External Media Tests Documentation

Release 0.1

Syd Polk and Maja Frydrychowicz

April 05, 2016

Contents

External Media Tests is a library built on top of Firefox Puppeter and the Marionette python client. It is designed to test playback of video elements embedded in web pages, independent of vendor. Using this library, you can write tests which play, pause, and stop videos, as well as inspect properties such as currentTime().

CHAPTER 1

Installation

External Media Tests lives in External Media Tests Source. Documentation for installation and usage lives on External Media Tests; this documentation is API documentation for the various pieces of the test library.

CHAPTER 2

Contents

2.1 external_media_harness package

Test case classes for use in video tests

2.1.1 external_media_harness.testcase module

class external_media_harness.testcase.**MediaTestCase**(**args*, ***kwargs*)
Bases: firefox_puppeteer.testcases.base.BaseFirefoxTestCase, marionette.marionette_test.MarionetteTestCase

Necessary methods for MSE playback

Parameters **video_urls** – Urls you are going to play as part of the tests.

check_playback_starts(*video*)
Check to see if a given video will start. Raises if the video does not start.

Parameters **video** – VideoPuppeteer instance to play.

log_video_debug_lines()
Log the debugging information that Firefox provides for video elements.

run_playback(*video*)
Play the video all of the way through, or for the requested duration, whichever comes first. Raises if the video stalls for too long.

Parameters **video** – VideoPuppeteer instance to play.

save_screenshot()
Make a screenshot of the window that is currently playing the video element.

skipTest(*reason*)
Skip this test.

Skip with marionette.marionette_test import SkipTest so that it gets recognized a skip in marionette.marionette_test.CommonTestCase.run

class external_media_harness.testcase.**NetworkBandwidthTestCase**(**args*, ***kwargs*)
Bases: *external_media_harness.testcase.MediaTestCase*

Test MSE playback while network bandwidth is limited. Uses browsermobproxy (https://bmp.lightbody.net/). Please see https://developer.mozilla.org/en-US/docs/Mozilla/QA/external-media-tests for more information on setting up browsermob_proxy.

run_videos()
Run each of the videos in the video list. Raises if something goes wrong in playback.

class external_media_harness.testcase.**VideoPlaybackTestsMixin**
Bases: object

Test MSE playback in HTML5 video element.

These tests should pass on any site where a single video element plays upon loading and is uninterrupted (by ads, for example).

This tests both starting videos and performing partial playback at one minute each, and is the test that should be run frequently in automation.

test_playback_starts()
Test to make sure that playback of the video element starts for each video.

test_video_playback_partial()
Test to make sure that playback of 60 seconds works for each video.

2.2 external_media_tests package

This document highlights the utility classes for tests. In general, the indvidiual tests are not documented here.

2.2.1 Test pacakges

VideoPuppeteer

video_puppeteer.VideoPuppeteer

class external_media_tests.media_utils.video_puppeteer.**VideoPuppeteer**(*marionette*, *url*, *video_selector='video'*, *interval=1*, *set_duration=0*, *stall_wait_time=0*, *timeout=60*)
Bases: object

Wrapper to control and introspect HTML5 video elements.

A note about properties like current_time and duration: These describe whatever stream is playing when the property is called. It is possible that many different streams are dynamically spliced together, so the video stream that is currently playing might be the main video or it might be something else, like an ad, for example.

Parameters

- **marionette** – The marionette instance this runs in.
- **url** – the URL of the page containing the video element.
- **video_selector** – the selector of the element that we want to watch. This is set by default to 'video', which is what most sites use, but others should work.
- **interval** – The polling interval that is used to check progress.

- **set_duration** – When set to >0, the polling and checking will stop at the number of seconds specified. Otherwise, this will stop at the end of the video.
- **stall_wait_time** – The amount of time to wait to see if a stall has cleared. If 0, do not check for stalls.
- **timeout** – The amount of time to wait until the video starts.

corrupted_frames

Returns Number of video frames corrupted since the creation of this video element. A corrupted frame may be created or dropped.

current_time

Returns Current time of whatever stream is playing right now.

dropped_frames

Returns Number of video frames created and dropped since the creation of this video element.

duration

Returns Duration in seconds of whatever stream is playing right now.

execute_video_script(*script*)
Execute JS script in 'content' context with access to video element.

Parameters **script** – script to be executed

Returns value returned by script

get_debug_lines()
Get Firefox internal debugging for the video element.

Returns A text string that has Firefox-internal debugging information.

lag

Returns The difference in time between where the video is currently playing and where it should be playing given the time we started the video.

pause()
Tell the video element to Pause.

play()
Tell the video element to Play.

remaining_time

Returns How much time is remaining given the duration of the video and the duration that has been set.

total_frames

Returns Number of video frames created and dropped since the creation of this video element.

update_expected_duration()
Update the duration of the target video at self.test_url (in seconds).

expected_duration represents the following: how long do we expect playback to last before we consider the video to be 'complete'? If we only want to play the first n seconds of the video, expected_duration is set to n.

video_src

Returns The url of the actual video file, as opposed to the url of the page with the video element.

video_url

Returns The URL of the video that this element is playing.

video_puppeteer.VideoException

exception external_media_tests.media_utils.video_puppeteer.**VideoException**
Exception class to use for video-specific error processing.

video_puppeteer.playback_started

external_media_tests.media_utils.video_puppeteer.**playback_started**(*video*)
Determine if video has started

Parameters **video** – The VideoPuppeteer instance that we are interested in.

Returns True if is playing; False otherwise

video_puppeteer.playback_done

external_media_tests.media_utils.video_puppeteer.**playback_done**(*video*)
If we are near the end and there is still a video element, then we are essentially done. If this happens to be last time we are polled before the video ends, we won't get another chance.

Parameters **video** – The VideoPuppeteer instance that we are interested in.

Returns True if we are close enough to the end of playback; False otherwise.

YoutubePuppeteer

youtube_puppeteer.YouTubePuppeteer

class external_media_tests.media_utils.youtube_puppeteer.**YouTubePuppeteer**(*marionette*, *url*, ***kwargs*)
Bases: *external_media_tests.media_utils.video_puppeteer.VideoPuppeteer*

Wrapper around a YouTube #movie_player element.

Partial reference: https://developers.google.com/youtube/js_api_reference. This reference is useful for site-specific features such as interacting with ads, or accessing YouTube's debug data.

ad_ended

Returns True if the current ad has ended.

ad_format
When ad is not playing, ad_format is False.

Returns integer representing ad format, or False

ad_inactive

Returns True if the current ad is inactive.

ad_playing

Returns True if an ad is playing.

ad_skippable

Returns True if the current ad is skippable.

ad_state
Get state of current ad.

Returns Returns one of the constants listed in https://developers.google.com/youtube/js_api_reference#Playback_status for an ad.

attempt_ad_skip()
Attempt to skip ad by clicking on skip-add button.

Returns True if clicking of ad-skip button occurred.

breaks_count

Returns Number of upcoming ad breaks.

deactivate_autoplay()
Attempt to turn off autoplay.

Returns True if successful.

execute_yt_script(*script*)
Execute JS script in 'content' context with access to video element and YouTube #movie_player element.

Parameters **script** – script to be executed.

Returns value returned by script

movie_id

Returns The string that is the YouTube identifier.

movie_title

Returns The title of the movie.

playback_quality
Please see https://developers.google.com/youtube/js_api_reference#Playback_quality for valid values.

Returns A string with a valid value returned via YouTube.

player_buffering

Returns True if the video is currently buffering.

player_cued

Returns True if the video is cued.

player_current_time

Returns Current time in seconds via YouTube API.

player_duration

Returns Duration in seconds via YouTube API.

player_ended

Returns True if the video playback has ended.

player_measure_progress()

Returns Playback progress in seconds via YouTube API.

player_pause()
Pause via YouTube API.

player_paused

Returns True if the video is paused.

player_play()
Play via YouTube API.

player_playing

Returns True if the video is playing.

player_remaining_time

Returns Remaining time in seconds via YouTube API.

player_stalled

Returns True if playback is not making progress for 4-9 seconds. This excludes ad breaks. Note that the player might just be busy with buffering due to a slow network.

player_state

Returns The YouTube state of the video. See https://developers.google.com/youtube/js_api_reference#getPlayerState for valid values.

player_unstarted
This and the following properties are based on the player.getPlayerState() call (https://developers.google.com/youtube/js_api_reference#Playback_status)

Returns True if the video has not yet started.

player_url

Returns The YouTube URL for the currently playing video.

process_ad()
Try to skip this ad. If not, wait for this ad to finish.

search_ad_duration()

Returns ad duration in seconds, if currently displayed in player

youtube_puppeteer.playback_started

external_media_tests.media_utils.youtube_puppeteer.**playback_started**(*yt*)
Check whether playback has started.

Parameters **yt** – YouTubePuppeteer

youtube_puppeteer.playback_done

external_media_tests.media_utils.youtube_puppeteer.**playback_done**(*yt*)
Check whether playback is done, skipping ads if possible.

Parameters **yt** – YouTubePuppeteer

youtbue_puppeteer.wait_for_almost_done

external_media_tests.media_utils.youtube_puppeteer.**wait_for_almost_done**(*yt*, *final_piece=120*)

Allow the given video to play until only *final_piece* seconds remain, skipping ads mid-way as much as possible. *final_piece* should be short enough to not be interrupted by an ad.

Depending on the length of the video, check the ad status every 10-30 seconds, skip an active ad if possible.

Parameters **yt** – YouTubePuppeteer

2.2.2 external_media_tests.utils.verbose_until

external_media_tests.utils.**verbose_until**(*wait*, *target*, *condition*, *message=''*)

Performs a *wait*.until(condition)‘ and adds information about the state of *target* to any resulting *TimeoutException*.

Parameters

- **wait** – a *marionette.Wait* instance
- **target** – the object you want verbose output about if a *TimeoutException* is raised This is usually the input value provided to the *condition* used by *wait*. Ideally, *target* should implement *__str__*
- **condition** – callable function used by *wait.until()*
- **message** – optional message to log when exception occurs

Returns the result of *wait.until(condition)*

2.2.3 external_media_tests.utils.save_memory_report

external_media_tests.utils.**save_memory_report**(*marionette*)

Saves memory report (like about:memory) to current working directory.

Parameters **marionette** – Marionette instance to use for executing.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

R

S

T

U

V

W

Y