Secure Browser Requirements and Specifications

Preview v0.3—18 July 2014

Introduction

When delivered online, high-stakes assessments rely on a number of mechanisms to ensure that students do not have access to unauthorized resources when taking the test. One of these tools is a secure browser. A secure browser prevents the student from accessing unauthorized applications or websites while the test is underway. In essence, it "locks down" the student's computing for the duration of the test.

The Smarter Balanced Assessment Consortium (Smarter Balanced) has sponsored the development of a set of secure browsers for devices commonly used in education. These browsers have been developed by the American Institutes for Research (AIR) and are being released under an open license on http://smarterapp.org in the fall of 2014. They will be used by Smarter Balanced assessments and other assessments that make use of the SmarterApp assessment delivery platform.

As new devices are released and existing devices receive software upgrades, new or updated secure browsers will be required to support assessment on those devices. This document includes the requirements and specifications for such secure browsers. Smarter Balanced hopes that these specifications can evolve into common practices for education and assessment efforts well beyond Smarter Balanced and SmarterApp.

These specifications are based on those published by AIR and are used with its permission.

Browser Requirements

For Smarter Balanced assessments, the following browser requirements are in addition to the "Technology Strategy Framework and Testing Device Requirements" document posted on the SmarterBalanced.org Technology Page.

Secure browsers to be used with the SmarterApp platform must support all the following web features. Each feature name includes a link to the corresponding specification.

Core platform

- HTML 5
- CSS
- DOM
- JavaScript (based on ECMAScript)

CSS features

- Animations
- Background-image options
- Border images
- Border radius (rounded corners)
- Box shadows
- Box sizing

- Downloadable fonts (@font-face)
- Fixed positioning (position:fixed)
- Gradients
- Hyphenation
- Media queries
- Multiple-column layout
- Multiple backgrounds
- Opacity
- Pointer events
- Selectors (level 3)
- Text overflow
- Text shadows
- Transforms (2D)
- Transitions

Graphics and typography

- SVG
- Canvas
- WebGL
- WOFF
- MathML

Platform interaction, events, messaging

- Device orientation
- DOM events
- Touch events
- Fullscreen

Storage and files

- Offline web applications
- Web storage

Foundations

- HTTP and URL
- TLS and X.509
- Cookies
- Unicode
- Origin
- MIME sniffing
- Encoding

API Specifications

In addition to the APIs specified in the web features listed in the previous section, a secure browser must expose a window global object called “browser.” The following methods are exposed on the “browser” object.

Required methods

void browser.security.enableLockDown (boolean lockDown)

Lock down environment to begin an assessment. The testing application will invoke this call prior to allowing students to start testing. The implementer is required to take any actions necessary to secure the testing environment. The steps taken to secure the environment are device-specific and include, for example, aspects such as disabling the ability to do screen captures, disabling the ability to voice chat when in secure mode, clearing the system clipboard, entering into a kiosk mode, disabling Spaces in OS X 10.7+, etc. The testing application will enable lockdown before an assessment commences and will disable the lockdown when the student has completed the assessment and is exiting the testing application.

boolean browser.security.isEnvironmentSecure()

Check if the environment is secure. The testing application will invoke this prior to allowing students to start testing and periodically when inside the test. The return type is boolean. This call must return true only if all necessary locks have been enabled (or necessary features disabled) to enable a secure testing environment and none of these have been compromised since lockdown mode was entered.

void browser.security.clearCache()

Clear browser cache. The testing application will invoke this call to clear any cached web resources. No secure content is ever marked with cache-control headers permitting it to be cached. However, static, non-secure resources, such as CSS files, header/footer graphics, etc., are marked with cache headers allowing them to be cached on the client machine (for performance reasons). This API call allows the testing web application to clear any such cached resources.

void browser.security.clearCookies()

Clear cookies. The testing application will invoke this call to clear any client-side cookies held in the browser's memory. This is a safety precaution to ensure that no cookies from any previous testing sessions are currently active. This is a backup to the server-side clearing of cookies.

string[] browser.security.getProcessList()

Retrieve current list of running processes. The testing application will invoke this to retrieve a list of all processes owned by the user context running on the client machine. This list is used to determine if the user is running any processes that have been deemed blacklisted during the testing cycle. This call will be invoked both at the start of an assessment and periodically when the student is taking the assessment, and, at any point, if a blacklisted app is detected, the assessment will be stopped to preserve test integrity.

Example response:

"['taskmgr.exe','chrome.exe','ccSvcHst.exe','Dropbox.exe','EXCEL.EXE','svchost.exe','System']"

void browser.security.close(boolean restart)

Shut down the browser. The testing application will invoke this to close the browser when the user elects to exit the browser. The boolean parameter indicates whether the browser should restart on exit or should simply exit.

void browser.tts.speak(string text, object options, function callback)

Speak text (text-to-speech synthesis). The testing application will invoke this to perform client-side text-to-speech (TTS) synthesis. The API call will be passed in a string with embedded speech markup, an options object to control the speech (optional), and a callback for TTS events (optional). The vendor can support one of the following markup standards: SSML, Microsoft speech markup (for Windows), or Apple speech markup (for OS X). The options object includes the following properties: voicename, rate, gender, language, pitch, and volume. The callback, if provided, is invoked for TTS events which include start, end, word boundary, sentence boundary, synchronization/marker encountered, paused, and error.

void browser.tts.stop()

Stop speech (text-to-speech synthesis). This is called by the testing application to stop any speech that may be in progress.

string browser.tts.getStatus()

Get speech status (text-to-speech synthesis). This is called by the testing application to inspect the current status of speech.

Where Status is one of:

- NOTSUPPORTED - tts initialization failed
- UNINITIALIZED - tts is not initialized
- INITIALIZING - tts initialization in progress
- STOPPED - tts is initialized and there is nothing playing
- PLAYING - playing is in progress
- PAUSED - playing was paused
- UNKNOWN – unknown status

string[] browser.tts.getVoices()

Get available voices (text-to-speech synthesis). This is called by the testing application to get a listing of the available voice packs in the current system.

Example Response:

['US English Female TTS','en-US','es-ES']

void browser.tts.pause()

Pause speech (text-to-speech synthesis). This is called by the testing application to temporarily pause speech. Corresponding events are fired to notify the callback provided in the speak function of the pause event.

void browser.tts.resume()

Resume speech (text-to-speech synthesis). This is called by the testing application to resume speech if it was previously paused.

Optional Audio Recording Methods

The following methods are for audio recording of student responses. They aren't required for the current Smarter Balanced assessment but they will be useful for other assessments and Smarter Balanced may add audio response items in the future.

void browser.recorder.initialize (eventListener)

Initialize audio recorder. This method is called by the testing application once to initialize the audio recording API after a page loads. The event listener passed in as argument is used to send notify events about progress. Any attempts to call this method when it has already been called should be treated as a reset and reinitialization.

Events expected:

- INITIALIZING—indicates that initialization is in progress.
- READY—Initialization is done and internal data structures are loaded.
- ERROR—Initialization failed, with information on failure cause.

string browser.recorder.getStatus()

Get audio recorder status. This method is called by the testing application to inquire about the status of the recorder. Return values expected are:

- IDLE—no recording in progress.
- ACTIVE—recording in progress.
- INITIALIZING—initialization in progress.
- ERROR—terminal error state; reinitialization is required.
- STOPPING—Recording is done and final bookkeeping and generation of encoded audio are in progress.

object browser.recorder.getCapabilities()

Get audio recorder capabilities. This method is called to inquire about the capabilities of the platform. Throws an error if called before initialization is completed successfully. The return values include:

- isAvailable—Recording is supported (Boolean).
- supportedInputDevices—a list of audio input devices detected. Each of entry in the list includes: device ID, device description/label, supported sample size(s), supported sample rate(s), supported channel count(s), and supported encoding format(s).

object browser.recorder.startCapture(options, eventListener)

Initiate audio capture. This method is called to initiate audio capture. Throws an error if called prior to successful initialization. Throws errors if the options passed in are not supported on the device. Throws an error if capture status is currently not IDLE.

The options object includes the following properties:

- captureDevice—the device ID to use for data capture (int).
- sample rate—the line rate to capture the raw audio in (e.g., 8 Khz, 11 Khz, . . .) (specified as int in hz).

- channel count—e.g., 1 (mono), 2 (stereo), . . . (specified as int).
- sample size—e.g., 8-bit, 16-bit, . . . (specified as int)
- encoding format—e.g., SPX, HE-AAC, Opus . . . (specified as string).
- quality indicator desired—whether to perform and report a recording quality check or not (Boolean).
- progressEventFrequency—how frequently the event listener should be called back to report progress events, either based on time or based on units of data collected. For example, the testing application could ask for periodic progress events every two seconds, in order to be notified as recording is happening, or at every 30 KB of new data collected.
- captureLimit—object literal that specifies data capture time or size after which the recorder should automatically stop capturing and fire an end event (specified as {duration:40} or {size:250}; duration unit is seconds, and size unit is KB).

The event listener is passed in to receive capture events. Expected events are:

- START—capture started.
- INPROGRESS—progress event, with progress data (e.g., 34 seconds of audio captured, 36 seconds of audio captured, ..., or 10 KB of audio captured, 30 KB of audio captured . . .)
- END—capture complete. The END event is important because this event provides the pointer to the data collection for the encoded audio. In addition, a quality check is performed on the captured audio stream, to evaluate whether it is good or not, and a unique ID for the recording is created.

void browser.recorder.stopCapture()

Stop recording. This method is called to stop audio capture. Throws an error if status is not currently "RECORDING."

string browser.recorder.retrieveAudio(id)

Retrieve recording. This method is called to retrieve base64-encoded audio data that was previously captured.

string browser.security.getDeviceInfo()

Retrieve device details. This method is called to retrieve detailed information about a device. The returned data must include the manufacturer name, device model number (with hardware revision number, if available), and OS version (major, minor, and build number). The format of this string is "<key>=<value>" pairs separated by the | symbol. The valid keys are "Manufacturer", "HWVer", and "SWver."

Optional Methods

void browser.security.emptyClipBoard()

Empty system clipboard (optional). The testing application will invoke this to force-clear any data that may be in the system clipboard. This is an optional method; if the browser.security.enableLockDown method clears the clipboard then this method is not required..

string[] browser.security.getMACAddress()

Retrieve system MAC address(es) (optional). The testing application will invoke this to assist in diagnostics. It is difficult to rely on source IP addresses to distinguish among end-user machines within testing servers, as firewalls, network address translations (NATs), and/or proxies are commonly in use at schools. The MAC addresses allow for distinctions among end client machines behind a common firewall, for diagnostic purposes.

Example response:

"['00','55','65','C0','00','EA']"

string[] browser.security.getIPAddressList()

Retrieve client IP address(es) (optional). The testing application will invoke this to assist in diagnostics. A listing of retrieved IP addresses can be presented on the client-side diagnostics screens.

Example response:

"['192.168.7.100','172.22.38.45']"

DateTime browser.security.getStartTime()

Get application start time (optional). The testing application will invoke this to determine the local client-side time at which the application was launched. This is mainly used to track application uptime. If this is not provided, the web application can track start time using local/session storage, but it is desirable to have this information natively supported.

Example response:

"Thu May 29 2014 17:35:24 GMT-0500 (Central Standard Time)"

Appendix A: Bonus Web Browser Features

The original specifications, as published by AIR, include the following "bonus" web browser features. These features are not required by Smarter Balanced but they should be considered in case of future needs.

HTML bonus features

- ARIA
- classList (DOMTokenList)
- Datasets
- async for scripts
- defer for scripts
- hashchange
- Drag and drop
- contentEditable
- HTML editing APIs

Real-time communication

- WebSocket protocol and WebSocket API
- XMLHttpRequest

Performance optimization and analysis

- Web workers
- Shared workers
- Page Visibility

Security and privacy

- Cross-Origin Resource Sharing (CORS)

Other core-platform bonus features

- Selectors API
- matches(*selector*) method
- matchMedia (*MediaQueryList*) method
- data URLs
- JSON parsing
- Quirks mode
- DOM parsing and serialization
- DOM XPath
- XML, XPath, XSLT, and xml-stylesheet