Performance Requirements for Web-based Signage Terminals

Version 1.0

March 2016

Authored by: Web platform performance benchmark Study Group

Issued by: Advanced Web Architecture Lab, Keio Research Institute at SFC

Contents

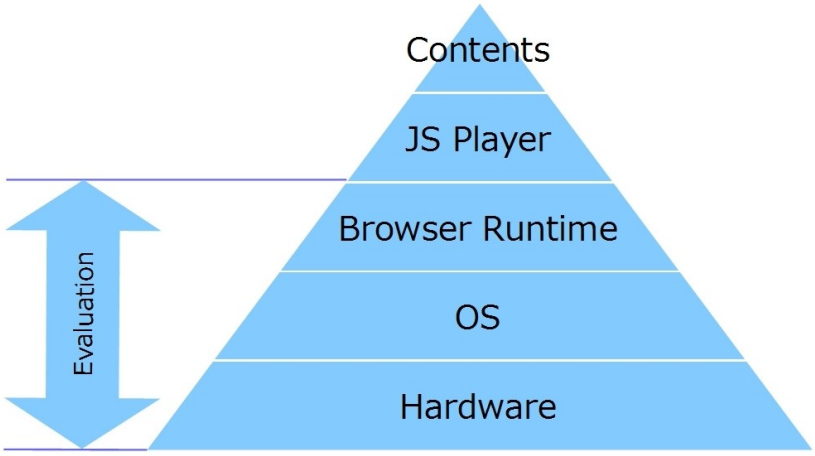
1 Introduction

This document describes requirements for Web-based Signage (WBS) terminals, especially on the Web / HTML5 technologies, for WBS to be widely used. Chapter 3 through 10 describes each requirement. The test procedure for each requirement is described in "Web-based Signage terminal performance test specification."

1.1 Range of performance evaluation

Web-based Signage consists of Content Management System (CMS; Manages content registration and playlists), Content Delivery Network (CDN; delivery cloud and network), and terminals to play contents. The subject of evaluation in this document is terminals.

A terminal consists of the following elements.



Hardware

It refers to components that physically make up the terminal. Specifically, they are the panel, SoC (CPU, GPU, memory, etc.), and input and output interface (HDMI, USB, power supply, etc.).

Operating system

It refers to the basic software, which is installed in order to use the terminal hardware as Web-based Signage. More specifically, it refers to Windows, Android, iOS, Linux, FreeBSD, Firefox OS, and so on.

Browser runtime

HTML, CSS, It refers to an application platform that can operate the standard Web technologies such as HTML, CSS, and JavaScript. More specifically it refers to the web browser, but it need not necessarily be a general web browser. It includes run-time environment such as Android WebView.

JS Player

It refers to an application for playing a content based on the playlist. In Web-based Signage, it is made as a web application that runs on the browser runtime. Specifically, HTML, CSS, and JavsScript are used for the development.

Content

The content is displayed or run on the JS player. It is what a user in front of the signage terminal can see. Content may include an image, video or a web application. It may include audio as well.

The subject of the performance evaluation of this document, the terminal, means the hardware, operating system, and the browser runtime among the above-mentioned components.

Moreover, instead of evaluating these components individually, the comprehensive evaluation as an actual terminal is assumed.

Of the signage terminals using web / HTML5 technologies, those without network connection or standalone-type terminals that do not use network when playing contents (such as contents in USB storage), are excluded from the evaluation.

The current performance requirements in this document are focused mainly on those for displaying using web / HTML5 technologies. General requirements for signage terminals such as long-term stability (that it does not hang up) and task scheduler-related requirements are not included. Also not included are requirements relating to external devices, in view of the implementation status of device API's in current browsers.

1.2 Evaluation policy

Usage of Web-based Signage is wide-ranging. The terminal does not necessarily need to meet all performance requirements in this document. Therefore, a simple "good or bad" evaluation is not assumed. Instead, several classes are defined for each requirement item to accommodate various combination of requirements.

2 Classification of requirements and performance evaluation

Here are the seven requirement items we define with two or three performance classes for each item. Thus, a web-based signage terminal can be classified according to its performance.

Classification table

Requirement item	classification	Description
Resolution	Small-density type	Conventional signage terminals that are assumed to be seen from a distance. Not suitable to display small characters.
	Medium-density type	Small characters (even smaller than movie captions) can be read clearly when seen from close range. In addition, if the size of the panel is small, it can be used for branding content.
	High-density type	This class of resolution allows you to view content with an emphasis on branding and reality.
Storage	Cache type	Refers to the terminal that does not have the ability to save the image and video files for content.
	Medium-capacity type	Refers to a terminal having a function of storing an image file or video file for content, with storing size not being large.
	Large-capacity type	Refers to a terminal having a function of storing an image file or video file for content, with storing size being large.
Rendering speed	Low-speed type	Rendering contents such as tickers, animation, or full-screen image transition (moving or fading) leads to frequent frame dropping that anyone can notice. Low-speed type suffices where only still images are to be shown.
	Medium-speed type	Rendering contents such as tickers, animation, or full-screen image transition (moving or fading) leads to some but not annoying frame dropping.
	High-speed type	Rendering contents such as tickers, animation, or full-screen image transition (moving or fading) does not lead to any noticeable frame dropping.
Parallel processing	Single-processing type	It refers to a terminal where parallel processing is not effective. This type suffices where only still images are to be shown.
	Multi-processing type	It refers to a terminal where simultaneous parallel processing is supported, and the parallel processing does not interfere with the normal playback of video and animation.
Playing video	Low-functioning type	Terminals of this type can play the video with resolution smaller than that supported by the browser runtime.

	Medium-functioning type	Terminals of this type can play the video in full-screen with the maximum resolution supported by the browser runtime.
	High-functioning type	In addition to the ability to play the video in full-screen with the maximum resolution supported by the browser runtime, terminals of this type can play video while scaling (reducing and magnifying), rotating, or moving.
Playing audio	Low-functioning type	This type of terminals can simply play audio.
	High-functioning type	This type of terminals support audio generation, synthesis, and audio effect as well as simply playing audio.
Playback start delay	Low-speed type	With this type of terminals, there is a certain amount of delay between the play command and the actual start of playing.
	High-speed type	With this type of terminals, there is only little delay between the play command and the actual start of playing.

3 Resolution

The following table shows the classification and evaluation criteria of the resolution requirements.

Performance Requirements

Classification	Evaluation criteria
Small-density type	It refers to a terminal where viewport resolution of the browser runtime is 720P (1280 x 720 pixels) equivalent or less in terms of the number of pixels.
Medium-density type	It refers to a terminal where viewport resolution of the browser runtime is larger than 720P (1280 x 720 pixels) and smaller than or equal to 1080P (1920 x 1080 pixels) in terms of the number of pixels.
High-density type	It refers to a terminal where viewport resolution of the browser runtime is larger than 1080P (1920 x 1080 pixels) in terms of the number of pixels.

Viewport resolution of the browser run-time depends on the size of the video memory to be allocated to the browser. Further, the higher the resolution, if the browser runtime does not support GPU acceleration, it is impossible to improve the performance as expected.

This item is classified by the size of the browser run-time viewport resolution. Every terminal belongs to one of the three classes.

3.1 Small-density type

It refers to conventional signage terminals that are assumed to be seen from a distance. It is not suitable to display small characters. More specifically, it refers to a terminal where viewport resolution of the browser runtime is 720P (1280 x 720 pixels) equivalent or less in terms of the number of pixels. This HD equivalent resolution is very popular for general use of signage, and is sufficient for most of the lightweight use cases.

3.2 Medium-density type

It refers to a terminal where small characters (even smaller than movie captions) can be read clearly when seen from close range. In addition, if the size of the panel is small, it can be used for branding content. More specifically, it refers to a terminal where viewport resolution of the browser runtime is larger than 720P

(1280 x 720 pixels) and smaller than or equal to 1080P (1920 x 1080 pixels) in terms of the number of pixels.

Medium-density type, compared with the small-density type, enables detailed pixel representation. Especially, even small characters can be read clearly from a short distance. It can be utilized for a message board such as floor guidance. In addition, a terminal with as large as about 42-inch display can be used to show branding contents.

It is probable that medium-density type becomes mandatory for Web-based Signage in the near future.

3.3 High-density type

This class of resolution allows you to view content with an emphasis on branding and reality. More specifically, it refers to a terminal where viewport resolution of the browser runtime is larger than 1080P (1920 x 1080 pixels) in terms of the number of pixels.

High-density type, even with a large panel, the pixels are very dense. In addition, with a small panel, it can provide contents with the very reality. Its quality will be equivalent to that of conventional high-end digital signage systems.

4 Storage

The following table shows the classification and evaluation criteria of the storage requirements.

Performance Requirements

Classification	Evaluation criteria
Cache type	It refers to a terminal that does not have the ability to save image or video content files, or a terminal that can save such files but does not have W3C standard "Indexed Database API"[*1] nor "File API: Directories and System"[*2] implemented in the browser runtime.
Medium-capacity type	It refers to a terminal that has the ability to save image and video content files, and has W3C standard "Indexed Database API"[*1] or "File API: Directories and System"[*2] implemented in the browser runtime, with the storage size less than 1024MB.
Large-capacity type	It refers to a terminal that has the ability to save image and video content files, and has W3C standard "Indexed Database API"[*1] or "File API: Directories and System"[*2] implemented in the browser runtime, with the storage size equal to or more than 1024MB.

Whether or not a terminal supports storage affects the performance such as network traffic, the time from the terminal start-up to the content playback, and stable long-term operation.

This performance item is classified by the presence or absence of storage, and the size of the storage. Every terminal belongs to one of the three classes.

Note that playing contents that are stored beforehand can also be realized by methods outside the scope of HTML5 (including JavaScript). As the target of this requirement document is HTML5-based methods, this performance item also conforms to it.

4.1 Cache type

This type refers to a terminal that does not have storage, or a terminal that has storage but does not have W3C standard "Indexed Database API"[*1] nor "File API: Directories and System"[*2] implemented in the browser runtime.

As a cache type terminal needs downloading contents every time it starts up, it takes time to start playing contents. Although browser cache (note that it is not application cache) may reduce the downloaded data at the next start-up time, the classification here does not consider whether browser cache exists.

Also, bad network conditions may make video playback unstable. On the other hand, omitting storage can reduce the cost of terminal hardware. In addition, if the network environment is good between the content delivery server and the terminal (such as on-premise delivery environment), network does not become a big issue.

4.2 Medium-capacity type

This type refers to a terminal that has storage, and has W3C standard "Indexed Database API"[*1] or "File API: Directories and System"[*2] implemented in the browser runtime, with the storage size less than 1024MB.

As a terminal of this type store contents before playing them, stable playback that does not depend on the network environment can be expected. Further, since it is not necessary to download the content each time, as long as the content is not updated, the time from the terminal start-up to content playback is greatly reduced.

4.3 Large-capacity type

This type refers to a terminal that has storage, and has W3C standard "Indexed Database API"*1 or "File API: Directories and System"*2 implemented in the browser runtime, with the storage size equal to or more than 1024MB.

∗1 W3C Indexed Database API

http://www.w3.org/TR/IndexedDB/

The medium- and large- capacity types need ability to save Blob objects and create Blob URL defined in File API specification (createObjectURL() method), in addition to Indexed Database API.

http://www.w3.org/TR/FileAPI/#dfn-createObjectURL

∗2 W3C File API: Directories and System

http://dev.w3.org/2009/dap/file-system/file-dir-sys.html

As of March 2016, W3C File API: Directories and System is obsolete, and File System API has newly been under development.

http://w3c.github.io/filesystem-api/

However, it is not a public draft yet, and there is no implementation in a browser. "File API: Directories and System" will, therefore, be used for the time being. In the future, it will be replaced by the File System API.

5 Rendering speed

The following table shows the classification and evaluation criteria of the rendering speed requirements.

Performance Requirements

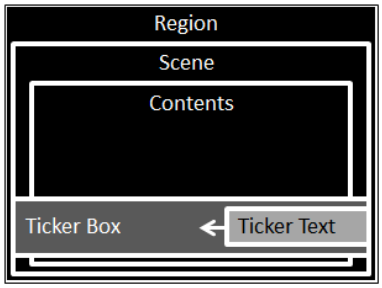
Classification	Evaluation criteria
Low-speed type	It refers to a terminal where dropped frames in 15 seconds is more than a total of 10 frames in either case that ticker text (implemented by fonts instead of an image, with the height of 10% of the screen height) is horizontally moving, or that a full-screen still image is horizontally moving using CSS transform / transition.
Medium-speed type	It refers to a terminal where dropped frames in 15 seconds is more than a total of 3 frames in either case that ticker text (implemented by fonts instead of an image, with the height of 10% of the screen height) is horizontally moving, or that a full-screen still image is horizontally moving using CSS transform / transition, and the dropped frames is less than or equal to a total of 10 frames in both cases.
High-speed type	It refers to a terminal where dropped frames in 15 seconds is less than or equal to a total of 3 frames in both cases that ticker text (implemented by fonts instead of an image, with the height of 10% of the screen height) is horizontally moving, and that a full-screen still image is horizontally moving using CSS transform / transition.

Frame dropping is an important indicator on measuring the performance of content animation. This performance item is classified based on the frame dropping. Every terminal belongs to one of the three classes.

The following comprehensive evaluation should be performed for the classification.

Ticker

Ticker Text moves through the ticker area that has been prepared in the scene area (Ticker Box), at an appropriate speed from right to left.



The display panel is assumed to be in landscape orientation. A full-screen still image is assumed to be displayed in the scene area. With this image being the background, Ticker Text moves at a certain speed from the right end to the left end in the Ticker Box, the size of which is 10% of the screen height. The movement of the ticker is evaluated under this condition. The Ticker Text is assumed to be implemented by fonts instead of an image, and to be sufficiently long.

Transition effect is realized by CSS transform and CSS transition.

Example CSS
Before transition: transform: translateX(1920px); transition: transform 30s linear 0s;
After transition: transform: translateX(-3840px);
* The argument of translateX() and the transform duration should be adjusted to take into account the screen width and the text length.

Screen transition

The performance of transition for scene change is evaluated, where a still image is horizontally moved (sliding). The resolution of the image is assumed to be the same as that of the browser runtime viewport. Although the duration of the transition is 1 second or less in a typical signage application, here for the purpose of performance evaluation, the appropriate number of seconds is assumed.

The screen transition includes fading-in and scaling as well as sliding, which are shown below.

Transition effect is realized by CSS transform and CSS transition.

Fading-in

Example CSS
Before transition: opacity: 0; transition: opacity 5s linear 0s;
After transition: opacity: 1;

Sliding

Example CSS
Before transition: transform: translateX(-1920px); transition: transform 5s linear 0s;
After transition: transform: translateX(0px);
* The argument of translateX() should be set to the screen width.

Scaling

Example CSS
Before transition: transform: scale(0); transition: transform 5s linear 0s;
After transition: transform: scale(1)

5.1 Low-speed type

Anyone can notice frame dropping that frequently occurs in this type of terminal. Ticker text is hard to read.

5.2 Medium-speed type

There is some but not annoying frame dropping in this type of terminal which is no problem in practical use.

5.3 High-speed type

There is no noticeable frame dropping in this type of terminal and the animation can be played exactly as expected.

6 Parallel processing

The following table shows the classification and evaluation criteria of the parallel processing requirements.

Performance Requirements

Classification	Evaluation criteria
Single-processing type	It refers to a terminal where parallel processing is not effective. This type suffices where only still images are to be shown. Specifically, it refers to a terminal where Web Worker is not implemented, or where it is implemented but the processing time does not decrease even if two worker threads are used.
Multi-processing type	It refers to a terminal where simultaneous parallel processing is supported, and the parallel processing does not interfere with the normal playback of video and animation. Specifically, it refers to a terminal where Web Worker is implemented and using two worker threads reduces the processing time compared to the case with a single thread.

A JS player for Web-based signage performs various processes in parallel with displaying contents according to a playlist. In addition, in order to support multi-region processing, a JS player needs to be able to play multiple contents independently in each region.

In this case, by executing non-rendering processes in the background, various parallel processing is possible without interfering with the processes of playing contents.

This performance item is classified by whether shortening of the processing time is observed by using multiple threads. Every terminal belongs to one of the two classes.

6.1 Single-processing type

It refers to a terminal where Web Worker is not implemented, or where it is implemented but the processing time does not decrease even if two worker threads are used. A terminal with a single-core CPU is generally classified here.

6.2 Multi-processing type

It refers to a terminal where Web Worker is implemented and using two worker threads reduces the processing time compared to the case with a single thread. A terminal with a multicore CPU is generally classified here.

7 Playing video

The following table shows the classification and evaluation criteria of the playing video requirements.

Performance Requirements

Classification	Evaluation criteria
Low-functioning type	A terminal of this type should meet the followings. (1) It should support at least either of mp4 (H.264 / AAC / MP4) or webm (VP8 / Vorbis / WebM). (2) The video playback should start when the play() method of a video element in JavaScript is executed. The playable video resolution may be smaller than the maximum resolution of the browser runtime.
Medium-functioning type	In addition to the above (1) and (2), it should play the video with the maximum resolution the browser runtime supports and with the frame rate of 30 fps or more, not dropping a frame.
High-functioning type	In addition to the above (1) and (2), it should play the video with the maximum resolution the browser runtime supports and with the frame rate of 30 fps or more, not dropping a frame, and it should support scaling, rotating, and moving before playing (specifically applying CSS Transforms to a video element should be supported).

Playing video requires a decent CPU power and GPU power. In particular, in order to play the video with the maximum resolution the browser runtime supports, it is necessary for the browser runtime to utilize hardware decoding and GPU acceleration. This influences the performance of playing video very much.

In order to belong to any of the three classes, a terminal needs to meet all of the following requirements.

1. To support at least one of the following video types.

MIME-Type	Video codec	Audio codec	Container
video/mp4	H.264	AAC	MP4
video/webm	VP8	Vorbis	WebM

2. To support automatic playing.

Some recent browsers for smartphones or tablets disable automatic playing in JavaScript without user interaction. In order to belong to any of the video playing classes, a terminal should avoid this limitation.

In practice, the video playback should start when the play() method of a video element in JavaScript is executed.

Even if a terminal can play video, if it does not meet the above requirements, it can not belong to any of the video playing classes.

7.1 Low-functioning type

It mainly refers to a terminal that only supports software decoding. Because decoding and rendering is carried out only by a CPU, smooth full-screen playback can not be expected, but the video with small resolution can be played smoothly.

The video element defined in HTML5 specification is assumed. Creation and playing of the video element should be possible from JavaScript.

7.2 Medium-functioning type

In addition to the requirements for the low-functioning type, a terminal of medium-functioning type should be able to play full-screen video, which is the most common usage as a signage. It should be able to play the full-screen video with the maximum resolution the browser runtime supports, and with the frame rate of 30 fps, not dropping a frame.

7.3 High-functioning type

In addition to the requirements for the medium-functioning type, a terminal of high-functioning type should be able to play the video after scaling (reducing / magnifying), rotating, and moving. In practice, applying CSS Transforms* to a video element should be supported.

* CSS Transform
  http://www.w3.org/TR/css3-transforms/

8 Playing audio

The following table shows the classification and evaluation criteria of the playing audio requirements.

Performance Requirements

Classification	Evaluation criteria
Low-functioning type	A terminal of this type should meet the followings. (1) It should support at least one of the following audio types: mp3 (MP3 / MP3), mp4 (AAC / M4A), webm (Vorbis / WebM), or ogg (Vorbis / Ogg). (2) The audio playback should start when the play() method of an audio element in JavaScript is executed.
High-functioning type	In addition to the above requirements, a terminal of this type should support audio generation, synthesis, and effects using Web Audio API's. Specifically, it should support at least the following interfaces. AudioContext, AudioNode, AudioDestinationNode, AudioBuffer, AudioBufferSourceNode, MediaElementAudioSourceNode, AudioParam, GainNode, DelayNode, OscillatorNode。

In order to belong to any of the two classes, a terminal needs to meet all of the following requirements.

1. To support at least one of the following audio types.

MIME-Type	Audio codec	Container
audio/mp3	MP3	MP3
audio/mp4	AAC	M4A
audio/webm	Vorbis	WebM
audio/ogg	Vorbis	Ogg

2. To support automatic playing.

   Some recent browsers for smartphones or tablets disable automatic playing in JavaScript without user interaction. In order to belong to any of the audio playing classes, a terminal should avoid this limitation.

   In practice, the audio playback should start when the play() method of a audio element in JavaScript is executed.

Even if a terminal can play audio, if it does not meet the above requirements, it can not belong to any of the audio playing classes.

8.1 Low-functioning type

The audio element defined in HTML5 specification is assumed. Creation and playing of the audio element should be possible from JavaScript.

8.2 High-functioning type

In addition to the audio playback using an audio element defined in the HTML5 specification, a terminal of this type should support audio generation, synthesis, and effects using Web Audio API's.

By using Web Audio API's, not only reducing the downloaded data by generating a sound source from a script, various audio representation become possible by synthesizing multiple sound sources.

The Web-based Signage does not require all of the advanced audio synthesizing functions that Web Audio API's provide. Basically, it is sufficient to cover the following use cases.

- ✓ To control the audio element using Web Audio API's
- ✓ To play multiple audio sources at the same time
- ✓ To apply the volume control and the delay to multiple sound sources individually
- ✓ To generate simple sine wave sound from JavaScript

In order to be classified as high-functioning type, a terminal should be able to realize the above use cases. They can be realized if the terminal support at least the following interfaces.

- ✓ AudioContext
- ✓ AudioNode
- ✓ AudioDestinationNode
- ✓ AudioBuffer
- ✓ AudioBufferSourceNode
- ✓ MediaElementAudioSourceNode
- ✓ AudioParam
- ✓ GainNode
- ✓ DelayNode
- ✓ OscillatorNode

9 Playback start delay

The following table shows the classification and evaluation criteria of the playback start delay requirements.

Performance Requirements

Classification	Evaluation criteria
Low-speed type	It refers to a terminal where the delay between the play command execution and the actual start of playing is equal to or more than 0.5 seconds for at least either of video and audio.
High-speed type	It refers to a terminal where the delay between the play command execution and the actual start of playing is less than 0.5 seconds for both of video and audio.

This evaluation is to be carried out in the following procedure. It is described for video, and replacing "video" with "audio" makes the audio description.

1. Create a video element in JavaScript (HTMLVideoElement object).
2. Set the preload property of the HTMLVideoElement object "auto".
3. Set a timeupdate event listener on the HTMLVideoElement object.
4. Set the src property of the HTMLVideoElement object the URL of a video file.
5. Insert the HTMLVideoElement object to the document.
6. After a few seconds, execute the play() method of the HTMLVideoElement object.
7. The time until the first timeupdate event from the play() method execution is the delay to be evaluated.

A typical browser, after the creation of a video or an audio element and setting its src property the URL of the video / audio file, downloads and decodes a certain amount of the video / audio data from the beginning to stand by. Thus the playback can be started anytime, except when "none" is intentionally set for the preload property of the video / audio element.

Some of the recent browsers for smartphones and tablets prohibit preloading. It is difficult for such a browser to be classified as high-speed type.

In many browsers, a timeupdate event occurs every 250 milliseconds after playing video is started. Therefore, if the environment is such that playing can be started as

soon as the play() method is called, a timeupdate event occurs at least once during 500 milliseconds (0.5 seconds).

It is important to ensure a network bandwidth and server response sufficient to preload enough data before the play() method execution.