# Context Capture Library

Smartling's Context Capture Library is a javascript library, which can be embedded in a web page to automatically send HTML snapshots of the current page state to a Smartling Project.

## Embedding the Library

## JavaScript Embedding

For a Single-Page Application or any other Javascript-based page, use the following snippet to load the Context Capture Library:

```
   (function (w, o) {
      try {
        var h = document.getElementsByTagName('head')[0];
        var s = document.createElement('script');
        s.type = 'text/javascript';
        s.async = 1;
        s.crossorigin = 'anonymous';
        s.src = '//d2c7xlmseob604.cloudfront.net/tracker.min.js';
        s.onload = function () {
          w.SmartlingContextTracker.init({ orgId: o });
        };
        h.insertBefore(s, h.firstChild);
      } catch (ex) {
      }
    })(window, 'xbPHx3Meq7RDCMuPKxeb7w')
```

See a full example here

## HTML Embedding

Just reference the script, and initialize the SmartlingContextTracker object

```
<script type="text/javascript"
src="//d2c7xlmseob604.cloudfront.net/tracker.min.js"></script>
<script>
    SmartlingContextTracker.init({
      orgId: 'xbPHx3Meq7RDCMuPKxeb7w'
    });
</script>
```

See a full example here

## Control the Library

Following methods can be used to temporarily enable/disable the library. Call them from JS console or your code.

- `SmartlingContextTracker.disable()` Disable capture indefinitely for current browser
- `SmartlingContextTracker.enable()` Re-enable capture after it has been disabled or paused
- `SmartlingContextTracker.pause(intervalInMinutes)` Disable capture for specified number of minutes

## Technical Information

## Performance

The Context Capture Library is carefully designed to have the least possible performance and memory impact on the page. Taking into account performance factors such as *page load*, *ui thread*, *upload bandwidth* and *DOM interaction*, the library implementation is carefully constructed to avoid any issues with performance or browser support..

### Page Load

The following steps were taken in order to make sure the script doesn't impact page load times::

- Asynchronous loading Unless the page has older design pattern, where scripts are loaded sequentially via <script> tags (see HTML Embedding), theContext Capture library is loaded via a tiny snippet, which, in turn, loads the actual library asynchronously, without any impact on page loading time.
- Small footprint *tracker.min.js* is about 50k, smaller than most images.
- Fetch time The context capture library script is edge-cached on Amazon Cloudfront CDN, and takes under 20ms to fetch. This allows the browser to complete requests fast, without exhausting limited number of concurrent resource requests. *tracker.min.js* is served to your visitors from a system of strategically positioned servers around the globe, which offers both fast loading and better availability.

Summary The Context Capture Library will not slow your page-load time, and, if it becomes unavailable for any reason, it will not impact your page.

**UI Thread Utilization**

The browser's most precious resource is CPU time spent in the main UI thread. All javascript is executed on this thread and any delay of between 50ms and 100ms will be noticeable to the user. The Context Capture Library avoids these "hangs" by leveraging user event processing in conjunction with built-in page mutation events and throttling the event handlers with the corresponding interval. All event processing is non-blocking and control from the Library is returned as fast as possible to the browser event loop. The only CPU intensive task in the thread is updating the document elements that have new or changed visible text. We clone the document structure and instrument cloned elements with appropriate CSS classes, leaving original DOM intact. The HTML of the cloned DOM tree is then sent to*web worker* - a separate background thread running in the browser.

## DOM and Script interaction

The following identifiers are exposed to the global scope by the script:

`SmartlingContextTracker`

The script also uses the `slTranslate` DOM node property as an expando to track nodes that needs text content matched.

## Resource extraction and upload

Pretty much any HTML document that is used to represent a web page will have a number of associated resources (style sheets, images and fonts) linked to the document content. Capturing these resources along with HTML content is essential for displaying accurate context to translators.

We do our best to capture resources snapshots by processing the uploaded HTML snapshot the following way:

- CSS stylesheets, images and font links are extracted from the HTML snapshot.
- Extracted links are rewritten to point to Smartling resource storage.
- Actual resources are captured and uploaded to Smartling resource storage.

This way we can ensure that the captured HTML context will look exactly as it looked at the time it was captured, even if original style sheets, images or fonts no longer available. Smartling uses several methods to fetch the resources, which cover most of the use cases*

- Publicly available resources that reside on the same domain as the original page.
- Protected resources (private network or firewalled) that reside on the same domain as the original page.
- Publicly available resources that reside on a different domain than the original page.

*Smartling will not be able to fetch protected resources that reside on different domain than the original page, unless CORS headers are properly configured for that domain.