JS API Documentation

Embed Files

To access API functionality, the following script embed is required. This includes the CWMN API without our standard UI.

```
<script src="https://assets.clickwith.me/clickwithmenow-api.js"></script>
```

CwmnAPI Functions

These functions are exposed on the global `CwmnAPI` object (`window.CwmnAPI`).

Name/params	Valid for State	Parameters object values	Valid in Session Type (if applicable)	Description
`on(eventName, callback)`	All (although only the `init` event can be subscribed to before `init` fires)			Register for a callback for when `event` fires.
`once(eventName, callback)`	All (although only the `init` event can be subscribed to before `init` fires)			Register a callback for the next time `event` occurs (`callback` will be called at most once). Returns a function that when called, deregisters `callback` from `event`. Otherwise equivalent to `.on`
`off(eventName, callback)`	All			Remove a callback that was registered with on
`getCurrentState()`	All			Returns the current state (see "states" above)
`isCompatible()`	All			Returns true/false if the current browser is considered compatible
`getDomainInfo()`	All			Returns info about the current domain and CWMN features/settings for that domain
`session.start(params)`	`no session`	`hostName` (string: name of user hosting session), `hostEmail` (string: email address of user hosting session), `isReplay` (boolean: is this a replay session)		Starts a new session ("from me" realtime or replay). Returns: promise

`session.inviteToHost(params)`	`no session`	`hostName` (string: name of user sending the invite to host), `hostEmail` (string: email address of user hosting session), `inviteName` (string: name of user being invited), `inviteEmail` (string: email of user being invited), `message` (string: personal message to be included in the email invite), `url` (string: the full URL where the user's session will begin), `ithJoinCode` (string: join code for the user to begin hosting)		Sends and invite to host a session ("with me" realtime). Either hostEmail of ithJoinCode must be specified. If ithJoinCode is used, the user hosting the session will join and begin hosting at https://m.clickwith.me/join. Returns: promise
`session.endSession()`	`in session`		realtime and replay	Ends the current session being hosted
`session.getSessionInfo()`	`in session`		realtime and replay	Returns info about the current session
`guest.inviteGuest(params)`	`in session`	`guestName` (string: name of user being invited), `guestEmail` (string: email of user being invited), `guestPhone` (string: phone number for SMS invite), `message` (string: personal message to be included with email invite)	realtime	Sends an email or SMS invite to a guest. Either `guestEmail` or `guestPhone` is required. `message` is not used/required for SMS invites.
`guest.getInviteLink()`	`in session`		realtime	Returns a promise that fulfills with a one-time invite link ('private link') that can be manually sent by the host to a guest
`guest.removeGuest(guestId)`	`in session`		realtime	Removes a guest from the current session
`guest.muteGuest(guestId, isHidden)`	`in session`		realtime	Mutes/unmutes the mouse pointer of a guest in the session. `isHidden` specifies the new state of the guest's mouse pointer.
`guest.muteAllGuests(isHidden)`	`in session`		realtime	Mute/unmute all guests at once (independent of the mute status of individual guests)
`guest.getMuteAllGuests()`	`in session`		realtime	Gets the current status of `muteAllGuests`
`guest.getGuestList()`	`in session`		realtime	Returns an array of guest objects for the current session
`publicLink.setEnabled(enabled)`	`in session`		realtime	Sets the enabled status of the public link feature
`publicLink.getEnabled()`	`in session`		realtime	Returns the enabled status of the public link feature
`publicLink.generate()`	`in session`		realtime	Returns the public link that can be given for guests to join
`privacy.setEnabled(enabled)`	`in session`		realtime	Enable/disable the host privacy feature

`privacy.getEnabled()`	`in session`		realtime	Get the enabled status of the host privacy feature
`replay.muteAudio(muted)`	`in session`		replay	Mute audio recording in a replay session

`replay.sendVideo(params)`	`waiting for replay`
	
	`replay complete`
		`toEmails` (string: comma-separated list of emails that should receive the video),
		
		`subject` (string: subject for the email message),
		
		`message` (string: personal message included in the email)
				Send the video for a replay session. If in the `replay complete` state, the email is sent immediately. Otherwise, the video is sent when rendering is complete.

`replay.getVideoURL()`	`replay complete`			Returns the video URL for a rendered replay video

`replay.abandonVideo()`	`waiting for replay`
	
	`replay complete`
				Abandons a completed replay session without sending the video

`replay.getRenderingProgress()`	`waiting for replay`
	
	`replay complete`
				Gets rendering progress and completed status for a replay session

`replay.dismissError()`	`replay error`			Call to dismiss the `replay error` state and return to `no session`

Events

Client code can subscribe to events using `CwmnAPI.on` and `CwmnAPI.once`.

Name	Parameters	Occurs in State(s)	Description
`init`		`init`	Fired when CWMN is initializing the first time on a page. The DOM is guaranteed to be ready, and client code can now subscribe to other events.
`reinit`		`init`	Fired when CWMN is reinitialized subsequent times on a page, such as after a session is over.

`init failed`		`incompatible`
		
		`unauthorized`
			Fired if either the browser is incompatible with CWMN or the domain is not authorized.

`incompatible`		`incompatible`	Fired on initialization if the current browser is incompatible with CWMN
`unauthorized`		`unauthorized`	Fired on initialization if the current domain is not authorized to use CWMN
`ready no session`		`no session`	Fired when CWMN is ready to begin a session and no session exists
`ith init`		`session pending`	Emitted in the `session pending` state to indicate that an ITH session is initializing on this page load
`session pending`		`session pending`	Fired when CWMN is in the process of connecting to a session
`in session`	`bool:isResuming`	`in session`	Fired when hosting begins for a session. `isResuming` indicates whether we're resuming an existing session or starting a new one
`session failed`		`session pending`	Fired when an error occurs connection to a session (occurs after `session pending`)
`expired`		`in session`	Fired before `end session` when the session is ending due to screen expiration on the server.
`end session`		`in session`	Fired when hosting ends for a session - CWMN transitions back to `no session` from a realtime session, and to `waiting for replay` from a replay session
`connection issue`		`in session`	When a network issue occurs that is preventing realtime communication

`connection issue resolved`		`in session`	After `connection issue` occurs; when the issue is no longer present and communication has resumed
`guest connected`	`string:guestId`, `object:guest`	`in session`	Fired when a guest connects to the session. Callback receives the guest id and guest object.
`guest disconnected`	`string:guestId`, `object:guest`	`in session`	Fired when a guest connects to the session. Callback receives the guest id and guest object.
`roomdata update`	`array:roomdata`	`in session`	Fired when the `roomdata` is updated. This could be when a guest is added or remove, or when a user's mouse coordinates change. Callback receives the current guest list (same as returned by `CwmnAPI.guest.getGuestList`)
`click`	`string:guestId`	`in session`	Fired when guest with id `guestId` has clicked their mouse
`waiting for replay`		`waiting for replay`	Fired when CWMN enters the `waiting for replay` state
`replay progress`	`object:replayProgress`	`waiting for replay`	When replay rendering progress update is received
`replay complete`		`replay complete`	When replay video finishes rendering successfully
`replay error`		`replay error`	When replay video fails to render due to an error
`replay timeout warning`	`string: warningMessage`	`in session`	When replay video is approaching its maximum length
`replay timeout`		`in session`	When replay video has hit its maximum length and is stopping, in order to start the video rendering

States

The CWMN client has a number of states that determine CWMN is doing and what actions are available. The current state can be determined using `CwmnAPI.getCurrentState`, and client code can observe transitions between events using `CwmnAPI.on` and `CwmnAPI.once`.

Name	Description
`init`	CWMN is initializing upon page load. It is not yet known whether a session exists or can be created.
`incompatible browser`	The browser is not compatible with CWMN.
`unauthorized`	The page's domain is not authorized to use CWMN.
`no session`	CWMN is initialized and no session currently exists. A new session can be created.
`session pending`	CWMN is in the process of initializing a session or connecting to an existing session
`in session`	Initialization is finished - user is currently hosting a session
`waiting for replay`	Not hosting a session, waiting for a replay video to render so the user can preview/send
`replay complete`	Replay rendering is complete, user can preview
`replay error`	There was an error rendering the replay video

Example/Suggested Usage

```
CwmnAPI.on('init', function () { //DOM is ready, CWMN is ready for other event
handlers. that's it at this point.

 CwmnAPI.on('unauthorized', function () {
        alert('unauthorized');
    });

    CwmnAPI.on('incompatible', function () {
```

```
        alert('incompatible');
    });

    CwmnAPI.on('ready no session', function() {
        alert('ready to start session');
    });

//called elsewhere to send an invite to host
    function sendInviteToHost() {
        CwmnAPI.session.inviteToHost({
            hostName: 'Click With Me Now demo',
            hostEmail: 'noreply@clickwithmenow.com',
            inviteName: name,
            inviteEmail: email,
            message: message,
            url: url
        }).then(function (result) {
            console.log(result);
        }).catch(function (error) {
            console.log(error);
        });
    }

//called elsewhere to start hosting a session
    function startHosting() {
        CwmnAPI.session.start({
            hostName: 'Host Name',
            hostEmail: 'noreply@clickwithmenow.com',
            isReplay: false
        }).then(function (result) {
            alert('You are now hosting a session');
        }).catch(function (error) {
            console.log(error);
        });
    }

//called elsewhere to start recording a replay
    function startReplay() {
        CwmnAPI.session.start({
            hostName: 'Host Name',
            hostEmail: 'noreply@clickwithmenow.com',
            isReplay: true
        }).then(function (result) {
            alert('You are now recording a replay.');
        }).catch(function (error) {
            console.log(error);
        });
    }

//generate a private link for the current session
    function privateLink() {
        CwmnAPI.guest.getInviteLink().then(function(result) {
            console.log(result)
        }).catch(function(error) {
            console.log('error');
        });
    }

    function emailInvite() {
```

```
    CwmnAPI.guest.inviteGuest({
        guestName: 'Guest Name',
        guestEmail: 'noreply@clickwithmenow.com',
        message: 'Join my session!'
    }).then(function(result) {
        console.log(result);
    }).catch(function(error) {
        console.log(error);
    });
}

function smsInvite() {
    CwmnAPI.guest.inviteGuest({
        guestName: 'Guest Name',
        guestPhone: '3145550000',
    }).then(function(result) {
        console.log(result);
    }).catch(function(error) {
        console.log(error);
    });
}

function endSession() {
```

```
        CwmnAPI.session.endSession();
    }
});
```