External Media Tests Documentation

Release 0.1

Syd Polk and Maja Frydrychowicz

Oct 08, 2017

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

external_media_harness package

Test case classes for use in video tests

external_media_harness.testcase module

class external_media_harness.testcase.**MediaTestCase**(**args*, ***kwargs*)
Bases: firefox_puppeteer.mixins.PuppeteerMixin, marionette_harness.marionette_test.testcases.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(*video*)
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(*timeout=60*)
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.

external_media_tests package

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

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*, *autostart=True*)
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 state is checked. 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.

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

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

pause()
Tell the video element to Pause.

play()
Tell the video element to Play.

playback_done()
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 **self** – The VideoPuppeteer instance that we are interested in.

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

playback_started()
Determine if video has started

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

Returns True if is playing; False otherwise

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

video_puppeteer.playback_done

YoutubePuppeteer

youtube_puppeteer.YouTubePuppeteer

class external_media_tests.media_utils.youtube_puppeteer.**YouTubePuppeteer**(*marionette*, *url*, *autostart=True*, ***kwargs*)

Bases: *external_media_tests.media_utils.video_puppeteer.VideoPuppeteer*

Wrapper around a YouTube .html5-video-player element.

Can be used with youtube videos or youtube videos at embedded URLS. E.g. both https://www.youtube.com/watch?v=AbAACm1IQE0 and https://www.youtube.com/embed/AbAACm1IQE0 should work.

Using an embedded video has the advantage of not auto-playing more videos while a test is running.

Compared to video puppeteer, this class has the benefit of accessing the youtube player object as well as the video element. The YT player will report information for the underlying video even if an add is playing (as opposed to the video element behaviour, which will report on whatever is play at the time of query), and can also report if an ad is playing.

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

mse_enabled()
Check if the video source indicates mse usage for current video. Refreshes state.

Returns True if MSE is being used, False if not.

playback_done()
Check whether playback is done. Refreshes state.

Returns True if play back has ended, False if not.

playback_started()
Check whether playback has started. Refreshes state.

Returns True if play back has started, False if not.

player_pause()
Pause via YouTube API.

player_play()
Play via YouTube API.

process_ad()
Wait for this ad to finish. Refreshes state.

wait_for_almost_done(*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.

This call refreshes state.

Parameters **final_piece** – The length in seconds of the desired remaining time

to wait until.

youtube_puppeteer.playback_started

youtube_puppeteer.playback_done

youtbue_puppeteer.wait_for_almost_done

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)*

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

Index

W

Y