearchResults.item() is called in this state. When search formula is changed (i.e., a new query is set), the current search is terminated and the state is changed to “standby.” The receiver must issue an EITSearch event with status=3. When all search has completed, the receiver must issue an EITSearch event with status=0, and change the state to “completed.” When search cannot be continued due to shortage of resources or other reasons, the receiver must issue an EITSearch event with status=4, and change the state to “standby.” When the SearchResults.getResults() method is called, the process of obtaining a search result must be terminated, and the receiver must try a new search with a new search condition.
Completed	The application can access the search result. It can obtain it by calling the SearchResults.item() method. This value cannot be changed, even when EIT schedule is changed, until the next time SearchResults.getResults() is called. When the EIT schedule is updated and a new version is detected, an EITUpdate event is issued. When SearchResult.getResults() is called later, the result obtained must be one based on the updated EIT. When SearchResults.getResults() is called, the state must be changed to “searching.” When the search condition is changed, the current search result must be discarded and the state must be changed to “standby.” The receiver issues an EITSearch event with status=3.

3.1.8.1. Interface definition

```
[NoInterfaceObject]
interface EITSearch {
        readonly SearchResults result;
        void setQuery(Query query);
        Query createQuery(DOMString field, Integer comparison, DOMString value);
};
```

3.1.8.2. Property

Result	
Description	SearchResults object identifying the result.

3.1.8.3. Method

setQuery		
Description	Sets the Query object created by the createQuery method to the EITSearch object. When this method is called, all search result obtained during the standby or searching states must be discarded. In other words, the processing equivalent to the abort() method of the SearchResults object is carried out.	
Arguments	query	Query object to be set
Return value	None	

createQuery		
Description	Compares the EIT schedule value identified in the argument *field* with the value of the argument *value* in accordance with the rule defined in the argument *comparison*, and creates a Query object to be returned when the search is successful.	
Arguments	field	Specifies an item that is subject to comparision. This item is startTime, name, eventID, serviceID, description, or item_desc. These correspond, respectively, to the start_time property, the name property, the event_id property, the service_id property, the description property, and the item_desc property of the EITSchedule object. The value of name (string) is case-insensitive. (Note 1)
	comparison	One of the following is specified 0: Search is successful if the value specified in *value* matches the value of the EIT item specified in *field*. 1: Search is successful if the value specified in *value* does not match the value of the EIT item specified in *field*. 2: Search is successful if the value specified in *value* is greater than the value of the EIT item specified in *field*.. 3: Search is successful if the value specified in *value* is greater than or equal to the value of the EIT item specified in *field*. 4: Search is successful if the value specified in *value* is smaller than the value of the EIT item specified in *field*. 5: Search is successful if the value specified in *value* is smaller than or equal to the value of the EIT item specified in *field*. 6: Search is successful if the value specified in *value* completely or partially matches the string of the EIT item specified in *field*. The string is case-insensitive. (Note 1) Search is unsuccessful if the value specified in *value* or *field* is not numerical when a value of 0-5 is specified in *comparison*.
	value	Value referred to in the argument *comparison*. When a non-string value is

		entered in the argument and is to be converted into a string, this conversion must follow the type conversion rule of ECMAScript (Refer to ECMA-262 Section 9).
Return value	Created Query object. Or, “null,” which indicates failure.	

(Note 1) This is applicable only to the characters defined in the character repertoire of ISO/IEC 10646 Basic Latin. In particular, it should be noted that double-byte alphanumeric characters and space are distinguished from single-byte characters.

Acceptable combinations of the values in the *field* and *comparison* properties are shown in Table 3-2.

Table 0-2 Acceptable combinations of the values in the *field* and *comparison* properties in createQuery

comparison ＼ field	startTime	name	eventID	serviceID	description	item_desc
0	○	×	○	○	×	×
1	○	×	○	○	×	×
2	○	×	○	○	×	×
3	○	×	○	○	×	×
4	○	×	○	○	×	×
5	○	×	○	○	×	×
6	×	○	×	×	○	○

○: This can be specified.

×: This cannot be specified. If this is specified, the method fails.

3.1.9. Query object

The Query object identifies the condition in which the application searches for EIT schedule information. This object is returned by the createQuery method of the EITSearch object. The constructor of the Query object is not provided.

3.1.9.1. Interface definition

```
[NoInterfaceObject]
interface Query {
        Query and(Query query);
        Query not();
};
```

3.1.9.2. Method

and		
Description	Generates a search formula that is a logical product of the Query object of the argument and the given Query object, and returns the result as a Query object	
Arguments	query	Query object used in the logical product
Return value	Query object that represents the generated search formula	

not	
Description	Generates a new Query object that is logical NOT of the given Query object
Return value	Query object that represents the generated search formula.

3.1.10. SearchResults object

The SearchResults object represents the search result information. It is generated by the property *result* of the EITSearch object. The constructor of the SearchResults object is not provided.

3.1.10.1. Interface definition

```
[NoInterfaceObject]
interface SearchResults {
        readonly Integer length;
        readonly Integer offset;
        readonly Integer totalSize;
        EITSchedule item(Integer index);
        void getResults(Integer offset, Integer count);
        void abort();
};
```

3.1.10.2. Properties

length	
Description	Number of items in the currently obtained result. If search is still ongoing and no result has been determined, “0” is returned until an EITSearchEvent event with status=0 occurs.

offset	
Description	Offset value for the total number of items in the search result

totalSize	
Description	Number of items in the search result. If search is still onging and no value has been determined, the value is “undefined.” The value is obtained when the getResults() method is called and the application is notified that a result has been obtained through an EITSearchEvent.

3.1.10.3. Methods

item		
Description	Returns an item about the currently valid array. If the result for the argument *index* is not determined, returns “undefined.”	
Arguments	index	Index to the result array
Return value	EITSchedule object that represents the result	

getResults		
Description	Executes search and obtains a result that matches the Query object that has been set. This processing must be asynchronous. The result is determined when an EITSearchEvent event with status=0 has occurred. While this method can be called repeatedly each time the EIT schedule is updated, the result of the last call of this method is valid. Therefore, if the EITschedule is updated after this method has been called, this update will not be reflected in the result.	
Arguments	offset	Number of items of the obtained data that are skipped. When “0” is specified, the result shall be returned from the start of the result.
	count	Number of items of result to be obtained
Return value	None	

abort	
Description	Terminates all searches. Items obtained so far are discarded. In other words, the value of the length property becomes “0”, and “undefined” shall be returned for any call of item()
Return value	None

3.1.11. EITSchedule object

The EITSchedule object represents a part of the event information table (EIT). It is returned by the *item* method of the SearchResults object.

The constructor of the EITSchedule object is not provided.

The encoding of the string of each property shall be defined as operational rules.[1]

3.1.11.1. Interface definition

```
[NoInterfaceObject]
interface EITSchedule{
        unsigned short event_id;
        unsigned short service_id;
```

[1] The data used in the EIT schedule object is encoded in an 8-level code. To reflect it on each property, code conversion is necessary. Rules for code conversion shall be defined as operation rules.

```
        unsigned short transport_stream_id;
        unsigned short original_network_id;
        Date start_time;
        long long duration;
        DOMString       name;
        DOMString       description;
        Boolean   free_CA_mode;
        unsigned short parental_rate;
        unsigned short service_type;
        DOMString service_provider_name;
        DOMString       service_name;
        DOMString       item_name;
        DOMString       item_desc;
};
```

3.1.11.2. Property

event_id	
Description	Value identifying event_id in the event information section of the EIT

service_id	
Description	Value identifying service_id in the event information section of the EIT

transport_stream_id	
Description	Value identifying transport_stream_id in the event information section of the EIT

original_network_id	
Description	Value identifying original_network_id in the event information section of the EIT

start_time	
Description	Value identifying start_time in the event information section of the EIT in date type

duration	
Description	Value identifying duration in in the event information section of the EIT in milliseconds

name	
Description	Character string identifying the value of the program name (event_name_char) of the short event descriptor

description	
Description	Character string identifying the value of event name of the short event descriptor (text_char)

free_CA_mode	
Description	Value identifying free_CA_mode in the event information section of the EIT in Boolean form

parental_rate	
Description	Value identifying age restriction rating (rating) of the parental rate descriptor

service_type	
Description	Value identifying the value of the service format type (service_type) of the service descriptor

service_provider_name	
Description	Character string identifying the value of the service provider (*char* following service_provider_name_length) of the service descriptor

service_name	
Description	Character string identifying the value of the program channel name (*char* following service_name_length) of the service descriptor

item_name	
Description	Character string identifying the value of the item name field (item_description_char) of the extended event descriptor

item_desc	
Description	Character string identifying the value of the item description field (item_char) of the extended event descriptor

3.1.12. Stream event target object

Stream event target provides a means by which the application uses an event sent in broadcast signals.

Interface definition

```
partial interface ReceiverDevice {
        readonly attribute StreamEventTarget streamEvent;
};

[NoInterfaceObject]
interface StreamEventTarget {
        boolean sourceIs(DOMString name);
};
```

sourceIs	
Description	Identifies the source of the broadcast signal associated with the application

Arguments	name	Character string identifying the source of the broadcast signal. This string shall be specified as operational rules. A string that can be specified in the argument *source* of the getContentSource function, which is an extended broadcast function defined in ARIB STD-B24 Part II Section 7.6.6.4, is assumed.
Return value	True is returned when the source of the broadcast signal associated with the application is what is specified in the argument *name*. False is returned otherwise.	

3.1.12.1. Reception of generic event messages

This section specifies the interface with which the application uses generic event messages that are based on ARIB STD-B24 Part III Section 7. Although this interface mainly assumes that the application makes shared use of generic event messages that are used for data broadcast, the application can also use generic event messages that are sent to the application in a dedicated ES.

It is assumed that generic event messages based on an extended version of the existing specification or a new specification will become available in the future and that applications will be able to use them. The interface for such cases will be specified in the future.

Interface definition

```
partial interface StreamEventTarget {
        void addGeneralEventMessageListener(
                GeneralEventMessageListenerParams param,
                GeneralEventMessageListener listener);
        void removeGeneralEventMessageListener(
                GeneralEventMessageListenerParams param,
                optional GeneralEventMessageListener? listener);
};
callback GeneralEventMessageListener = void (GeneralEventMessage msg);

dictionary GeneralEventMessageListenerParams {
        attribute ISDBResourceReference es_ref;
        attribute unsigned short message_group_id;
        attribute octet message_id;
        attribute octet message_version;
};

dictionary GeneralEventMessage {
        attribute ISDBResourceReference es_ref;
        attribute unsigned short message_group_id;
        attribute octet message_id;
        attribute octet message_version;
        attribute DOMString? private_data_byte;
};
```

addGeneralEventMessageListener

Description	Registers the event listener of a generic event message		
Arguments	param	es_ref	Object identifying the ES to be monitored
		message_group_id	Message group identifier of the applicable event message. The value has the same meaning as that of the message_group_id attribute specified in ARIB STD-B24 Part II Section 5.3.20.1.
		message_id	Message identifier of the applicable event message. The value has the same meaning as that of the message_id attribute specified in ARIB STD-B24 Part II Section 5.3.20.1.
		message_version	Message version of the applicable event message. The value has the same meaning as that of the message_version attribute specified in ARIB STD-B24 Part II Section 5.3.20.1.
	listener	Function that should be called when an event message that satisfied the relevant condition has been received and the ignition time mentioned in the applicable event message arrives. However, if event messages with the same message identifier are received several times, this fuction is executed only when the message version of the given event message received the second or later time is different from that of the previously received event message.	

removeGeneralEventMessageListener			
Description	Removes the event listener of a generic event message. The event listener that satisfies the specified monitoring condition is removed.		
Arguments	param	es_ref	Object identifying the ES to be monitored
		message_group_id	Message group identifier of the applicable event message
		message_id	Message identifier of the applicable event message
		message_version	Message version of the applicable event message
	listener	Event listener to be removed. When this is omitted, all event listeners that satisfy the condition specified in the argument *param* are removed.	

callback GeneralEventMessageListener			
Arguments	msg	Information about the ignited event message	
		es_ref	ES that has sent the applicable event message
		message_group_id	Message group identifier of the applicable event message
		message_id	Message identifier of the applicable event message
		message_version	Message version of the applicable event message
		private_data_byte	Private data byte of the applicable event message

Handling of each property in the argument *param* of the addGeneralEventMessageListener method

es_ref.original_network_id	The application must either specify or omit all of these. When all of these are omitted, the application engine assumes that the current
es_ref.transport_stream_id	

es_ref.service_id	service is specified.
es_ref.content_id	I*
es_ref.event_id	I
es_ref.component_tag	The application may either specify or omit this property. When this property is omitted, the application engine assumes that the entry component of the specified service is specified.
es_ref.channel_id	I
es_ref.module_id	I
es_ref.module_name	I
es_ref.resource_name	I
message_group_id	The application may specify or omit this property. The value that can be specified is “1”. How the application engines handles any value other than “1” is to be specified later. If this property is omitted, the application engine assumes that “1” is specified.
message_id	The application may specify or omit this property. When this property is omitted, the application engine assumes that any event message has the appropriate message identifier.
message_version	When the application omits message_id property, the application engine ignores this property. Otherwise, the application may either specify or omit this property. When this property is omitted, the application engine assumes that any event message has the appropriate message version.

*I: The Application engine ignores this property even if this property is present. It is recommended that the application omits this property.

Handling of each property in the argument *param* of the removeGeneralEventMessageListener method

es_ref.original_network_id	The application must either specify or omit all of these. When all of these are omitted, the application engine assumes that the current service is specified.
es_ref.transport_stream_id	
es_ref.service_id	
es_ref.content_id	I*
es_ref.event_id	I
es_ref.component_tag	○ (default)**
es_ref.channel_id	I
es_ref.module_id	I
es_ref.module_name	I
es_ref.resource_name	I
message_group_id	The application must either specify or omit all of these. When all of these are omitted, the application engine assumes that the current service is specified.
message_id	○ (match any)***
message_version	○ (match any)

*I: The Application engine ignores this property even if this property is present. It is recommended that the application omits this property.

**○ (default): The application may specify or omit this property. When this property is omitted, the application engine assumes that the entry component of the specified service is specified.

***○ (match any): The application may specify or omit this property. When this property is omitted, the application engine assumes that any event message satisfies the relevant condition.

Handling of each property in the argument *msg* in the GeneralEventMessageListener

es_ref.original_network_id	M*
es_ref.transport_stream_id	M
es_ref.service_id	M
es_ref.content_id	—**
es_ref.event_id	—
es_ref.component_tag	M
es_ref.channel_id	—
es_ref.module_id	—
es_ref.module_name	—
es_ref.resource_name	—
message_group_id	M
message_id	M
message_version	M
private_data_byte	M

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present.

3.1.12.2. Reception of a timer event based on NPT

This section specifies the interface used to handle a timer event that is based on normal play time (NPT). NPT becomes usable after it is associated with STC by an NPT reference descriptor, which is specified in ARIB STD-B24 Part Ⅲ 7. Therefore, an interface through which the application knows that an NPT reference descriptor has been received and has associated NPT with STC is also specified. This interface assumes that the application makes shared use of the NPT that is used for data broadcast. However, the application can also use NPT even when a dedicated ES is assigned for the application.

Interface definition

```
partial interface StreamEventTarget {
        void addNPTReferenceMessageListener(
                ISDBResourceReference es_ref,
                NPTReferenceMessageListener listener);
        void removeNPTReferenceMessageListener(
                ISDBResourceReference es_ref,
                optional NPTReferenceMessageListener? listener);
};
callback NPTReferenceMessageListener = void (ISDBResourceReference es_ref);

partial interface StreamEventTarget {
        unsigned long setAlarmByNPT(
                ISDBResourceReference es_ref,
                unsigned long long npt_value,
                NPTAlarmHandler handler);
        void unsetAlarmByNPT(
                unsigned long handle);
};
callback NPTAlarmHandler = void (
        ISDBResourceReference es_ref,
        unsigned long long npt_value);

partial interface StreamEventTarget {
        unsigned long long getNPT(ISDBResourceReference es_ref);
};
```

addNPTReferenceMessageListener		
Description	Registers an event listener that is executed when the function for using NPT becomes usable	
Arguments	es_ref	Object identifying the ES to be monitored
	listener	Function that is called when an NPT reference descripter has been received and has associated NTP with STC and a function that uses NPT becomes usable. If the application is in the relevant state at the time when this method is executed, the function specified here is executed without waiting for reception of the next NPT reference descriptor. After this function has been executed while the application is already in the relevant state, this function is not executed again, even if a new NPT reference descriptor is received.

removeNPTReferenceMessageListener		
Description	Removes the event listener registered in addNPTReferenceMessageListener	
Arguments	es_ref	Object identifying the ES to be processed
	listener	Event listener to be removed. When this is omitted, all event listeners registered for the ES specified in the argument *es_ref* are removed

callback NPTReferenceMessageListener		
Arguments	es_ref	ES for which the function for using NPT has become usable.

setAlarmByNPT		
Description	Registers the processing to be executed at the specified NPT time. If the specified NPT time has already passed by the time when this method is executed, the function specified in the handler is executed immediately.	
Arguments	es_ref	Object identifying the ES to be registered
	npt_value	NPT time at which the processing is to be executed
	handler	Function to be executed when NPT time specified in npt_value in the ES specified in es_ref has arrived

Return value	Handle identifying the registered processing

unsetAlarmByNPT		
Description	Cancels the registration made using setAlarmByNPT	
Arguments	handle	Handle identifying what is cancelled. The value returned by setAlarmByNPT is specified.

callback NPTAlarmHandler		
Arguments	es_ref	ES specified as the target when setAlarmByNPT was executed
	npt_value	NPT time specified when setAlarmByNPT was executed

getNPT		
Description	Obtains NPT time	
Arguments	es_ref	Object identifying ES to be processed
Return value	NPT time	

Handling of each property in the argument *es_ref* of addNPTReferenceMessageListener, removeGeneralEventMessageListener, and setAlarmByNPT, getNPT

original_network_id	The application must either specify or omit all of these. When all of these are omitted, the application engine assumes that the current service is specified.
transport_stream_id	
service_id	
content_id	I*
event_id	I
component_tag	The application may specify or omit this property. When this property is omitted, the application engine assumes that the entry component of the specified service is specified.
channel_id	I
module_id	I
module_name	I
resource_name	I

*I: The application engine ignores this property even if this property is present. It is recommended that the application omits this property.

Handling of each property in the argument *es_ref* of NPTReferenceMessageListener and NPTAlarmHandler

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	—**
event_id	—
component_tag	M
channel_id	—

module_id	—
module_name	—
resource_name	—

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present

3.1.12.3. Reception of event update notice

Interface definition

```
partial interface StreamEventTarget {
        void addEventIDUpdateListener(
                EventIDUpdateListener listener);
        void removeEventIDUpdateListener(
                optional EventIDUpdateListener listener);
};
callback EventIDUpdateListener = void (ISDBResourceReference event_ref);
```

addEventIDUpdateListener		
Description	Registers the event listener associated with the update of event ID	
Arguments	listener	Function to be called when event ID is updated in the service to which the application belongs

removeEventIDUpdateListener		
Description	Removes the event listeners registered using addEventIDUpdateListener	
Arguments	listener	Event listener to be removed. When this is omitted, all registered event listeners are removed

callback EventIDUpdateListener		
Arguments	event_ref	Object identifying the event after update

Handling of each property in the argument *event_ref* in EventIDUpdateListener

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	—**
event_id	M
component_tag	—
channel_id	—
module_id	—
module_name	—
resource_name	—

*M: The application engine must always set this property.

**—: It is recommended that the application engine does not allow this property to be present.

3.1.12.4. Reception of AIT update notice

This section specifies the interface used to receive AIT update notices. The AIT whose update is watched through this interface is an AIT transmitted over the broadband signal. Its details shall be specified in operational rules.

Interface definition

```
partial interface StreamEventTarget {
        void addAITUpdateListener(
                AITUpdateListener listener);
        void removeAITUpdateListener(
                optional AITUpdateListener listener);
};

callback AITUpdateListener = void (ApplicationInformationTable? ait);
```

addAITUpdateListener		
Description	Registers an event listener that is used to check for updates to an AIT transmitted over the broadband signal	
Arguments	listener	Function that should be called when an AIT being watched through this interface is updated. When the transimission of the AIT is stopped, or conversely when the AIT that has been stopped begins to be transmitted again, it is deemed that the AIT has been updated. If the given AIT is an AIT that controls the application that executes this method, the control of the application specified by this AIT takes precedence.

removeAITUpdateListener		
Description	Removes the event listener that has been registered using the addAITUpdateListener method	
Arguments	listener	Event listener that needs to be removed. If this argument is omitted, all registered event listeners are removed.

callback AITUpdateListener		
Arguments	ait	Object that indicates the updated AIT. This is “null” when this method is called as the result of the transmission of the AIT having been stopped

3.1.12.5. Reception of other events

To be specified later.

3.1.13. Interface for sharing functions with data broadcast browsers

This section specifies an interface that allows an application based on this Specification to use some of the functions provided by data broadcast browsers that are specified in ARIB STD-B24.

3.1.13.1. Interface definition

```
[NoInterfaceObject]
interface NavigatorBMLCompat {
    readonly attribute BMLCompatObject bmlCompat;
};
Navigator implements NavigatorBMLCompat;

[NoInterfaceObject]
interface BMLCompatObject {
    readonly attribute BMLBrowserPseudoObject browserPseudo;
};

[NoInterfaceObject]
interface BMLBrowserPseudoObject {
};
```

3.1.13.2. Methods

3.1.13.2.1. Access to NVRAM

```
partial interface BMLBrowserPseudoObject {
    signed long writePersistentArray(
        DOMString filename, DOMString structure, sequence<any> data);
    sequence<any> loadPersistentArray(
        DOMString filename, optional DOMString structure);
};
```

writePersistentArray		
Description	Writes data specified in the argument *data* to the area specified in the argument *filename* in the format specified by the argument *structure*	
Arguments	filename	Character string identifying the area into which data is to be written. This is specified in the format specified in ARIB TR-B14 Part III Subpart 2 Sections 5.2.3 to 5.2.5 and ARIB TR-B15 Part I Subpart 3 Sections 8.2.1 to 8.2.2
	structure	Character string identifying data storage format. This follows the specification of the argument structure for writePersistentArray defined in ARIB STD-B24 Part II Section 7.6.5.1
	data	Data to be written. This will follow the specification of the argument *structure* for writePersistentArray of ARIB STD-B24 Part II Section 7.6.5.1.
Return value	Value identifying the processing result. This is the same as the return value of writePersistentArray specified in ARIB STD-B24 Part II Section 7.6.5.1	

readPersistentArray	
Description	Reads data from the area specified in the argument *filename* in the format specified in the argument *structure*

Arguments	filename	Character string identifying the area from which data is to be read. This is specified in the format specified in ARIB TR-B14 Part III Subpart 2 Sections 5.2.3 to 5.2.5 and in ARIB TR-B15 Part I Subpart 3 Sections 8.2.1-8.2.2.
	structure	Character string identifying the data storage format. It follows the specification of the argument *structure* for readPersistentArray of ARIB STD-B24 Part II Section 7.6.5.1
Return value	Array read out as a result of the processing, or, "null," which indicates failure.	

3.1.13.2.2. Obtaining information about the viewer resident area

To obtain information about the viewer resident area, use the readPersistentArray method specified in 3.1.13.2.1. The arguments and return value are as follows.

readPersistentArray		
Description	Returns information that is set in the receiver in accordance with the string specified in the argument *filename* as the return value	
Arguments	filename	Character string identifying the type of information to be obtained. Specify one of the strings listed as <regiontype> in ARIB TR-B14 Part III Subpart 2 Section 5.2.7 Table 5-2.
	structure	This is always omitted. The application engine assumes that a "Field type" in ARIB TR-B14 Part III Subpart 2 Section 5.2.7 Table 5-2 is specified in accordance with the string specified in filename.
Return value	Obtained area information. The information shall be returned in the format specified in the "field type" in the above-mentioned table depending on the string specified in the argument *filename*.	

3.1.13.2.3. Access to Greg

Greg is a memory area within the receiver specified in ARIB STD-B24 Part II Section 7.6.16. The following interface is specified as a means of accessing (reading or writing) this area.

```
partial interface BMLBrowserPseudoObject {
	attribute sequence<DOMString> Greg;
};
```

Greg	
Description	Character string array identifying Greg specified in ARIB STD-B24 Part II Section 7.6.16. The number of array elements and the length of the string of each element also follows the above specification.

3.1.14. Application operation interface assuming the use of a TV remote controller unit.

This section specifies an event interface that assumes that the application is operated using a standard remote control unit of a TV receiver[2].

3.1.14.1. Interface specification

The application engine based on this Specification shall provide the KeyboardEvent interface specified in Sections 5.2.5 and B.1 of W3C Recommendation "Document Object Model (DOM) Level 3 Events Specification" (http://www.w3.org/TR/2012/WD-DOM-Level-3-Events-20120614/). However, it is not mandatory to provide all the functions of this interface.

3.1.14.2. Keyboard events that must be supported by the application engine

It is recommended that the application engine generate keydown, keyup and keypress in accordance with W3C Recommendation DOM3 Events Section 5.2.5.1.

3.1.14.3. Handling of implementation-dependent keyboard events

Depending on its implementation, the application engine may generate keyboard events that are not specified in 3.1.14.2. It is recommended that the application ignore such keyboard events as appropriate. However, this is not meant to prevent the application from using these keyboard events deliberately.

3.1.14.4. Key Code

The application shall use the keyCode property when extracting the key code from a keyboard event. Irrespective of what is specified in W3C Recommendation DOM3 Events, the application engine provides the constants listed below as defined global symbols, and stores their values in the keyCode property. While the value of each constant depends on the receiver implementation, it must satisfy all the conditions described below:

1. The constant shall be a non-negative integer.

[2] The specification defined in this section is based on W3C Draft Recommendation "Document Object Model (DOM) Level 3 Events Specification" (http://www.w3.org/TR/2012/WD-DOM-Level-3-Events-20120614/). As was specified in 3.1.14.4, the means by which the application reads a keycode shall be the keyCode property, which is specified in the above draft recommendation to ensure backwards compatibility, and the value read from this property is a constant symbol not specified in the above draft recommendation. This reflects consideration about the implementation of the application engine assumed at the time of the development of these Specifications. When Section 5.2.5 of the above-mentioned draft recommendation becomes a recommendation, and it becomes common in the future that application engines are implemented in accordance with that recommendation, this Specification is expected to shift to the specification in the above recommendation. However, since the specification defined in this section does not conflict with the specification in the draft recommendation, application engines can be implemented to satisfy both specifications. While details shall be specified as operational rules, it is desirable that application engines are implemented in this way during the transitional period.

2. No symbol's value shall be identical to the value of any other symbol. An exception is that VK_0 alone can have the same value as the value of either VK_10 or VK_11.

(Note) This description is intended to list symbols. As such, it does not present correct IDL codes.

```
 [NoInterfaceObject]
interface KeyCodeGlobalSymbols {
  const unsigned short VK_RED;
  const unsigned short VK_GREEN;
  const unsigned short VK_YELLOW;
  const unsigned short VK_BLUE;
  const unsigned short VK_UP;
  const unsigned short VK_DOWN;
  const unsigned short VK_LEFT;
  const unsigned short VK_RIGHT;
  const unsigned short VK_ENTER;
  const unsigned short VK_BACK;
  const unsigned short VK_0;
  const unsigned short VK_1;
  const unsigned short VK_2;
  const unsigned short VK_3;
  const unsigned short VK_4;
  const unsigned short VK_5;
  const unsigned short VK_6;
  const unsigned short VK_7;
  const unsigned short VK_8;
  const unsigned short VK_9;
  const unsigned short VK_10;
  const unsigned short VK_11;
  const unsigned short VK_12;
  const unsigned short VK_DBUTTON;
  const unsigned short VK_PLAY_PAUSE;
  const unsigned short VK_PLAY;
  const unsigned short VK_PAUSE;
  const unsigned short VK_STOP;
  const unsigned short VK_FAST_FWD;
  const unsigned short VK_REWIND;
  const unsigned short VK_VCR_OTHER;
  const unsigned short VK_PAGE_UP;
  const unsigned short VK_PAGE_DOWN;
  const unsigned short VK_TAB;};

Window implements KeyCodeGlobalSymbols;
```

3.1.14.5. Key group

This section specifies the interface through which the application controls the scope of the keys it wants to use. It is recommended that the application receive only the necessary and sufficient scope of key events through this interface.

(Note) This description is intended to list symbols. As such, it does not present correct IDL codes.

```
interface KeySet {
        const unsigned short RED;
        const unsigned short GREEN;
        const unsigned short YELLOW;
        const unsigned short BLUE;
        const unsigned short NAVIGATION;
        const unsigned short NUMERIC;
        const unsigned short VCR;
        const unsigned short DBUTTON;
};
```

Key group	Keys handled by the group
RED	VK_RED
GREEN	VK_GREEN
YELLOW	VK_YELLOW
BLUE	VK_BLUE
NAVIGATION	VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_ENTER, VK_BACK, VK_PAGE_UP, VK_PAGE_DOWN, VK_TAB
NUMERIC	VK_0, VK_1, VK_2, VK_3, VK_4, VK_5, VK_6, VK_7, VK_8, VK_9, VK_10, VK_11, VK_12
VCR	VK_PLAY, VK_PAUSE, VK_PLAY_PAUSE, VK_STOP, VK_FAST_FWD, VK_REWIND, VK_VCR_OTHER
DBUTTON	VK_DBUTTON

```
partial interface Application {
        readonly attribute KeySet keySet;
};

partial interface KeySet {
        readonly attribute unsigned long value;
        unsigned long setValue(unsigned long value);
};
```

value	
Description	Value identifying the current key group setting. This is represented as a logical sum of key group symbols.

setValue		
Description	Sets the scope of keys to be sent to the application	
Arguments	value	Value identifying the scope of keys to be sent to the application. This is represented as a logical sum of key group symbols.
Return value	Value identifying the new key group setting that has been set as the result of the processing	

3.1.15. Interface for operating the application using a pointing device

To be specified later.

3.1.16. Interface for operating the application using other devices

To be specified later.

3.1.17. Interfaces for synchronized control of playback

3.1.17.1. Interface for obtaining the broadcast system clock

This section specifies the interface used to obtain the reference broadcast clock, which is used to enable an application to draw images in synchronization with the broadcast audio/video signal.

Interface definition

```
partial interface ReceiverDevice {
        unsigned long long getSTC();
};
```

getSTC	
Description	Obtains the STC time of the service being selected in the receiver
Return value	STC time

3.1.17.2. Interface for control of synchronized playback of the broadcast content and a communication stream

This section specifies the interface used to control the playback of a communication stream in synchronization with the broadcast content.

```
partial interface BroadcastVideoObjectElement {
    void addSlave(HTMLElement element, optional unsigned long long offset);
    void removeSlave(HTMLElement element);
    long getSyncStatus(HTMLElement element);
    void addSyncStateChangeListener(
        HTMLElement element,
        SyncStateChangeListener listener);
    void removeSyncStateChangeListener(
        HTMLElement element,
        SyncStateChangeListener listener);
};

callback SyncStateChangeListener = void (HTMLElement element, long status);
```

addSlave		
Description	Instructs the receiver to use the reference clock of the broadcast service that is referred to by the given broadcast audio/video object in playing the media stream specified in the argment *element*.	
Arguments	element	Specifies the element that is to be played in synchronization with the broadcast service
	offset	Offset value that is added to the reference clock of the broadcast service when a stream is to be played in synchronization with the broadcast service. What to do when this offset value is omitted and how to add the offset value shall be specified in operational rules

removeSlave		
Description	Removes the requirement for synchronization in playing the media stream specified in the argument *element*	
Arguments	element	Element that is no longer required to be played in synchronization

getSyncStatus		
Description	Obtains the state of synchronization with the broadcast services in playing the element specified in the argument *element*	
Arguments	element	Element whose state of synchronization is to be obtained
Return value	Constant value indicating the synchronization state	

addSyncStateChangeListener		
Description	Registers an event listener that checks for a change in the synchronization state	
Arguments	element	Element whose state of synchronization is to be monitored
	listener	Function that is to be called when the synchronization state of the specified element has changed

removeSyncStateChangeListener

Description	Removes an event listener that has been registered using addSyncStateChangeListener	
Arguments	element	Element that has been monitored by the event listener to be removed
	listener	Event listener to be removed. If this argument is omitted, all event listeners that are registered for monitoring the given element are removed

callback SyncStateChangeListener		
Arguments	element	Element of the media stream whose synchronization state has changed
	status	Constant value indicating the state after the change. The same value as the return value for the getSyncStatus function is returned

3.1.18. Interface for caption control

This is specified in 3.2.2.

3.1.19. Obtaining information about the receiver implementation

3.1.19.1. Obtaining information about the product

The getSystemInformation method of the ReceiverDevice object is specified as an interface through which the application obtains information about the receiver product. While it is assumed that information that can be obtained with this method includes the receiver manufacturer and the software version of the application engine, the scope of this information should be specified as operational rules.

3.1.19.2. Interface definition

```
partial interface ReceiverDevice {
        object getSystemInformation(sequence<DOMString>? query);
};
```

3.1.19.3. Method

getSystemInformation		
Description	Obtains information about the application engine or the receiver	
Arguments	query	Character string array identifying the name of the item to be queried. The item name that can be specified shall be defined as operational rules. If this is omitted, it is assumed that the default item specified as operational rules is to be obtained.
Return value	Object that stores information about the item requested in the argument. The property name of this object is the item name, and the property value of the object is the value of that item. If no information can be returned about any specified item, an object without property shall be returned.	

3.1.19.4. Obtaining information about function

The hasCapability method of the Capabilities object specified in 3.1.5 shall be used.

3.1.19.5. Obtaining information about performance of the application engine

To be specified later.

3.1.20. Obtaining information about the receiver location

3.1.20.1. Obtaining information about the receiver installation location

The method described in 3.1.13.2.2 shall be used.

3.1.20.2. Obtaining information about the detailed location of the receiver

To be specified later.

3.1.21. Obtaining the state of parental control setting

To be specified later.

3.1.22. Obtaining information about pay services

To be specified later.

3.1.23. Interface for coordinated operation of the receiver with a terminal

```
typedef sequence<object> DeviceArray;

partial interface ReceiverDevice {
        void setURLForCompanionDevice(DOMString url, CompanionAppOpts options);
        void clearURLForCompanionDevice();
        void getCompanionDeviceList(CompanionDeviceListCallback resultCallback);
        void sendTextToCompanionevice(DOMString? text, optional object devid);
        void addCompanionDeviceTextMessageListener(
                CompanionDeviceTextMessageListener listener);
        void removeCompanionDeviceTextMessageListener(
                optional CompanionDeviceTextMessageListener? listener);
};

callback CompanionDeviceListCallback = void (DeviceArray devlist);
callback CompanionDeviceTextMessageListener =
         void (DOMString? text, object devid);

dictionary CompanionAppOpts {
        boolean auto_start;
        DOMString? app_title;
        DOMString? app_desc;
};
```

setURLForCompanionDevice			
Description	Sets the URL at which the companion application starts its processing, and related information in the host device. The host device must hold the set information while the application that has executed this method is still alive, and send it to the companion application that has requested it. Whether this information is sent to the companion application or not while this method is being executed shall depend on the implementation of the application. When the application that has executed this method is terminated, the given informaiton shall be cleared immediately.		
Arguments	url	URL at which the companion application starts its processing	
	options		
		auto_start	Flag indicating that the URL specified as the starting point is read automatically
		app_title	Character string identifying the title of the associated application.The companion application uses it to present the title to the user.
		app_desc	Character string identifying the description of the associated application. The companion application uses it to present the description to the user.

clearURLForCompanionDevice	
Description	Clears the information that is set in setURLForCompanionDevice

getCompanionDeviceList		
Description	Obtains the list of companion devices connected to the host device	
Arguments	resultCallback	Function to be called when the processing completes

callback CompanionDeviceListCallback		
Arguments	devlist	Device ID array obtained as a result. The type and value of device ID shall depend on the implementation of the device concerned. It shall be ensured that the companion device can be identified uniquely and that whether or not this device is identical with the companion device is identified can be determined through comparison. It is recommended that the host device seeks to maintain the identity of the companion device as far as possible even if the comapnion device is repeatedly disconnected and connected and that the same device ID is asigned to the same companion device.

sendTextToCompanionDevice	
Description	Sends text to the companion application. This Specification does not specify any means by

	which the companion device and companion application receive the string sent by this method. However, a reference example is provided in Appendix A.	
Arguments	text	Character string to be sent to the companion application
	devid	Identifier of the companion device to which the text is sent. If this is omitted, it is assumed that text will be sent to all companion devices connected.

addCompanionDeviceTextMessageListener		
Description	Registers the function to be executed when a string is received from the companion application	
Arguments	listener	Function to be called when a string is received from the companion application

removeCompanionDeviceTextMessageListener		
Description	Removes registration of the function registered by addCompanionDeviceTextMessageListener	
Arguments	listener	Function whose registration is to be removed. if this is omitted, the registration of all registered functions is removed.

callback CompanionDeviceTextMessageListener		
Description	Function to be executed when a string is received from the companion application. This Specification does not specify any means by which the companion application sends a string. However, a reference example is provided in Appendix A.	
Arguments	text	Character string sent from the companion device
	devid	Identifier of the companion device that has sent the string

3.1.24. Interface to the non-volatile memory area

3.1.24.1. Memory area on a local device

To be specified later.

3.1.24.2. Memory area on a remote device

To be specified later.

3.1.24.3. Memory area on the server side

To be specified later.

3.1.25. Reservation for recording and viewing

To be specified later.

3.1.26. Playing of a recorded video

3.1.26.1. Interface for playing a recorded video

3.1.26.1.1. RecordedContent reference object

The RecordedContent reference object inherits the ISDB resource reference object specified in Section 3.1.1. This is used as an argument for specifying a recorded video or used as a return value in the interface for playing a recorded video.

```
dictionary RecordedContentInformation:  ISDBResourceReference{
        attribute Date start_time;
        attribute long long duration;
        attribute boolean isRecorded;
        attribute boolean isAvailable;
        attribute boolean isComplete;
};
```

The meaning of each property is as follows.

original_network_id	Refer to 3.1.1
transport_stream_id	
service_id	
content_id	
event_id	
component_tag	
channel_id	
module_id	
module_name	
resource_name	
start_time	Recording start time of the recorded content to be played
duration	Play duration time (in milliseconds)
isRecorded	Indicates whether the specified content has been recorded. The value is "true" if the content has been recorded.
isAvailable	Indicates whether the specified content has been recorded and is available for playing. The value is "true" if the content is available for playing.
isComplete	Indicates whether the specified content has been completely recorded. The value is "true" if the content has been completely recorded.

Details of isRecorded, isAvailable, and isComplete shall be specified in operational rules.

3.1.26.1.2. Acquisition of information about the availability of the recording/playing function

The hasCapability method of the Capabilities object specified in 3.1.5 is used.

3.1.26.1.3. Acquisition of information about the type of content associated with the application

The sourceIs method of the stream event target object specified in 3.1.12 is used.

3.1.26.1.4. Starting of playing of recorded content

```
partial interface ReceiverDevice {
	void startPVRPlayback(
			ISDBResourceReference content_ref,
			Date start_time,
			long long duration);
};
```

startPVRPlayback		
Description	Starts to play a recorded video that is associated with the argument *content_ref* and includes the time specified in *start_time* and the duration specified in *duration*. When this function has been executed, the application that initiated this function is terminated, the specified recorded content is played and an application associated with the content is launched.	
Arguments	content_ref	Object that indicates the service for the recorded content to be played
	start_time	Recording start time of the recorded content to be played
	duration	Play duration time of the recorded content to be played (in milliseconds)

Handlign of each propoerty in the argument *content_ref* in the startPVRPlayback method.

original_network_id	The application must specify all these.
transport_stream_id	
service_id	
content_id	I*
event_id	I
component_tag	I
channel_id	I
module_id	I
module_name	I
resource_name	I

*I: The application engine ignores this property even if this property is present. It is recommended that the application omits this property.

The recorded content that has been found to be playable from a combination of the recording start time, the recording duration and the broadcast channel specified in the function for obtaining information about recorded content specified in Section 3.1.26.1.5 is specified. Starting the playing of the specified content is equivalent to “selecting the channel of” the given recorded content. (For example, the application that has been launched as a result of the start of playing of the content refers to the broadcast video using the broadcast audio/video object specified in Section 3.2.2.)

3.1.26.1.5. Acquisition of information about recorded content

```
partial interface ReceiverDevice {
        void getRecordedContentInformation(
                        RecordedContentInformation rec_content_info,
                        RecordedContentInformationCallback resultCallback);
};
callback RecordedContentInformationCallback
                                                = void (RecordedContentInformation info);
```

getRecordedContentInformation		
Description	Returns information about recorded content associated with the value of the argument *content_ref*. Since it can be assumed that the identifier returned by this function is associated with the personal information of the receiver user, the developer of an application that uses this function shall pay due consideration to the need to handle the identifier returned by this function in an appropriate manner.	
Arguments	rec_content_info	Object indicating the recorded content to be obtained
	resultCallback	Function to be called when the processing is completed.

callback RecordedContentInformationCallback		
Arguments	info	Object indicating information about the specified recorded content

Handling of each property of the argument *rec_content_info* of the getRecordedContentInformation method

original_network_id	M*
transport_stream_id	M
service_id	M
content_id	I**
event_id	I
component_tag	I
channel_id	I
module_id	I
module_name	I
resource_name	I
start_time	M
duration	M
isRecorded	I
isAvailable	I
isComplete	I

*M: The application engine must always set this property.

**I: The application engine ignores this property even if this property is present. It is recommended that the application omits this property.

Handling of each property of the argument *info* of RecordedContentInformationCallback

original_network_id	M
transport_stream_id	M
service_id	M
content_id	—
event_id	—
component_tag	—
channel_id	—
module_id	—
module_name	—
resource_name	—
start_time	M
duration	M
isRecorded	M
isAvailable	M
isComplete	M

M: The application engine must always set this property.

—: It is recommended that the application engine does not allow this property to be present.

3.1.27. DLNA function control

To be specified later.

3.1.28. About the definition of an interface to functions unique to a receiver

The application engine may have an interface to unique functions not specified in this Specification as a property of the ReceiverDevice object. In this case, at least one of the first four characters of the property name shall be “_”. This Specification will not specify any property with such a name in the future.

It should be noted in developing an application that the ReceiverDevice object may have properties not defined in this Specification.

3.2. Broadcast audio/video object

3.2.1. Application of an object element for broadcast audio/video

This section specifies an object element used to present a broadcast audio/video.

In this Specification, a broadcast audio/video object is defined by inheriting the object element specified in W3C Recommendation HTML5. For details of the object element, refer to W3C Recommendation HTML5 - Section 4.7.4 “The object element”

(http://www.w3.org/TR/html5/embedded-content-0.html#the-object-element).

When an object element with its type attribute is specified as “video/x-iptvf-broadcast,” the default stream of the current channel’s broadcast service shall be presented.

Table 0-3 Attribute of the broadcast audio/video object

Attribute	Value
type	video/x-iptvf-broadcast

With the broadcast audio/video object, initial parameters to be used when the object is generated can be handed over using param elements. Even if the attributes and param element of the object is overwritten by a DOM operation using JavaScript, they shall not be reflected on the receiver’s processing part. The relevant broadcast audio/video is controlled using functions provided by the object.

3.2.2. Broadcast video/object definition

The broadcast audio/video object is defined as follows.

Broadcast audio/video object definition

```
interface BroadcastVideoObjectElement :   HTMLObjectElement {
  boolean enableFullscreen();
  boolean disableFullscreen();
  boolean isFullscreen();
  boolean enableAudioMute();
  boolean disableAudioMute();
  boolean isAudioMute();
  boolean setAudioSrc(DOMString url);
  DOMString getAudioSrc();
  Boolean setVideoSrc(DOMString url);
  DOMString getVideoSrc();
  boolean setCaptionSrc(DOMString url);
```

```
   DOMString? getCaptionComponentURL();
   boolean isCaptionExistent(DOMString url);
   boolean setCaptionVisibility(optional boolean flag);
   boolean isCaptionVisible();
   void addCaptionListener(CaptionListener listener, optional DOMString url);
   void removeCaptionListener(optional CaptionListener listener);
      callback CaptionListener = void (DOMString captiondata)
};
```

When the enableFullscreen() function is called, the receiver can display the broadcast video at the forefront in full screen. If the full screen display is successful, “true” shall be returned. If not, “false” shall be returned. If enableFullscreen() is called while the display is already full screen, full screen display shall be maintained, and true shall be returned. If disableFullscreen() is called, full screen display shall be terminated. If the termination is successful, “true” shall be returned. If not, “false” shall be returned. If disableFullscreen() is called while the display is not full screen, that state shall be maintained, and “true” shall be returned. When Fullscreen() is called, “true” shall be returned if the broadcast video display is full screen, and “false” shall be returned if not.

Code examples are shown below.

Code example of the enableFullscreen function

```
var video = document.getElementById('video');

if ('function' === typeof(video.enableFullscreen)) {
   video.enableFullScreen();
} else {
   /* Width, height, and z-index are overwritten to bring the element to the forefront and display it in full screen */
}
```

Code example of the disableFullscreen function

```
var video = document.getElementById('video');

if ('function' === typeof(video.disableFullscreen)) {
   video.disableFullScreen();
} else {
   /* Width, height, and z-index are overwritten to bring the element back to its original size */
}
```

Code example of the isFullscreen function

```
var video = document.getElementById('video');

if ('function' === typeof(video.isFullscreen)) {
   if (video.isFullScreen()) {
          /* In the case of full screen display */
   } else {
          /* In the case of normal display */
   }
}
```

When the enableAudioMute() function is called, the receiver shall make a transition to the mute state. If muting is successful, “true” shall be returned. If not, “false” shall be returned. If enableAudioMute() is called while the receiver is in the mute state, that state shall be maintained and “true” shall be returned. When disableAudioMute() is called, muting shall be disabled. If this disabling is successful, “true” shall be returned. If not, “false” shall be returned. If disableAudioMute() is called while the receiver is in the mute-disabled state, that state shall be maintained and “true” shall be returned. If isAudioMute() is called while the receiver is in the mute state, “true” shall be returned. If not, “false” shall be returned.

Code examples are shown below.

Code example of the enableAudioMute function

```
var video = document.getElementById('video');

if ('function' === typeof(video.enableAudioMute)) {
   video.enableAudioMute();
} else {
   /* When muting is not posssible */
}
```

Code example of the disableAudioMute function

```
var video = document.getElementById('video');

if ('function' === typeof(video.disableAudioMute)) {
   video.disableAudioMute();
} else {
   /* When muting diabling is not possible */
}
```

Code example of the isAudioMute function

```
var video = document.getElementById('video');

if ('function' === typeof(video.isAudioMute)) {
   if (video.isAudioMute()) {
          /* When in the muting state */
   } else {
          /* When audio output is possible */
   }
}
```

The setAudioSrc() and setVideoSrc() functions specify an audio stream and a video stream, respectively, transmitted in MPEG2-TS. The URL specified in these functions is identified uniquely by the following name. Refer to ARIB STD-B24 9.2.3 “References of AV stream and caption component.”

arib://<original_network_id>.<transport_stream_id>.<service_id>[;<content_id>]
[.<event_id>]/<component_tag>[;<channel_id>]

If the stream is specified successfully, “true” shall be returned. If not, “false” shall be returned. Both the getAudioSrc() and getVideoSrc() functions shall return the URL of the stream selected by the receiver.

Code examples are shown below.

Code example of setAudioSrc function

```
var video = document.getElementById('video');

if ('function' === typeof(video.setAudioSrc)) {
   /* When specifying the second audio channel in a dual monoral audio stream */
   video.setAudioSrc(“arib://-1.-1.-1/-1;2”);
} else {
   /* When src specification of audio is not possible */
}
```

Code example of setVideoSrc function

```
var video = document.getElementById('video');

if ('function' === typeof(video.setVideoSrc)) {
   video.setVideoSrc(“arib://-1.-1.-1/-1”);
} else {
   /* When src specification of video is not possible */
}
```

Code example of getAudioSrc function

```
var video = document.getElementById('video');

if ('function' === typeof(video.getAudioSrc)) {
   var src = video.getAudioSrc();
   /* processing pursuant t src informaiton */
} else {
   /* When acquisition of audio src is not possible */
}
```

Code example of getVideoSrc function

```
var video = document.getElementById('video');

if ('function' === typeof(video.getVideoSrc)) {
   var src = video.getVideoSrc();
   /* processing pursuant to src informaiton */
} else {
   /* When acquisition of video src is not possible */
}
```

The setCaptionSrc() function specifies the caption component transmitted in MPEG2-TS. The value “true” is returned if the caption has been specified successfully, and “false” is returned if not.

The getCaptionComponentURL() function returns the character string of information about the caption component that the receiver is instructed to present. Note that the language identification displayed through user operations may differ from the initial value set by a param element or the value specified using the setCaptionSrc() function. The value “null” is returned if there is no caption that the receiver is instructed to present.

The isCaptionExistent() function specifies a caption component and determines whether the given caption component is currently played by the receiver. The value “true” is returned if the given caption component is present, and “false” is returned if not.

The setCaptionVisibility() function specifies whether the receiver should present captions. If its argument is omitted or is “true,” the receiver presents the currently specified caption component. If the argument is “false,” it is deemed that the broadcast caption is not to be displayed.

When the isCaptionVisible() function is called, the value “true” is returned if captions are being displayed, and “false” is returned if not.

The addCaptionListener() function adds the event listener that obtains the broadcast caption text specified by the url, and calls a callback function CaptionListener. The url indicates the caption stream that is to be obtained. If this url is omitted, it is deemed that the value set in a param element or the caption stream specified using the setCaptionSrc() function is specified. CaptionListener is a function that is called when a caption that satisfies a given condition has been received. The given caption text is sent in units of PES. In CaptionListener, captiondata shall be text data within the readable range extracted from the given caption. Details of the text data shall be specified in operational rules. The removeCaptionListener() function removes caption listeners. The argument *listener* can be omitted. If it is omitted, all event listeners that are associated with the currently specified caption component are removed.

Caption components specified in the respective functions are uniquely identified using the following names. No abbreviations are used. For details of each parameter, refer to ARIB STD-B24 Volume 2 Section 9.2.3 “Reference to AV streams and caption components.”

arib://<original_network_id>.<transport_stream_id>.<service_id>[;<content_id>]
[.<event_id>]/<component_tag>[;<channel_id>]

The following specification is added to specify caption components.

In cases where a component tag is specified to be -1, it is deemed that the currently selected caption component is specified. channel_id shall not be specified. A language identification that uniquely identifies the language to be used is attached to the end of the url as [;<language_tag>]. The value of language identification shall be the same as those specified in language_tags in caption management data described in ARIB STD-B24 Volume 1 Section 9.3.1.

Code examples are shown below.

Code example of setCaptionSrc function

```
var video = document.getElementById('video');

if ('function' === typeof(video.setCaptionSrc)) {
    /* Case where the first language is specified in the caption of the broadcast service */
   video.setCaptionSrc(“arib://-1.-1.-1/-1;0”);
} else {
   /* Case where the caption cannot be specfied using an src */
}
```

Code example of getCaptionComponentURL function

```
var video = document.getElementById('video');

if ('function' === typeof(video.getCaptionComponentUrl)) {
   var url = video.getCaptionComponentUrl();
   /* Processing appropriate for the caption component URL information */
} else {
   /* Case where acquisition of caption text cannot be used */
}
```

Code example of isCaptionExistent function

```
var video = document.getElementById('video');

if ('function' === typeof(video.isCaptionExistent)) {
   if(video.isCaptionExistent(arib://-1.-1.-1/-1;0)){
   /* Case where the first language is broadcast in the caption of the broadcast service */
    } else {
   /* The first language is not broadcast */
}
```

Code example of setCaptionVisibility function

```
var video = document.getElementById('video');

if ('function' === typeof(video.setCaptionVisibility)) {
  /* Instructs presentation of the caption in the broadcast service */
   video.setCaptionVisibility();
} else {
   /* Case where setCaptionVisibility cannot be used */
}
```

Code example of isCaptionVisible function

```
var video = document.getElementById('video');

if ('function' === typeof(video.isCaptionVisible)) {
   if (video.isCaptionVisible()) {
         /* State of caption being displayed */
   } else {
         /* State of caption not being displayed */
   }
}
```

Code example of addCaptionListener function

```
var video = document.getElementById('video');

if ('function' === typeof(video.addCaptionListener)) {
   /* Case where a callback function processes the obtained text in the caption of the broadcast service */
   video.addCaptionListener(function(captiondata) , “arib://-1.-1.-1/-1;0”);
} else {
   /* Case where the addtion of caption listener cannot be used */
}
```

Code example of removeCaptionListener function

```
var video = document.getElementById('video');

if ('function' === typeof(video.removeCaptionListener)) {
   /* Removes all listeners related to the caption of the broadcast service */
   video.removeCaptionListener();
} else {
   /* Case where the removal of caption listeners in the caption text cannot be used */
}
```

A param element shall be used when specifying the operation state from the initial operation of the broadcast audio/video object. Table 3-4 shows the parameter names and values that can be specified.

Table 3-4 Broadcast audio/video object parameter list

Name	Value
fullscreen	enable: Launch in full screen mode disable: Terminate full screen mode (default)
video_src	URL indicating the video stream
audio_src	URL indicating the audio stream
audio_mute	enable: Mute mode disable: Terminate mute mode (default)
caption_src	“; (semicolon)” is attached to the URL character string that expressed the caption stream attached. It is followed by the language identification.

Parameter code example

```
<object id=”video” type="video/x-iptvf-broadcast">
    <param name=”video_src” value=”arib://-1.-1.-1/-1”>
    <param name=”audio_src” value=”arib://-1.-1.-1/-1;2”>
    <param name=”audio_mute” value=”disable”>
    <param name=”fullscreen” value=”enable”>
    <param name=”caption_src” value=”arib://-1.-1.-1/-1;0”>
</object>
```

3.2.2.1. Playing of recorded content

This section adds an API related to playing recorded content to the broadcast audio-video object definition specified in Section 3.2.2.

```
partial interface BroadcastVideoObjectElement   {
  void addPVRPlaybackStateListener(PVRPlaybackStateListener stateListener);
  void removePVRPlaybackStateListener(
                          PVRPlaybackStateListener stateListener);
     callback PVRPlaybackStateListener       = void (unsigned long state);
  Date? getPVRPlaybackPoint();
  void resumePVRPlayback();
  void pausePVRPlayback();
  void stopPVRPlayback();
  void fastForwardPVRPlayback();
  void backwardPVRPlayback();
  void jumpPVRPlayback(attribute Date position);
};
```

The addPVRPlaybackStateListener function registers a listener that is executed when the playback state has changed, and calls a callback function PVRPlaybackStateListener. The removePVRPlaybackStateListener function removes a listener that is executed when the playback state has changed.

The callback function sends the state of the recorded content that has just begun to be played. It sends “0” if the content has come to EOF, “1” if the playing speed has changed from a non-regular speed to the regular speed, “2” if the speed has changed from the regular speed to a non-regular speed, and “3” if the user has stopped the playing of the content by pressing the stop key. Applications detect the playback state from a callback event. If an application detects a shift from a non-regular speed to the regular speed, it can obtain the playback position by executing getPVRPlaybackPoint().

When a getPVRPlaybackPoint function is called, the receiver returns the recording time of the current playback position of the recorded content being played.

If the receiver fails to get this information, it returns “null.”

When a pausePVRPlayback function is called, the receiver temporarily pauses the given recorded content.

When a resumePVRPlayback function is called, the receiver resumes a regular-speed playback of the given recorded content.

When a stopPVRPlayback function is called, the receiver stops playing the given recorded content.

When a backwardPVRPlayback function is called, the receiver fast rewinds the given recorded content while playing it. When a fastForwardPVRPlayback function is called, the receiver fast forwards the recorded content while playing it. When a jumpPVRPlayback function is called, the receiver jumps to that recording time of the currently played recorded content that is specified in its argument, and plays the content from there.

3.2.3. Operation at the time of a transition of an HTML application

When an HTML application makes a transition into another application in accordance with the application control information, the video currently presented by the original HTML application before the transition shall be maintained until the newly loaded HTML application determines whether to present a broadcast video. If the new HTML application is to present the same broadcast video, that video shall continue to be presented without interruption.

An example of this operation is shown in Fig. 3-2　Example of continued display of a broadcast video at the time of a transition of an HTML application. In accordance with the application control information, the receiver instructs the browser to load an HTML application (http://example.com/01.html). The browser displays this application. It is assumed that the application includes a broadcast video object and that the broadcast video is to be displayed. As the program proceeds, the control information instructs the browser to load the next application (http://example.com/02.html). Until the layout of the next application has been determined and it has been determined whether the same broadcast video object should be used or not, the display of the current broadcast video object shall continue to be maintained in the layout of the original application without any interruption to the broadcast video.

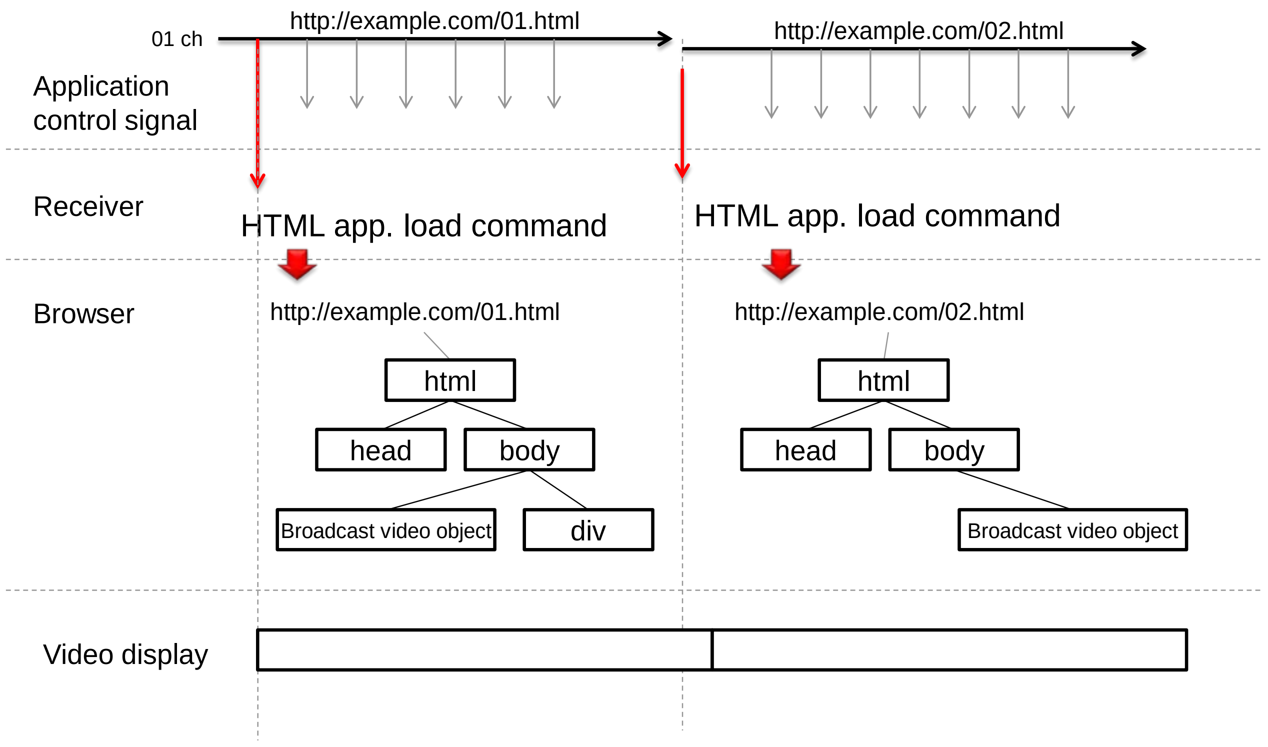


Fig. 3-2 Example of continued display of a broadcast video at the time of a transition of an HTML application

3.2.4. How to present a broadcast video in a broadcast video object area

The receiver shall display a broadcast video in the layout (position and size) specified by the broadcast video object. The scaling process that should be performed by the receiver is described below.

3.2.4.1. When the aspect ratios of the broadcast video and the broadcast video object are the same

When the aspect ratio (ratio of the width to the height) of the broadcast video is the same as the one specified in the broadcast video object, the receiver shall scale the broadcast video to the size of the broadcast video object and display it. The relation in size between the broadcast video and the broadcast video object is shown in Fig. 3-3 Scaling operation (when the two aspect ratios are the same)(for a case where the aspect ratios of the two are the same). Let the width and the height of the broadcast video be Wv and Hv, respectively, and let those of the broadcast object be We and He, respectively. The scaling ratio, i.e., the ratio of scaling the broadcast video, can be expressed by the following formula:

Scaling ratio = We / Wv = He / Hv

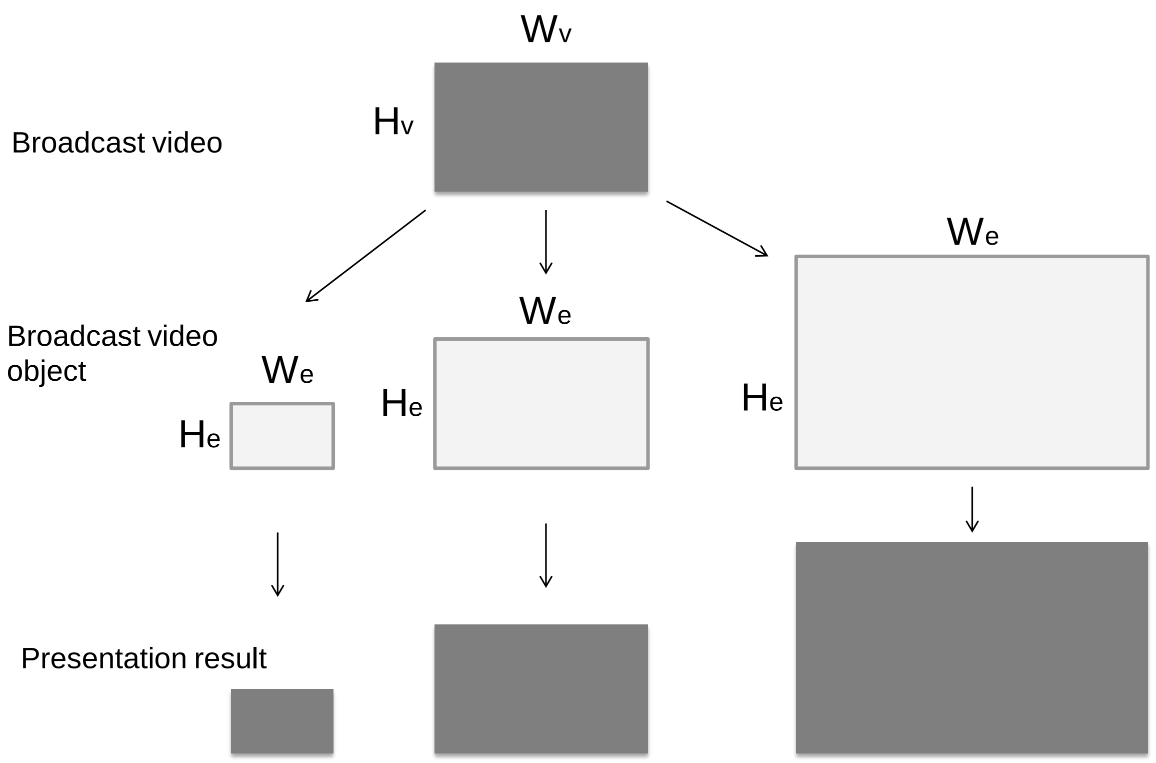


Fig. 3-3 Scaling operation (when the two aspect ratios are the same)

3.2.4.2. When the two aspect ratios are different

When the aspect ratio of the broadcast video is different from that specified in the broadcast video object, the receiver shall maintain the aspect ratio of the broadcast video and display it in such a way that no part of the video image is lost. The scaling ratio shall be determined in accordance with the following algorithm:

- When Wv / Hv < We / He
  Scaling ratio = He / Hv

- When Wv / Hv > We / He
  Scaling ratio = We / Wv

Further study is required to determine what to do with the gaps between the scaled broadcast video and the broadcast video object.

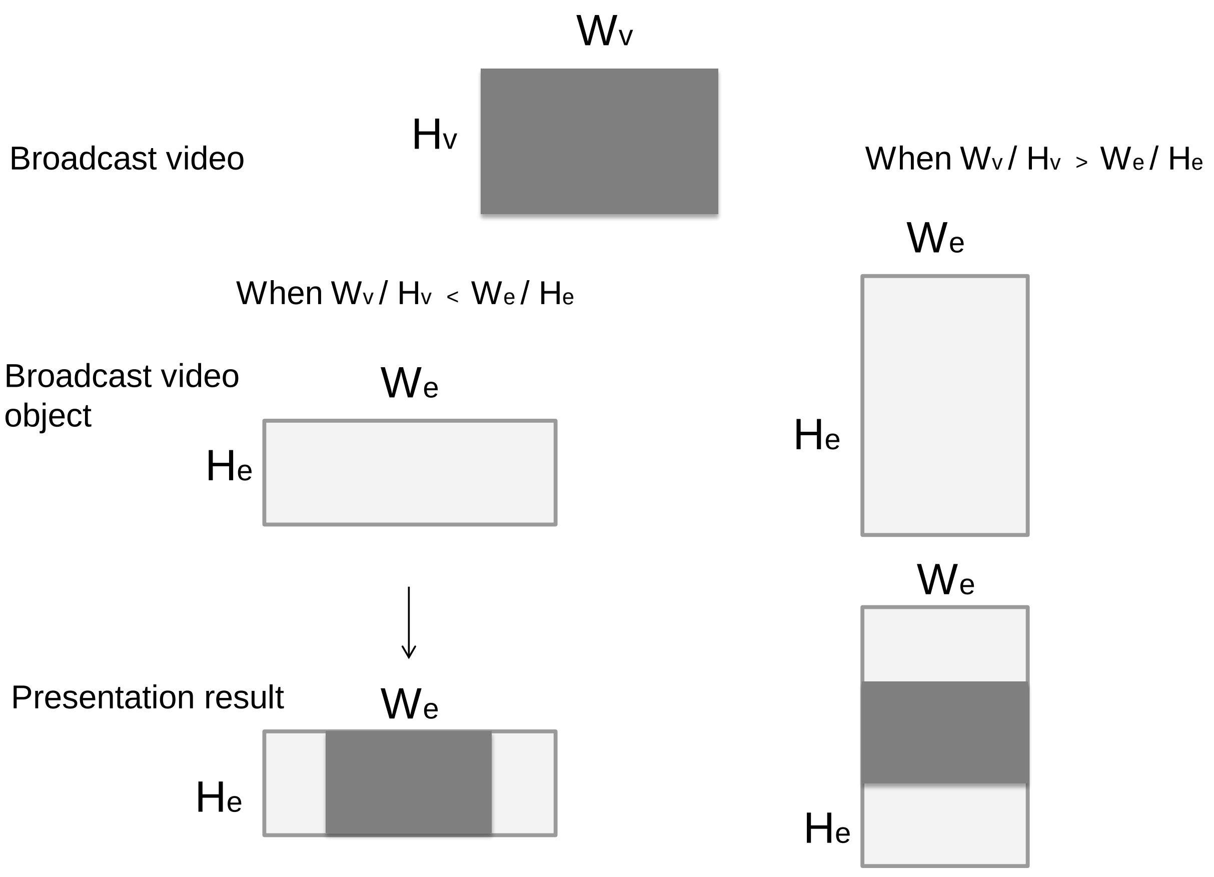


Fig. 3-4 Scaling operation (for a case where the two aspect ratios are different)

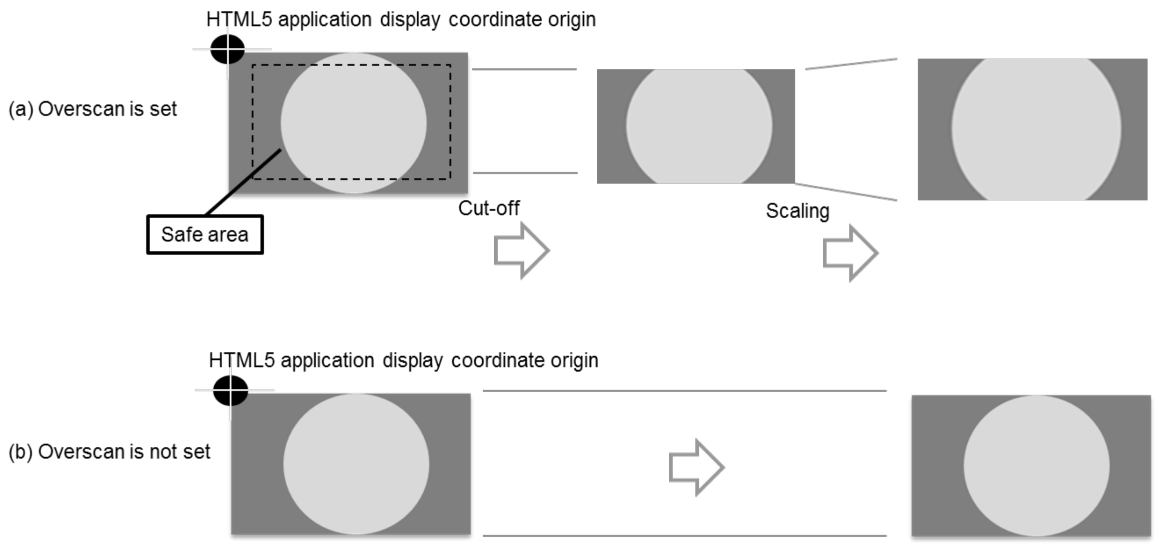
3.2.4.3. Handling of overscan

If the broadcast video is displayed in full screen, i.e., if the following requirements are satisfied, the receiver may overscan. If not, the receiver may not overscan.

1. The upper left position of the broadcast video object is at (0, 0) of the coordinates of the display area of the HTML application (browser).
2. The width and height of the broadcast video object are the same as those of the display area of the HTML application.

When the receiver overscans, it cuts off a part of the video irrespective of whether the video is a broadcast video or another presentation element, and scales the video so that it will be displayed in full screen. An example of this operation is shown in Fig. 3-5(a). When receiver does not overscan, no scaling operation is carried out and the video is displayed as it is, as shown in Fig. 3-5(b). If the position and size of the broadcast video object do not satisfy the above requirements, no overscan operation shall be allowed regardless of the setting of the receiver. In this case, the video is displayed as shown in Fig. 3-6.

With a receiver for which overscan is set, whether overscan occurs or not depends on whether the position and size of the broadcast video object satisfies the above requirements for full screen display. Therefore, it should be noted that a difference of position or size by just a single pixel may cause a significant difference in the display of a broadcast video.

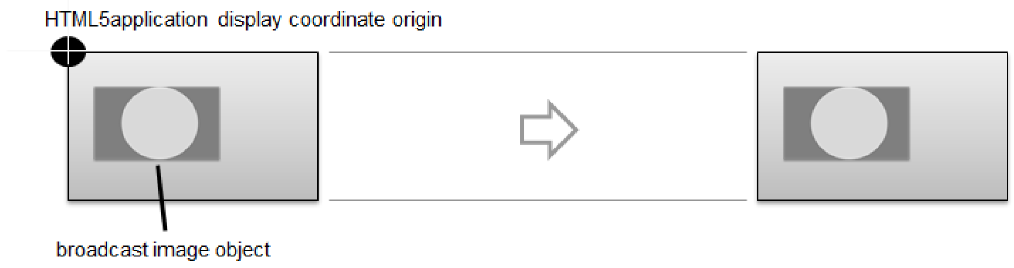

HTML5 application display coordinate origin
(a) Overscan is set
Safe area
Cut-off
Scaling
HTML5 application display coordinate origin
(b) Overscan is not set

Origin of the display coordinates of the HTML5 application

(a) Overscan is set

(b) Overscan is not set

Fig. 3-5 Example of operation when the requirements for full screen display are satisfied



Origin of the display coordinates of the HTML5 application

Broadcast video object

Fig. 3-6 Example of operation when the requirements for full screen display are not satisfied

3.2.4.4. When the height and width of the broadcast video object are omitted

When both the width and height attributes, which specify the width and height of the broadcast video object, or the specification of the height and width in a style sheet are omitted, the video object shall be displayed with the same width and height as those of the broadcast video. It is recommended to specify the width and height using attributes of the broadcast video object or a style sheet in order to keep the video layout consistent irrespective of the environment in which the video is displayed.

3.3. Recorded video playback object

3.3.1. Application of an object element to the playback of a recorded video

This section specifies an object element that is used to play a recorded video.

In this Specification, a recorded video playback object is defined by inheriting an object element specified in W3C HTML5 recommendation. For details of the object element, refer to W3C Recommendation HTML5 Section 4.8.4 “The object element” (http://www.w3.org/TR/html5/the-object-element.html#the-object-element).

Recorded content can be presented by specifying “video/x-iptvf-playbackvideo” in the type attribute of this object element.

Table 3-5 Attribute of the recorded video playback object

Attribute	Value
type	video/x-iptvf-playbackvideo

Initial parameters used at the time when the recorded video playback object was generated can be handed over to the object using respective param elements. Playback can be controlled using a function it provides.

The method of presenting a broadcast video object described in Section 3.2.4 is applied to the presentation of this object.

3.3.2. Definition of the recorded video playback object

The definition of the recorded video playback object is described below.

Definition of the recorded video playback object

```
interface PlaybackVideoObjectElement : HTMLObjectElement {
  boolean startPVRPlayback(
    ISDBResourceReference content_ref,
    Date start_time,
    long long duration,
  );
  void addPVRPlaybackStateListener(PVRPlaybackStateListener stateListener);
  void removePVRPlaybackStateListener(
                          PVRPlaybackStateListener stateListener);
    callback PVRPlaybackStateListener         = void (unsigned long state);
  Date? getPVRPlaybackPoint();
  void resumePVRPlayback();
  void pausePVRPlayback();
  void stopPVRPlayback();
  void fastForwardPVRPlayback();
  void backwardPVRPlayback();
  void jumpPVRPlayback(attribute Date position);
};
```

startPVRPlayback plays recorded content that contains the program specified by content_ref for a duration specified in *duration* from the time specified in *start_time*. The value “true” is returned if the playback is successful, and “false” is returned if not. The recorded content specified here shall be one that has been found to be playable from a combination of recording start time, recording duration and the program editing service specified in the recorded content information acquisition function described in Section 3.1.26.1.5. Even if an application executes this function within this object, it is not terminated but continues its execution.

For details of functions other than startPVRPlayback, refer to Sections 3.2.2 and 3.2.2.1. Details on the state of the object after completion of playback shall be specified in operational rules.

[Appendix A] API of the terminal for second screen services (informative)

The interface specification for coordinated operation with a terminal specified in Section 3.1.23 of this Specification defines the APIs (e.g. a means of sending or receiving text data) of the receiver, with a view to realizing a coordinated terminal control model, which is specified in IPTVFJ STD-0010 11.7.1, and the coordinated terminal operation service scenario, which is shown as an example in Appendix C.5 of the above specification.

The model mentioned above indicates that similar APIs are also needed at the associated terminal. Such APIs are out of the scope of this Specification and not defined in this Specification. However, for developers of applications for such second screen services, it is useful if there are APIs that are as common as possible to all applications, irrespective of the implementations of companion applications[3].

This Appendix shows an example of implementing such common APIs.

A.1 Assumptions

This Appendix is based on the following assumptions:

- The method of communication between the receiver and a companion application depends on the receiver. The companion application shall be implemented to suit the receiver.
  (Therefore, it is generally assumed that the implementer of a companion application is a receiver manufacturer.)
- The requirements for the platform on which a companion application runs are left to the implementer of the companion application. However, it is recommended that a companion application that is implemented for a certain platform can operate, as far as possible, irrespective of the manufacturer, model, etc. of the terminal that runs this platform.
- The companion application is implemented using a framework that is normally provided by the platform on which the application runs, and it can be implemented without adding any extended specification to the framework.

A.2 Example of realizing a common API on the companion application side

Based on the above assumptions, this section shows an example of a means of enabling HTML documents on the companion application side to use a common API.

For the sake of simplicity, the following description assumes an API in JavaScript within an HTML document, but this does not mean that common APIs are limited to such cases.

[3] IPTVF STD-0010 does not specify the method of communication between the receiver and an associated terminal. It only shows an example in which it is assumed that companion applications are provided for each receiver manufacturer and for each receiver model (refer to Appendix C.5 in IPTVF STD-0010).

A.2.1 Concept for providing a common API

The means of providing a common API shown as an example in this Appendix is intended to enable the implementer of a companion application to also implement the communication processing that is performed by a means dependent on the companion application. It is also intended to enable the developer of the HTML document on the companion application side to call the API.

This will be realized using the following three elements.

- Common API implementation code

  This is a JavaScript code describing the implementation of a common API. An HTML document on the companion application side can use a common API by loading this JavaScript code as part of its own script. Since a communication method dependent on each companion application is needed to implement the API, it is assumed that the implementer of the companion application provides the API code. Two main ways of allocating the code can be conceived. The implementer of the companion application (as mentioned in A.1, this implementer is generally assumed to be the receiver manufacturer) selects the way of allocation. Whichever way is selected, the location of the code must be indicated in the value of a location indication query parameter (shown below). The two ways are:

  1. Allocate the code on the server managed by the implementer of the companion application
  2. Embed the code in the companion application.

- Location indication query parameter

  This query parameter specifies the URL of the location where the common API code exists. It is assumed that, when setURLForCompanionDevice is executed at the receiver side, the receiver or a companion application adds this query parameter to the URL specified by the argument *url*. That is, either of the following is assumed:

  1. The query parameter is added within the implementation of setURLForCompanionDevice in the receiver platform or the receiver's application engine.
  2. The companion application adds the query parameter when it is informed of the URL by the receiver.

  Requests including this query parameter are sent to the server on which the HTML document of the companion application side is located (generally, a server managed by the creator of terminal-coordinated applications, such as a broadcaster). The name of the location indication query parameter is not restricted to any particular one but it needs to be agreed to by the receiver and companion application as well as the above server so that the server can handle this query appropriately, such as ignoring it.

- API implementation reading script

This is a script that extracts the URL string specified in the query parameter value from the URL of the own document, and adds the resource (i.e., the common API implementation code) indicated by this URL to the script of the own document. It is assumed that the creator of the HTML document on the companion application side creates this query and includes it within the document, and that this query is executed before the common API is used. As long as the document includes this function, details of its implementation and timing of its execution are left to the creator of the HTML document on the companion application side.

A.2.2 Processing flow

This section illustrates how the processing flows up to the point where the common API becomes usable to the HTML document on the companion application side. The common API implementation code, the location indication query parameter, and the common API implementation reading script described in the previous section are respectively represented as maker.js, host-params and tvc-init() for the purpose of giving an example, but the names are not restricted to these. Refer to Fig. A-1.

1. (① in Fig. A-1) The application on the receiver side executes setURLForCompanionDevice. The figure shows a case where 'http://broadcaster.jp/ca/top.html' is specified in the argument *url*.

2. (② in Fig. A-1) The implementation of setURLForCompanionDevice within the receiver platform adds a location indication query parameter to the URL specified in 1, and notifies the companion application of it. The figure shows a case where the name of the query parameter is host-params and the common API implementation code is located at 'http://maker.jp/maker.js'.
   The URL that the companion application is notified of is
   http://broadcaster.jp/ca/top.html?host-params="http://maker.jp/maker.js"

3. (③ in Fig. A-2) The companion application receives the URL notified in 2, and sends a request to that URL.

4. (④ in Fig. A-2) A request is sent to the broadcaster server http://broadcaster.jp by 3 above. This server ignores a query parameter called host-params, and returns a response assuming that the request is meant to be sent to http://broadcaster.jp/ca/top.html.
5. (⑤ in Fig. A-3) The companion application loads the top.html of the HTML document. The creator of the top.html includes the definition and execution instruction of the API implementation loading script in the top.html. The figure shows a case where tvc-init() is executed using the *onload* attribute of the body element. An example of the script of tvc-init() is as follows.

```
function tvc-init() {
var queryies, knv, elem;

if (location.search.length > 1) {
   queries = location.search.substring(1).split('&');
   for (i in queries) {
      knv = queries[i].split('=');
      if (knv[0] == "host-params") {
         elem = document.createElement("script");
         elem.src = knv[1];
         document.head.appendChild(elem);
      }
   }
}
}
```

6. (⑥ in Fig. A-4) As a result of 5, maker.js, which is the common API implementation code, is added to the script of top.html. Thereafter, the common API is usable to scripts within top.html.

7. When there is a transition from top.html to another HTML document, a conceivable method of enabling the latter document to also use the common API is to add the host-params query parameter included in the URL of the own document to the destination URL when specifying the destination URL in top.html. This enables the destination document to use the common API by executing the above procedure from 3.

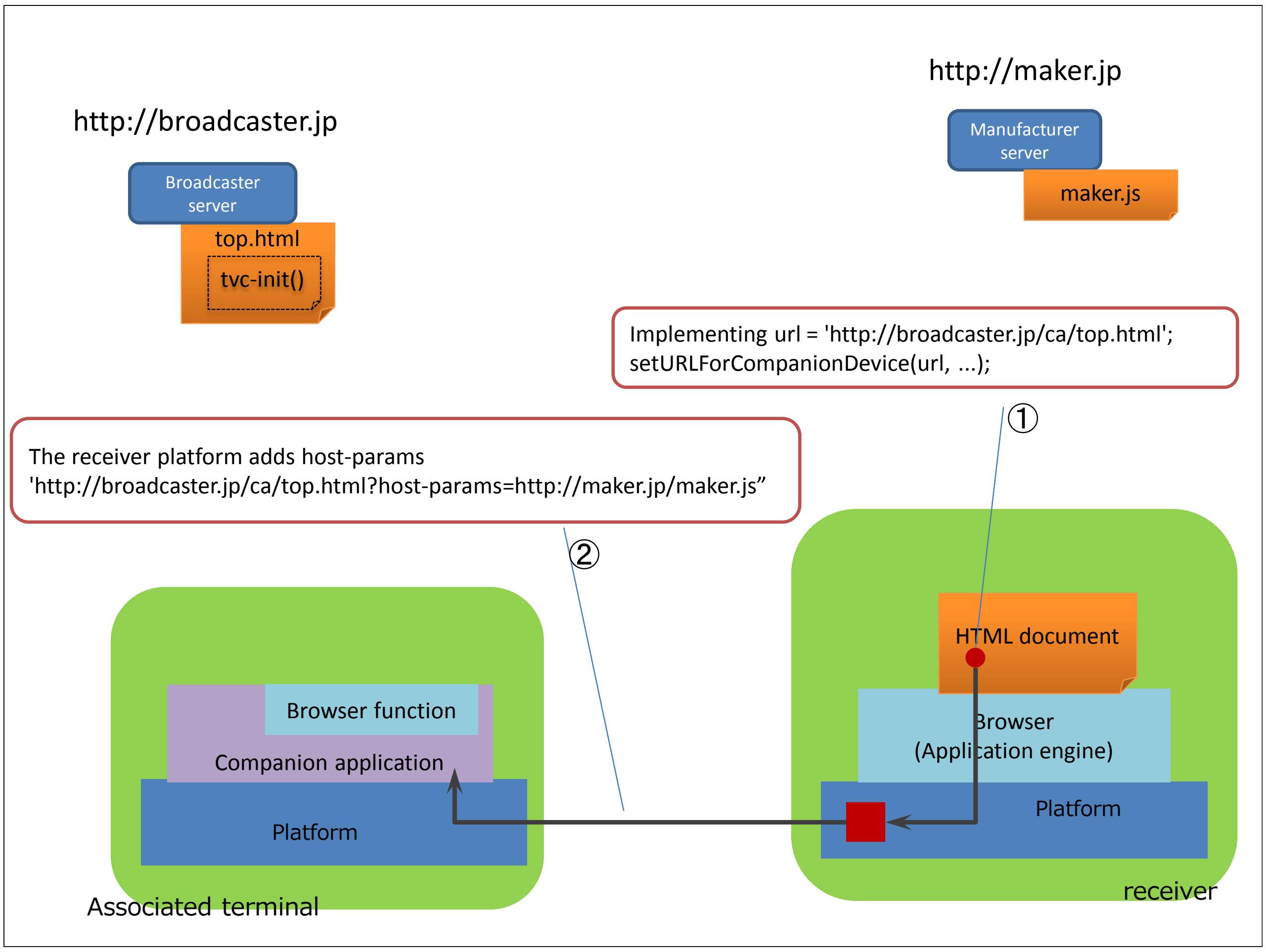


Fig. A-1 Processing flow of the common API ①-②

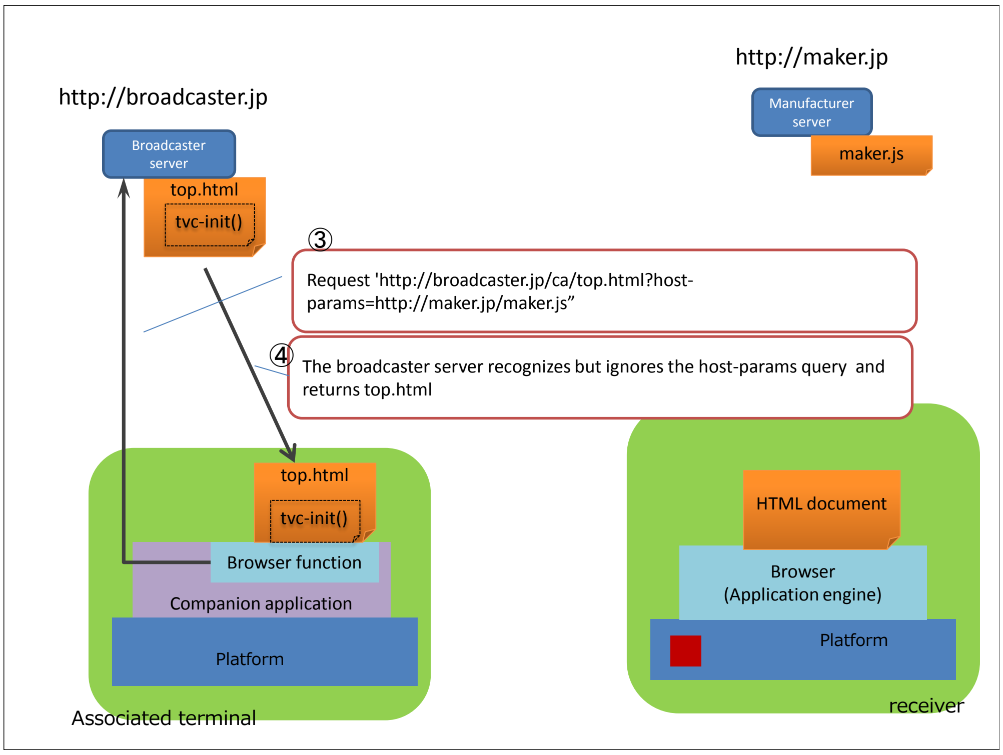


Fig. A-2 Processing flow of the common API ③-④

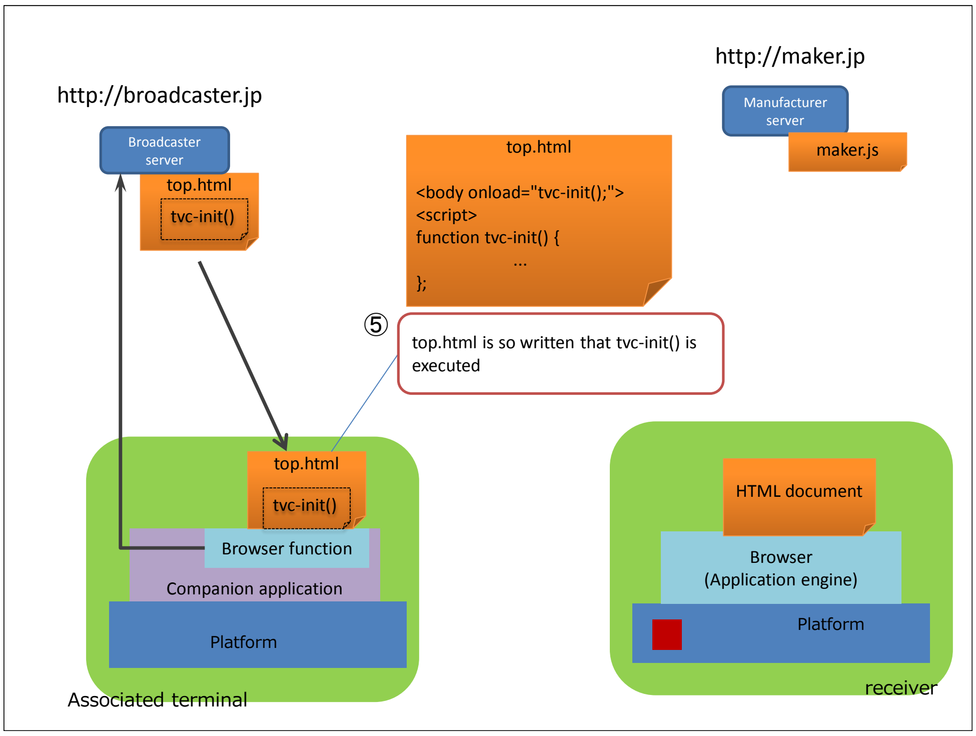


Fig. A-3 Processing flow of the common API ⑤

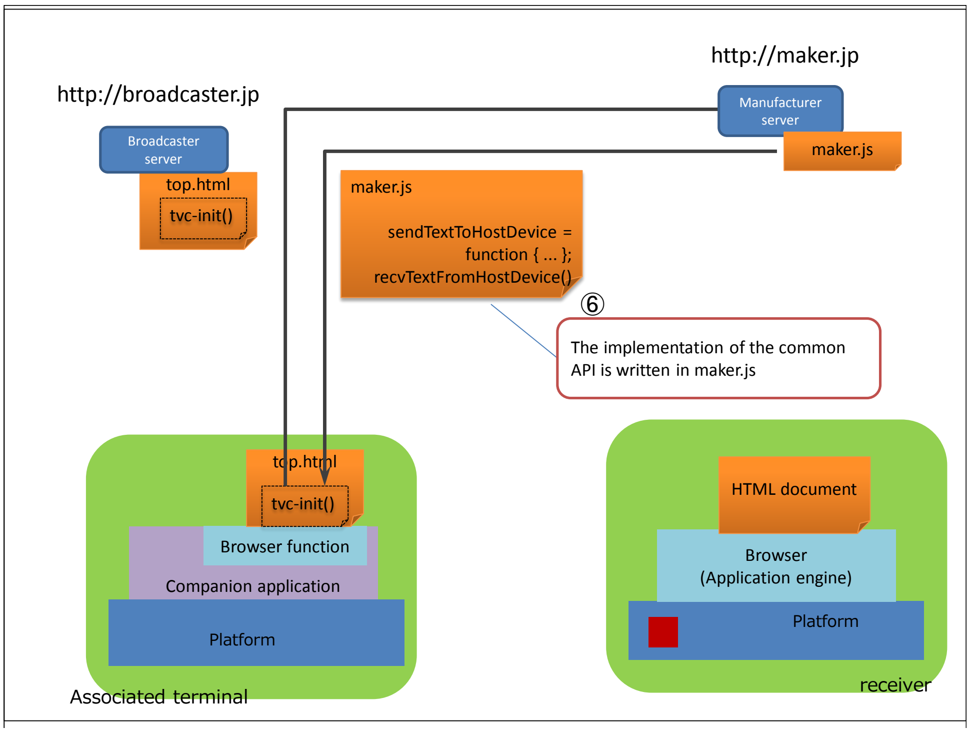


Fig. A-4 Processing flow of the common API ⑥

History of revision to Version 2.1

Page	Section	Version 2.1	Version 2	Reason for revision
32	3.1.3	by the application method getOwnerApplication of the application manager object, or by the method getApplications of the ApplicationInformationTable object	by the method getOwnerApplication of the application manager object	A new method is introduced
32	3.1.3	New properties, type, organization_id, appliation_id, control_code, and autostart_priority, are added to the application interface		New property is introduced
32	3.1.3	An ApplicationBoundaryAndPermissionDescriptor interface is added		A new interface is introduced as a result of modifications to the Specification
35	3.1.4.1	The AITApplicationBoundaryAndPermissionDescriptor method is removed from the ApplicationInformationTable interface		A method is removed as a result of modifications to the Specification
35	3.1.4.1	A getApplications method is added to the ApplicationInformationTable interface		A new method is introduced
35	3.1.4.1	The AITApplicationBoundaryAndPermissionDescriptor interface is removed		An interface is removed as a result of modifications to the Specification
57	3.1.12.4	A new interface is defined to receive AIT update notices (a new section is created)		A new interface is introduced

74	3.2.2	void addCaptionListener(CaptionListener listener, optional DOMString url);	void addCaptionListener(CaptionListener listener);	An argument is introduced
77	3.2.2	The addCaptionListener() function adds the event listener that obtains the broadcast caption text specified by the url, and calls the callback function CaptionListener. The url indicates the caption stream that is to be obtained. If this url is omitted, it is deemed that the value set in a param element or the caption stream specified using the setCaptionSrc() function is specified.	The addCaptionListener() function adds the event listener that obtains the broadcast caption text, and calls the callback function CaptionListener.	An argument is introduced
78	3.2.2	In cases where a component tag is specified to be -1, it is deemed that the currently selected caption component is specified. channel_id shall not be specified.	In cases where a component tag is specified to be -1, it is deemed that 0x30 is specified. channel_id shall not be specified.	Modification to the Specification
78	3.2.2	url	URL	A varied notation is removed
79	3.2.2	addCaptionListener(function(captiondata) , “arib://-1.-1.-1/-1;0”);	addCaptionListener(function(captiondata));	Modification as a result of the addition of an argument

IPTV Standard
HTML5 Browser Specifications
IPTVFJ STD-0011 Version 2.1

Created on March 22, 2013: Version 1.0
Revised on June 29, 2014: Version 2.0
Revised on December 15, 2014: Version 2.1

IPTV Forum Japan
Futaba Akasaka Bld. 3F, 8-5-43,
Akasaka, Minato-ku, Tokyo, 107-0052
Tel: 03-5858-6685
FAX: 03-5858-6675
e-mail: sec@iptvforum.jp