html5.h

The C++ APIs in **html5.h** define the Emscripten low-level glue bindings to interact with **HTML5** events from native code.

> **Tip**
>
> The C++ APIs map closely to their equivalent **HTML5** JavaScript APIs. The **HTML5** specifications listed below provide additional detailed reference "over and above" the information provided in this document.
>
> In addition, the Test/Example code can be reviewed to see how the code is used.

The **HTML5** specifications for APIs that are mapped by **html5.h** include:

- DOM Level 3 Events: Keyboard, Mouse, Mouse Wheel, Resize, Scroll, Focus.
- Device Orientation Events for gyro and accelerometer.
- Screen Orientation Events for portrait/landscape handling.
- Fullscreen Events for browser canvas fullscreen modes transitioning.
- Pointer Lock Events for relative-mode mouse motion control.
- Vibration API for mobile device haptic vibration feedback control.
- Page Visibility Events for power management control.
- Touch Events.
- Gamepad API.
- Beforeunload event.
- WebGL context events

Table of Contents

- Device orientation
- Device motion
- Orientation
- Fullscreen
- Pointerlock
- Visibility
- Touch
- Gamepad
- Battery
- Vibration
- Page unload
- WebGL context
- CSS

How to use this API

Most of the APIs use an event-based architecture; functionality is accessed by registering a callback function that will be called when the event occurs.

> **Note**
>
> The Gamepad API is currently an exception, as only a polling API is available. For some APIs, both an event-based and a polling-based model are exposed.

Registration functions

The typical format of registration functions is as follows (some methods may omit various parameters):

```
EMSCRIPTEN_RESULT emscripten_set_some_callback(
  const char *target,   // ID of the target HTML element.
  void *userData,   // User-defined data to be passed to the callback.
  EM_BOOL useCapture,   // Whether or not to use capture.
  em_someevent_callback_func callback   // Callback function.
);
```

The `target` parameter is the ID of the HTML element to which the callback registration is to be applied. This field has the following special meanings:

- `0` or `NULL`: A default element is chosen automatically based on the event type, which should be reasonable most of the time.
- `#window`: The event listener is applied to the JavaScript `window` object.
- `#document`: The event listener is applied to the JavaScript `document` object.
- `#screen`: The event listener is applied to the JavaScript `window.screen` object.
- `#canvas`: The event listener is applied to the Emscripten default WebGL canvas element.
- Any other string **without a leading hash "#"** sign: The event listener is applied to the element on the page with the given ID.

The `userData` parameter is a user-defined value that is passed (unchanged) to the registered event callback. This can be used to, for example, pass a pointer to a C++ class or similarly to enclose the C API in a clean object-oriented manner.

The `useCapture` parameter maps to `useCapture` in EventTarget.addEventListener . It indicates whether or not to initiate *capture*: if `true` the callback will be invoked only for the DOM capture and target phases; if `false` the callback will be triggered during the target and bubbling phases. See DOM Level 3 Events for a more detailed explanation.

Most functions return the result using the type `EMSCRIPTEN_RESULT`. Zero and positive values denote success. Negative values signal failure. None of the functions fail or abort by throwing a JavaScript or C++ exception. If a particular browser does not support the given feature, the value `EMSCRIPTEN_RESULT_NOT_SUPPORTED` will be returned at the time the callback is registered.

Callback functions

When the event occurs the callback is invoked with the relevant event "type" (for example `EMSCRIPTEN_EVENT_CLICK`), a `struct` containing the details of the event that occurred, and the `userData` that was originally passed to the registration function. The general format of the callback function is:

```
typedef EM_BOOL (*em_someevent_callback_func) // Callback function. Return true if event is "consumed".
  (
  int eventType, // The type of event.
  const EmscriptenSomeEvent *someEvent, // Information about the event.
  void *userData // User data passed from the registration function.
  );
```

Callback handlers that return an `EM_BOOL` may specify `true` to signal that the handler *consumed* the event (this suppresses the default action for that event by calling its `.preventDefault();` member). Returning `false` indicates that the event was not consumed —

the default browser event action is carried out and the event is allowed to pass on/bubble up as normal.

Calling a registration function with a `null` pointer for the callback causes a de-registration of that callback from the given `target` element. All event handlers are also automatically unregistered when the C `exit()` function is invoked during the `atexit` handler pass. Either use the function *`emscripten_set_main_loop()`* or set `Module.noExitRuntime = true;` to make sure that leaving `main()` will not immediately cause an `exit()` and clean up the event handlers.

Functions affected by web security

Some functions, including *`emscripten_request_pointerlock()`* and *`emscripten_request_fullscreen()`*, are affected by web security.

While the functions can be called anywhere, the actual "requests" can only be raised inside the handler for a user-generated event (for example a key, mouse or touch press/release).

When porting code, it may be difficult to ensure that the functions are called inside appropriate event handlers (so that the requests are raised immediately). As a convenience, developers can set `deferUntilInEventHandler=true` to automatically defer insecure requests until the user next presses a keyboard or mouse button. This simplifies porting, but often results in a poorer user experience. For example, the user must click once on the canvas to hide the pointer or transition to full screen.

Where possible, the functions should only be called inside appropriate event handlers. Setting `deferUntilInEventHandler=false` causes the functions to abort with an error if the request is refused due to a security restriction: this is a useful mechanism for discovering instances where the functions are called outside the handler for a user-generated event.

Test/Example code

The **HTML5** test code demonstrates how to use this API:

- test_**html5**.c
- test_**html5**_fullscreen.c
- test_**html5**_mouse.c

General types

`EM_BOOL`

This is the Emscripten type for a `bool`. Possible values:

> `EM_TRUE`
>
> This is the Emscripten value for `true`.

> `EM_FALSE`
>
> This is the Emscripten value for `false`.

`EM_UTF8`

This is the Emscripten type for a UTF8 string (maps to a `char`). This is used for node names, element ids, etc.

Function result values

Most functions in this API return a result of type `EMSCRIPTEN_RESULT`. None of these functions fail or abort by throwing a JavaScript or C++ exception. If a particular browser does not support the given feature, the value `EMSCRIPTEN_RESULT_NOT_SUPPORTED` will be returned at the time the callback is registered.

`EMSCRIPTEN_RESULT`

This type is used to return the result of most functions in this API. Zero and positive values denote success, while negative values signal failure. Possible values are listed below.

> `EMSCRIPTEN_RESULT_SUCCESS`
>
> The operation succeeded.

> `EMSCRIPTEN_RESULT_DEFERRED`
>
> The requested operation cannot be completed now for web security reasons, and has been deferred for completion in the next event handler.

> `EMSCRIPTEN_RESULT_NOT_SUPPORTED`
>
> The given operation is not supported by this browser or the target element. This value will be returned at the time the callback is registered if the operation is not supported.

> `EMSCRIPTEN_RESULT_FAILED_NOT_DEFERRED`
>
> The requested operation could not be completed now for web security reasons. It failed because the user requested the operation not be deferred.

> `EMSCRIPTEN_RESULT_INVALID_TARGET`

T h e operation failed because the specified target element is invalid.

`EMSCRIPTEN_RESULT_UNKNOWN_TARGET`

T h e operation failed because the specified target element was not found.

`EMSCRIPTEN_RESULT_INVALID_PARAM`

T h e operation failed because an invalid parameter was passed to the function.

`EMSCRIPTEN_RESULT_FAILED`

Generic failure result message, returned if no specific result is available.

`EMSCRIPTEN_RESULT_NO_DATA`

T h e operation failed because no data is currently available.

Keys

Defines

`EMSCRIPTEN_EVENT_KEYPRESS` `EMSCRIPTEN_EVENT_KEYDOWN` `EMSCRIPTEN_EVENT_KEYUP`

Emscripten key events.

`DOM_KEY_LOCATION`

T h e location of the key on the keyboard; one of the values below.

`DOM_KEY_LOCATION_STANDARD` `DOM_KEY_LOCATION_LEFT` `DOM_KEY_LOCATION_RIG H T`

`DOM_KEY_LOCATION_NUMPAD`

Locations of t h e key on the keyboard.

Struct

`EmscriptenKeyboardEvent`

T h e event structure passed in keyboard events : `keypress`, `keydown` and `keyup`.

Note t h at since t h e DOM Level 3 Events spec is very recent at the time of writing (2014-03), uniform support for the different fields in the spec is still in flux. Be sure to check the results in multiple browsers. See the unmerged pull request #2222 for an example of how to interpret the legacy key events.

EM_UTF8 `key`

The printed representation of the pressed key.

Maximum size 32 `char` (i.e. `EM_UTF8 key[32]`).

EM_UTF8 `code`

A string that identifies the physical key being pressed. The value is not affected by the current keyboard layout or modifier state, so a particular key will always return the same value.

Maximum size 32 `char` (i.e. `EM_UTF8 code[32]`).

unsigned long `location`

Indicates the location of the key on the keyboard. One of the *`DOM_KEY_LOCATION`* values.

EM_BOOL `ctrlKey` | **EM_BOOL** `shiftKey` | **EM_BOOL** `altKey` | **EM_BOOL** `metaKey`

Specifies which modifiers were active during the key event.

EM_BOOL `repeat`

Specifies if this keyboard event represents a repeated press.

EM_UTF8 `locale`

A locale string indicating the configured keyboard locale. This may be an empty string if the browser or device doesn't know the keyboard's locale.

Maximum size 32 char (i.e. `EM_UTF8 locale[32]`).

EM_UTF8 `charValue`

The following fields are values from previous versions of the DOM key events specifications. See the character representation of the key . This is is the field `char` from the docs, but renamed to `charValue` to avoid a C reserved word.

Maximum size 32 `char` (i.e. `EM_UTF8 charValue[32]`).

> **Warning**
>
> This is attribute has been dropped from DOM Level 3 events.

unsigned long `charCode`

T **h** e Unicode reference number of t **h** e key; this attribute is used only by the keypress event. For keys whose `char` attribute contains multiple characters, this is the Unicode value of the first character in that attribute.

> **Warning**
>
> T **h** is attribute is deprecated, you s **h** ould use the field `key` instead, if available.

unsigned long `keyCode`

A system and implementation dependent numerical code identifying t **h** e unmodified value of the pressed key.

> **Warning**
>
> T **h** is attribute is deprecated, you s **h** ould use the field `key` instead, if available.

unsigned long `w h ich`

A system and implementation dependent numeric code identifying t **h** e unmodified value of t **h** e pressed key; this is usually the same as `keyCode`.

> **Warning**
>
> T **h** is attribute is deprecated, you s **h** ould use t **h** e field `key` instead, if available. Note thought that while this field is deprecated, the cross-browser support for `which` may be better than for the other fields, so experimentation is recommended. Read issue https://github.com/kripken/emscripten/issues/2817 for more information.

Callback functions

`em_key_callback_func`

Function pointer for t **h** e *`keypress callback functions`*, defined as:

```
typedef EM_BOOL (*em_key_callback_func)(int eventType, const EmscriptenKeyboardEvent *keyEvent, void
*userData);
```

Parameters:

- **eventType** (*int*) – The type of `key event`.
- **keyEvent** (*const EmscriptenKeyboardEvent**) – Information about the key event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_keypress_callback`(const char *target, void *userData, EM_BOOL useCapture, em_key_callback_func callback)

EMSCRIPTEN_RESULT `emscripten_set_keydown_callback`(const char *target, void *userData, EM_BOOL useCapture, em_key_callback_func callback)

EMSCRIPTEN_RESULT `emscripten_set_keyup_callback`(const char *target, void *userData, EM_BOOL useCapture, em_key_callback_func callback)

Registers a callback function for receiving browser-generated keyboard input events.

Parameters:

- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_key_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

See also:

- https://developer.mozilla.org/en/DOM/Event/UIEvent/KeyEvent
- http://www.javascriptkit.com/jsref/eventkeyboardmouse.shtml

Mouse

Defines

`EMSCRIPTEN_EVENT_CLICK` `EMSCRIPTEN_EVENT_MOUSEDOWN` `EMSCRIPTEN_EVENT_MOUSEUP`

`EMSCRIPTEN_EVENT_DBLCLICK` `EMSCRIPTEN_EVENT_MOUSEMOVE` `EMSCRIPTEN_EVENT_MOUSEENTER`

`EMSCRIPTEN_EVENT_MOUSELEAVE`

Emscripten mouse events.

Struct

`EmscriptenMouseEvent`

The event structure passed in mouse events : click , mousedown , mouseup , dblclick , mousemove , mouseenter and mouseleave .

`double timestamp;`

A timestamp of when this data was generated by the browser. This is an absolute wallclock time in milliseconds.

long `screenX` **long** `screenY`

The coordinates relative to the browser screen coordinate system.

long `clientX` **long** `clientY`

The coordinates relative to the viewport associated with the event.

EM_BOOL `ctrlKey` **EM_BOOL** `shiftKey` **EM_BOOL** `altKey` **EM_BOOL** `metaKey`

Specifies which modifiers were active during the mouse event.

unsigned short `button`

Identifies which pointer device button changed state (see MouseEvent.button):

- 0 : Left button
- 1 : Middle button (if present)
- 2 : Right button

unsigned short `buttons`

A bitmask that indicates which combinations of mouse buttons were being held down at the time of the event.

long `movementX` `long movementY;`

If pointer lock is active, the two extra fields give relative mouse movement since the last event.

long `targetX` **long** `targetY`

These fields give the mouse coordinates mapped relative to the coordinate space of the target DOM element receiving the input events (Emscripten-specific extension).

long `canvasX` **long** `canvasY`

These fields give the mouse coordinates mapped to the Emscripten canvas client area (Emscripten-specific extension).

long `padding`

Internal, and can be ignored.

Note

Implementers only: pad this struct to multiple of 8 bytes to make `WheelEvent` unambiguously align to 8 bytes.

Callback functions

`em_mouse_callback_func`

Function pointer for the *`mouse event callback functions`*, defined as:

```
typedef EM_BOOL (*em_mouse_callback_func)(int eventType, const EmscriptenMouseEvent *mouseEvent, void
*userData);
```

Parameters:

- **eventType** (*int*) – The type of *`mouse event`*.
- **mouseEvent** (*const EmscriptenMouseEvent**) – Information about the mouse event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *`EM_BOOL`*

Functions

EMSCRIPTEN_RESULT `emscripten_set_click_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_mousedown_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_mouseup_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_dblclick_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_mousemove_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_mouseenter_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_mouseleave_callback` **(const c h ar *target, void *userData, EM_BOOL *useCapture*, em_mouse_callback_func *callback*)**

Registers a callback function for receiving browser-generated mouse input events .

Parameters:

- **target** (*const c h ar**) – Target H TML element id.
- **userData** (*void**) – User-defined data to be passed to t h e callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_mouse_callback_func*) – A callback function. T h e function is called wit h the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of t h e other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_mouse_status` **(EmscriptenMouseEvent *mouseState)**

Returns t h e most recently received mouse event state.

Note t h at for t h is function call to succeed, `emscripten_set_xxx_callback` must have first been called with one of the mouse event types and a non-zero callback function pointer to enable the Mouse state capture.

Parameters:
- **mouseState** (*EmscriptenMouseEvent**) – The most recently received mouse event state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Wheel

Defines

`EMSCRIPTEN_EVENT_WHEEL`

Emscripten wheel event.

`DOM_DELTA_PIXEL`

The units of measurement for the delta must be pixels (from spec).

`DOM_DELTA_LINE`

The units of measurement for the delta must be individual lines of text (from spec).

`DOM_DELTA_PAGE`

The units of measurement for the delta must be pages, either defined as a single screen or as a demarcated page (from spec).

Struct

`EmscriptenWheelEvent`

The event structure passed in mousewheel events .

EmscriptenMouseEvent `mouse`

Specifies general mouse information related to this is event.

double `deltaX` **double** `deltaY` **double** `deltaZ`

Movement of the wheel on each of the axis. Note that these values may be fractional, so you should avoid simply casting them to integer, or it might result in scroll values of 0. The positive Y scroll direction is when scrolling the page downwards (page CSS pixel +Y direction), which corresponds to scrolling the mouse wheel downwards (away from the screen) on Windows, Linux, and also on OSX when the 'natural scroll' option is disabled.

unsigned long `deltaMode`

One of the `DOM_DELTA_` values that indicates the units of measurement for the delta values.

Callback functions

`em_wheel_callback_func`

Function pointer for the *wheel event callback functions*, defined as:

```
typedef EM_BOOL (*em_wheel_callback_func)(int eventType, const EmscriptenWheelEvent *wheelEvent,
void *userData);
```

Parameters:

- **eventType** (*int*) – The type of wheel event (*EMSCRIPTEN_EVENT_WHEEL*).
- **wheelEvent** (*const EmscriptenWheelEvent**) – Information about the wheel event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_wheel_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_wheel_callback_func callback)**

Registers a callback function for receiving browser-generated mousewheel events .

Parameters:

- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_wheel_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

UI

Defines

`EMSCRIPTEN_EVENT_RESIZE` `EMSCRIPTEN_EVENT_SCROLL`

Emscripten UI events.

Struct

`EmscriptenUiEvent`

The event structure passed in DOM element UIEvent events: resize and scroll.

long `detail`

Specifies additional detail/information about the event.

int `documentBodyClientWidth` **int** `documentBodyClientHeight`

The clientWidth/clientHeight of the `document.body` element.

int `windowInnerWidth` **int** `windowInnerHeight`

The innerWidth/innerHeight of the browser window.

int `windowOuterWidth` `int windowOuterHeight;`

The outerWidth/outerHeight of the browser window.

int `scrollTop` **int** `scrollLeft`

The page scroll position.

Callback functions

`em_ui_callback_func`

Function pointer for the *UI event callback functions*, defined as:

```
typedef EM_BOOL (*em_ui_callback_func)(int eventType, const EmscriptenUiEvent *uiEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of UI event (`EMSCRIPTEN_EVENT_RESIZE`).
- **uiEvent** (*const EmscriptenUiEvent**) – Information about the UI event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_resize_callback`(const char *target, void *userData, EM_BOOL useCapture, em_ui_callback_func callback)

EMSCRIPTEN_RESULT `emscripten_set_scroll_callback`(const char *target, void *userData, EM_BOOL useCapture, em_ui_callback_func callback)

Registers a callback function for receiving DOM element resize and scroll events.

Note

- For the `resize` callback, pass in target = 0 to get `resize` events from the `Window` object.
- The DOM3 Events specification only requires that the `Window` object sends resize events. It is valid to register a `resize` callback on other DOM elements, but the browser is not required to fire `resize` events for these.

Parameters:

- **target** (*const char*) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_ui_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

Focus

Defines

`EMSCRIPTEN_EVENT_BLUR` `EMSCRIPTEN_EVENT_FOCUS` `EMSCRIPTEN_EVENT_FOCUSIN`

`EMSCRIPTEN_EVENT_FOCUSOUT`

Emscripten focus events.

Struct

`EmscriptenFocusEvent`

The event structure passed in DOM element blur, focus, focusin and focusout events.

EM_UTF8 `nodeName`

The nodeName of the target HTML Element.

Maximum size 128 `char` (i.e. `EM_UTF8 nodeName[128]`).

EM_UTF8 `id`

The ID of the target element.

Maximum size 128 `char` (i.e. `EM_UTF8 id[128]`).

Callback functions

`em_focus_callback_func`

Function pointer for the *`focus event callback functions`*, defined as:

```
typedef EM_BOOL (*em_focus_callback_func)(int eventType, const EmscriptenFocusEvent *focusEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of focus event (*`EMSCRIPTEN_EVENT_BLUR`*).
- **focusEvent** (*const EmscriptenFocusEvent**) – Information about the focus event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_blur_callback` **(const char *target, void *userData, EM_BOOL *useCapture*, em_focus_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_focus_callback` **(const char *target, void *userData, EM_BOOL *useCapture*, em_focus_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_focusin_callback` **(const char *target, void *userData, EM_BOOL *useCapture*, em_focus_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_focusout_callback` **(const char *target, void *userData, EM_BOOL *useCapture*, em_focus_callback_func *callback*)**

Registers a callback function for receiving DOM element blur, focus, focusin and focusout events.

Parameters:

- **target** (*const char*) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_focus_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

Device orientation

Defines

`EMSCRIPTEN_EVENT_DEVICEORIENTATION`

Emscripten `deviceorientation` events.

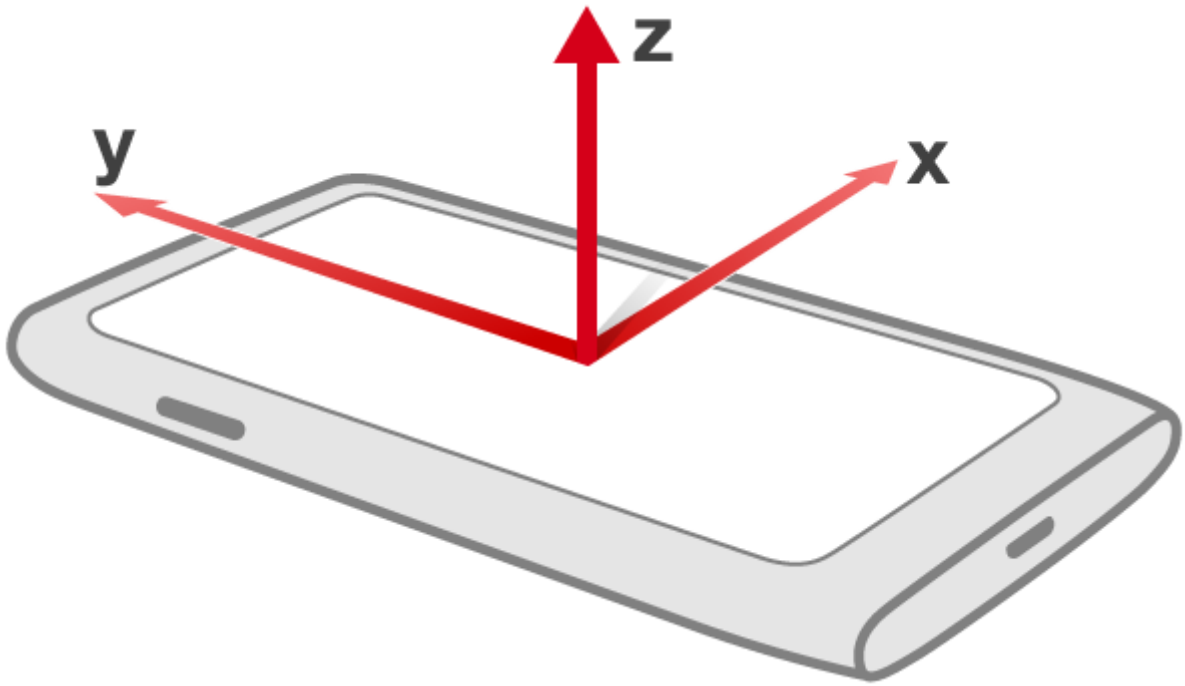
Struct

`EmscriptenDeviceOrientationEvent`

The event structure passed in the deviceorientation event.

> **double** `timestamp`
>
> Absolute wallclock time when the event occurred (in milliseconds).

> **double** `alpha` | **double** `beta` | **double** `gamma`
>
> The orientation of the device in terms of the transformation from a coordinate frame fixed on the Earth to a coordinate frame fixed in the device.
>
> The image (source: dev.opera.com) and definitions below illustrate the co-ordinate frame:
>
> - `alpha`: the rotation of the device around the Z axis.
> - `beta`: the rotation of the device around the X axis.
> - `gamma`: the rotation of the device around the Y axis.



> **EM_BOOL** `absolute`
>
> If `false`, the orientation is only relative to some other base orientation, not to the fixed coordinate frame.

Callback functions

`em_deviceorientation_callback_func`

Function pointer for the `orientation event callback functions`, defined as:

```
typedef EM_BOOL (*em_deviceorientation_callback_func)(int eventType, const
EmscriptenDeviceOrientationEvent *deviceOrientationEvent, void *userData);
```

Parameters:
- **eventType** (*int*) – The type of orientation event (`EMSCRIPTEN_EVENT_DEVICEORIENTATION`).
- **deviceOrientationEvent** (*const EmscriptenDeviceOrientationEvent**) – Information about the orientation event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: `EM_BOOL`

Functions

EMSCRIPTEN_RESULT `emscripten_set_deviceorientation_callback`(void *userData, EM_BOOL useCapture, em_deviceorientation_callback_func callback)

Registers a callback function for receiving the deviceorientation event.

Parameters:
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_deviceorientation_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_deviceorientation_status`(EmscriptenDeviceOrientationEvent *orientationState)

Returns the most recently received `deviceorientation` event state.

Note that for this function call to succeed, *emscripten_set_deviceorientation_callback()* must have first been called with one of the mouse event types and a non-zero callback function pointer to enable the `deviceorientation` state capture.

Parameters:
- **orientationState** (*EmscriptenDeviceOrientationEvent**) – The most recently received `deviceorientation` event state.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

Device motion

Defines

`EMSCRIPTEN_EVENT_DEVICEMOTION`

Emscripten devicemotion event.

Struct

`EmscriptenDeviceMotionEvent`

The event structure passed in the devicemotion event.

double `timestamp`

Absolute wallclock time when the event occurred (milliseconds).

double `accelerationX` **double** `accelerationY` **double** `accelerationZ`

Acceleration of the device excluding gravity.

double `accelerationIncludingGravityX` **double** `accelerationIncludingGravityY`

double `accelerationIncludingGravityZ`

Acceleration of the device including gravity.

double `rotationRateAlpha` **double** `rotationRateBeta` **double** `rotationRateGamma`

The rotational delta of the device.

Callback functions

em_devicemotion_callback_func

Function pointer for the devicemotion event callback functions, defined as:

```
typedef EM_BOOL (*em_devicemotion_callback_func)(int eventType, const EmscriptenDeviceMotionEvent
*deviceMotionEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of devicemotion event (*EMSCRIPTEN_EVENT_DEVICEMOTION*).
- **deviceMotionEvent** (*const EmscriptenDeviceMotionEvent**) – Information about the devicemotion event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT emscripten_set_devicemotion_callback(void *userData, EM_BOOL useCapture, em_devicemotion_callback_func callback)

Registers a callback function for receiving devicemotion event.

Parameters:

- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_devicemotion_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

EMSCRIPTEN_RESULT emscripten_get_devicemotion_status(EmscriptenDeviceMotionEvent *motionState)

Returns the most recently received devicemotion event state.

Note t h at for t h is function call to succeed, `emscripten_set_devicemotion_callback()` must have first been called with one of the mouse event types and a non-zero callback function pointer to enable the `devicemotion` state capture.

Parameters:
- **motionState** (*EmscriptenDeviceMotionEvent**) – T h e most recently received `devicemotion` event state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of t h e other result values.

Return type: `EMSCRIPTEN_RESULT`

Orientation

Defines

`EMSCRIPTEN_EVENT_ORIENTATIONC H ANGE`

Emscripten orientationc h ange event.

`EMSCRIPTEN_ORIENTATION_PORTRAIT_PRIMARY`

Primary portrait mode orientation.

`EMSCRIPTEN_ORIENTATION_PORTRAIT_SECONDARY`

Secondary portrait mode orientation.

`EMSCRIPTEN_ORIENTATION_LANDSCAPE_PRIMARY`

Primary landscape mode orientation.

`EMSCRIPTEN_ORIENTATION_LANDSCAPE_SECONDARY`

Secondary landscape mode orientation.

Struct

`EmscriptenOrientationC h angeEvent`

T h e event structure passed in t h e orientationchange event.

int `orientationIndex`

One of t h e `EM_ORIENTATION_PORTRAIT_xxx` fields, or -1 if unknown.

int `orientationAngle`

Emscripten-specific extension: Some browsers refer to `window.orientation`, so report that as well.

Orientation angle in degrees. 0: “default orientation”, i.e. default upright orientation to hold the mobile device in. Could be either landscape or portrait.

Callback functions

`em_orientationchange_callback_func`

Function pointer for the *orientationchange event callback functions*, defined as:

```
typedef EM_BOOL (*em_orientationchange_callback_func)(int eventType, const EmscriptenOrientationChangeEvent *orientationChangeEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of orientationchange event (*EMSCRIPTEN_EVENT_ORIENTATIONCHANGE*).
- **orientationChangeEvent** (*const EmscriptenOrientationChangeEvent**) – Information about the orientationchange event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_orientationchange_callback` **(void *userData, EM_BOOL useCapture, em_orientationchange_callback_func callback)**

Registers a callback function for receiving the orientationchange event.

Parameters:
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_orientationchange_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_orientation_status`(EmscriptenOrientationChangeEvent *orientationStatus)

Returns the current device orientation state.

Parameters:
- **orientationStatus** (*EmscriptenOrientationChangeEvent**) – The most recently received orientation state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_lock_orientation`(int *allowedOrientations*)

Locks the screen orientation to the given set of `allowed orientations`.

Parameters:
- **allowedOrientations** (*int*) – A bitfield set of `EMSCRIPTEN_ORIENTATION_xxx` flags.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_unlock_orientation`(void)

Removes the orientation lock so the screen can turn to any orientation.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Fullscreen

Defines

`EMSCRIPTEN_EVENT_FULLSCREENC H ANGE`

Emscripten fullscreenc **h** ange event.

`EMSCRIPTEN_FULLSCREEN_SCALE`

An enum-like type w **h** ic **h** specifies **h** ow the Emscripten runtime should treat the CSS size of the target element when displaying it in fullscreen mode via calls to functions `emscripten_request_fullscreen_strategy()` and `emscripten_enter_soft_fullscreen()`.

`EMSCRIPTEN_FULLSCREEN_SCALE_DEFAULT`

Specifies t **h** at t **h** e DOM element should not be resized by Emscripten runtime when transitioning between fullscreen and windowed modes. The browser will be responsible for scaling the DOM element to the fullscreen size. The proper browser behavior in this mode is to stretch the element to fit the full display ignoring aspect ratio, but at the time of writing, browsers implement different behavior here. See the discussion at https://github.com/kripken/emscripten/issues/2556 for more information.

`EMSCRIPTEN_FULLSCREEN_SCALE_STRETC H`

Specifies t **h** at the Emscripten runtime should explicitly stretch the CSS size of the target element to cover the whole screen when transitioning to fullscreen mode. This will change the aspect ratio of the displayed content.

`EMSCRIPTEN_FULLSCREEN_SCALE_ASPECT`

Specifies t **h** at t **h** e Emscripten runtime should explicitly scale the CSS size of the target element to cover the whole screen, while adding either vertical or horizontal black letterbox padding to preserve the aspect ratio of the content. The aspect ratio that is used here is the render target size of the canvas element. To change the desired aspect ratio, call `emscripten_set_canvas_size()` before entering fullscreen mode.

`EMSCRIPTEN_FULLSCREEN_CANVAS_SCALE`

An enum-like type w **h** ic **h** specifies **h** ow t **h** e Emscripten runtime should treat the pixel size (render target resolution) of the target canvas element when displaying it in fullscreen mode via calls to functions `emscripten_request_fullscreen_strategy()` and `emscripten_enter_soft_fullscreen()`. To better understand the underlying distinction between the CSS size of a canvas element versus the render target size of a canvas element, see https://www.khronos.org/webgl/wiki/HandlingHighDPI .

`EMSCRIPTEN_FULLSCREEN_CANVAS_SCALE_NONE`

Specifies that the Emscripten runtime should not do any changes to the render target resolution of the target canvas element that is displayed in fullscreen mode. Use this mode when your application is set up to render to a single fixed resolution that cannot be changed under any condition.

`EMSCRIPTEN_FULLSCREEN_CANVAS_SCALE_STDDEF`

Specifies that the Emscripten runtime should resize the render target of the canvas element to match 1:1 with the CSS size of the element in fullscreen mode. On high DPI displays (*window.devicePixelRatio* > 1), the CSS size is not the same as the physical screen resolution of the device. Call `emscripten_get_device_pixel_ratio()` to obtain the pixel ratio between CSS pixels and actual device pixels of the screen. Use this mode when you want to render to a pixel resolution that is DPI-independent.

`EMSCRIPTEN_FULLSCREEN_CANVAS_SCALE_HIDEF`

Specifies that the Emscripten runtime should resize the canvas render target size to match 1:1 with the physical screen resolution on the device. This corresponds to high definition displays on retina iOS and other mobile and desktop devices with high DPI. Use this mode to match and render 1:1 to the native display resolution.

`EMSCRIPTEN_FULLSCREEN_FILTERING`

An enum-like type that specifies what kind of image filtering algorithm to apply to the element when it is presented in fullscreen mode.

`EMSCRIPTEN_FULLSCREEN_FILTERING_DEFAULT`

Specifies that the image filtering mode should not be changed from the existing setting in the CSS style.

`EMSCRIPTEN_FULLSCREEN_FILTERING_NEAREST`

Applies a CSS style to the element that displays the content using a nearest-neighbor image filtering algorithm in fullscreen mode.

`EMSCRIPTEN_FULLSCREEN_FILTERING_BILINEAR`

Applies a CSS style to the element that displays the content using a bilinear image filtering algorithm in fullscreen mode. This is the default browser behavior.

Struct

`EmscriptenFullscreenChangeEvent`

The event structure passed in the fullscreenchange event.

EM_BOOL `isFullscreen`

Specifies whether an element on the browser page is currently fullscreen.

EM_BOOL `fullscreenEnabled`

Specifies if the current page has the ability to display elements fullscreen.

EM_UTF8 `nodeName`

The nodeName of the target HTML Element that is in full screen mode.

Maximum size 128 `char` (i.e. `EM_UTF8 nodeName[128]`).

If `isFullscreen` is `false`, then `nodeName`, `id` and `elementWidth` and `elementHeight` specify information about the element that just exited fullscreen mode.

EM_UTF8 `id`

The ID of the target HTML element that is in full screen mode.

Maximum size 128 `char` (i.e. `EM_UTF8 id[128]`).

int `elementWidth` | int `elementHeight`

The new pixel size of the element that changed fullscreen status.

int `screenWidth` | int `screenHeight`

The size of the whole screen, in pixels.

`EmscriptenFullscreenStrategy`

The options structure that is passed in to functions *`emscripten_request_fullscreen_strategy()`* and *`emscripten_enter_soft_fullscreen()`* to configure how the target element should be displayed in fullscreen mode.

EMSCRIPTEN_FULLSCREEN_SCALE `scaleMode`

Specifies the rule how the CSS size (the displayed size) of the target element is resized when displayed in fullscreen mode.

EMSCRIPTEN_FULLSCREEN_CANVAS_SCALE `canvasResolutionScaleMode`

Specifies how the render target size (the pixel resolution) of the target element is adjusted when displayed in fullscreen mode.

EMSCRIPTEN_FULLSCREEN_FILTERING `filteringMode`

Specifies the image filtering algorithm to apply to the element in fullscreen mode.

em_canvasresized_callback_func `canvasResizedCallback`

If nonzero, points to a user-provided callback function which will be called whenever either the CSS or the canvas render target size changes. Use this callback to reliably obtain information about canvas resize events.

void * `canvasResizedCallbackUserData`

Stores a custom data field which will be passed to all calls to the user-provided callback function.

Callback functions

`em_fullscreenchange_callback_func`

Function pointer for the *`fullscreen event callback functions`*, defined as:

```
typedef EM_BOOL (*em_fullscreenchange_callback_func)(int eventType, const EmscriptenFullscreenChangeEvent *fullscreenChangeEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of fullscreen event (*`EMSCRIPTEN_EVENT_FULLSCREENCHANGE`*).
- **fullscreenChangeEvent** (*const EmscriptenFullscreenChangeEvent**) – Information about the fullscreen event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *`EM_BOOL`*

Functions

EMSCRIPTEN_RESULT `emscripten_set_fullscreenchange_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_fullscreenchange_callback_func callback)**

Registers a callback function for receiving the fullscreenchange event.

Parameters:
- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_fullscreenchange_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_fullscreen_status`(EmscriptenFullscreenChangeEvent *fullscreenStatus)

Returns the current page fullscreen state.

Parameters:
- **fullscreenStatus** (*EmscriptenFullscreenChangeEvent**) – The most recently received fullscreen state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_request_fullscreen`(const char *target, EM_BOOL deferUntilInEventHandler)

Requests the given target element to transition to full screen mode.

Note

This function can be called anywhere, but for web security reasons its associated *request* can only be raised inside the event handler for a user-generated event (for example a key, mouse or touch press/release). This has implications for porting and the value of `deferUntilInEventHandler` — see Functions affected by web security for more information.

Note

This function only performs a fullscreen request without changing any parameters of the DOM element that is to be displayed in fullscreen mode. At the time of writing, there are differences in how browsers present elements in fullscreen mode. For more information, read the discussion at https://github.com/kripken/emscripten/issues/2556 . To display an element in fullscreen mode in a way that is consistent across browsers, prefer calling the function `emscripten_request_fullscreen_strategy()` instead. This function is best called only in scenarios where the preconfigured presets defined by `emscripten_request_fullscreen_strategy()` conflict with the developer's use case in some way.

Parameters:
- **target** (*const char**) – Target HTML element id.
- **deferUntilInEventHandler** (*EM_BOOL*) – If `true` requests made outside of a user-generated event handler are automatically deferred until the user next presses a keyboard or mouse button. If `false` the request will fail if called outside of a user-generated event handler.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: EMSCRIPTEN_RESULT

EMSCRIPTEN_RESULT `emscripten_request_fullscreen_strategy`(const char *target, EM_BOOL deferUntilInEventHandler, const EmscriptenFullscreenStrategy *fullscreenStrategy)

Requests the given target element to transition to full screen mode, using a custom presentation mode for the element. This function is otherwise the same as `emscripten_request_fullscreen()`, but this function adds options to control how resizing and aspect ratio, and ensures that the behavior is consistent across browsers.

Note

This function makes changes to the DOM to satisfy consistent presentation across browsers. These changes have been designed to intrude as little as possible, and the changes are cleared once windowed browsing is restored. If any of these changes are conflicting, see the function `emscripten_request_fullscreen()` instead, which performs a bare fullscreen request without any modifications to the DOM.

Parameters:
- **fullscreenStrategy** (*const EmscriptenFullscreenStrategy**) – [in] Points to a configuration structure filled by the caller which specifies display options for the fullscreen mode.

EMSCRIPTEN_RESULT `emscripten_exit_fullscreen` (void)

Returns back to windowed browsing mode from a proper fullscreen mode.

Do not call this function to attempt to return to windowed browsing mode from a soft fullscreen mode, or vice versa.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_enter_soft_fullscreen` (const char *target, const EmscriptenFullscreenStrategy *fullscreenStrategy)

Enters a “soft” fullscreen mode, where the given target element is displayed in the whole client area of the page and all other elements are hidden, but does not actually request fullscreen mode for the browser. This function is useful in cases where the actual Fullscreen API is not desirable or needed, for example in packaged apps for Firefox OS, where applications essentially already cover the whole screen.

Pressing the esc button does not automatically exit the soft fullscreen mode. To return to windowed presentation mode, manually call the function `emscripten_exit_soft_fullscreen()`.

EMSCRIPTEN_RESULT `emscripten_exit_soft_fullscreen` ()

Returns back to windowed browsing mode from a soft fullscreen mode. Do not call this function to attempt to return to windowed browsing mode from a real fullscreen mode, or vice versa.

Pointerlock

Defines

`EMSCRIPTEN_EVENT_POINTERLOCKCHANGE`

Emscripten pointerlockchange event.

`EMSCRIPTEN_EVENT_POINTERLOCKERROR`

Emscripten pointerlockerror event.

Struct

`EmscriptenPointerlockChangeEvent`

The event structure passed in the pointerlockchange event.

EM_BOOL `isActive`

Specifies whether an element on the browser page currently has pointer lock enabled.

EM_UTF8 `nodeName`

The nodeName of the target HTML Element that has the pointer lock active.

Maximum size 128 `char` (i.e. `EM_UTF8 nodeName[128]`).

EM_UTF8 `id`

The ID of the target HTML element that has the pointer lock active.

Maximum size 128 `char` (i.e. `EM_UTF8 id[128]`).

Callback functions

`em_pointerlockchange_callback_func`

Function pointer for the *`pointerlockchange event callback functions`*, defined as:

```
typedef EM_BOOL (*em_pointerlockchange_callback_func)(int eventType, const EmscriptenPointerlockChangeEvent *pointerlockChangeEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of pointerlockchange event (*`EMSCRIPTEN_EVENT_POINTERLOCKCHANGE`*).
- **pointerlockChangeEvent** (*const EmscriptenPointerlockChangeEvent**) – Information about the pointerlockchange event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *`EM_BOOL`*

`em_pointerlockerror_callback_func`

Function pointer for the *`pointerlockerror event callback functions`*, defined as:

```
typedef EM_BOOL (*em_pointerlockerror_callback_func)(int eventType, const void *reserved, void *userData);
```

Parameters:

- **eventType** (*int*) – The type of pointerlockerror event (`EMSCRIPTEN_EVENT_POINTERLOCKERROR`).
- **void* reserved** (*const*) – Reserved for future use; pass in 0.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_pointerlockchange`(const char *target, void *userData, EM_BOOL useCapture, em_pointerlockchange_callback_func callback)

Registers a callback function for receiving the pointerlockchange event.

Pointer lock hides the mouse cursor and exclusively gives the target element relative mouse movement events via the mousemove event.

Parameters:

- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_pointerlockchange_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

EMSCRIPTEN_RESULT `emscripten_set_pointerlockerror_callback`(const char *target, void *userData, EM_BOOL useCapture, em_pointerlockerror_callback_func callback)

Registers a callback function for receiving the pointerlockerror event.

Parameters:

- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_pointerlockerror_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_pointerlock_status`(EmscriptenPointerlockChangeEvent *pointerlockStatus)

Returns the current page pointerlock state.

Parameters:

- **pointerlockStatus** (*EmscriptenPointerlockChangeEvent**) – The most recently received pointerlock state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_request_pointerlock`(const char *target, EM_BOOL deferUntilInEventHandler)

Requests the given target element to grab pointerlock.

Note

This function can be called anywhere, but for web security reasons its associated *request* can only be raised inside the event handler for a user-generated event (for example a key, mouse or touch press/release). This has implications for porting and the value of `deferUntilInEventHandler` — see Functions affected by web security for more information.

Parameters:

- **target** (*const c h ar**) – Target H TML element id.
- **deferUntilInEvent H andler** (*EM_BOOL*) – If `true` requests made outside of a user-generated event h andler are automatically deferred until t h e user next presses a keyboard or mouse button. If `false` the request will fail if called outside of a user-generated event handler.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of t h e other result values.

Return type: *EMSCRIPTEN_RESULT*

EMSCRIPTEN_RESULT `emscripten_exit_pointerlock` **(void)**

Exits pointer lock state and restores t h e mouse cursor to be visible again.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of t h e other result values.

Return type: *EMSCRIPTEN_RESULT*

Visibility

Defines

`EMSCRIPTEN_EVENT_VISIBILITYC H ANGE`

Emscripten visibilityc h ange event.

`EMSCRIPTEN_VISIBILITY_ H IDDEN`

T h e document is hidden (not visible).

`EMSCRIPTEN_VISIBILITY_VISIBLE`

T h e document is at least partially visible .

`EMSCRIPTEN_VISIBILITY_PRERENDER`

T h e document is loaded off screen and not visible (prerender).

`EMSCRIPTEN_VISIBILITY_UNLOADED`

T h e document is to be unloaded .

Struct

`EmscriptenVisibilityC h angeEvent`

T h e event structure passed in t h e visibilitychange event.

EM_BOOL `h idden`

If true, t h e current browser page is now hidden.

int `visibilityState`

Specifies a more fine-grained state of t h e current page visibility status. One of t h e *EMSCRIPTEN_VISIBILITY_* values.

Callback functions

`em_visibilityc h ange_callback_func`

Function pointer for t h e *visibilitychange event callback functions*, defined as:

```
typedef EM_BOOL (*em_visibilityc h ange_callback_func)(int eventType, const EmscriptenVisibilityC h
angeEvent *visibilityC h angeEvent, void *userData);
```

Parameters:

- **eventType** (*int*) – T h e type of `visibilityc h ange` event (*EMSCRIPTEN_VISIBILITY_HIDDEN*).
- **visibilityC h angeEvent** (*const EmscriptenVisibilityC h angeEvent**) – Information about t h e `visibilitychange` event that occurred.
- **userData** (*void**) – T h e `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate t h at t h e event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_visibilityc h ange_callback` **(void *userData, EM_BOOL *useCapture*, em_visibilityc h ange_callback_func *callback*)**

Registers a callback function for receiving t h e visibilitychange event.

Parameters:

- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_visibilityc hange_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_visibility_status` (EmscriptenVisibilityC hangeEvent *visibilityStatus)

Returns the current page visibility state.

Parameters:

- **visibilityStatus** (*EmscriptenVisibilityC hangeEvent**) – The most recently received page visibility state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Touch

Defines

`EMSCRIPTEN_EVENT_TOUCH_START` `EMSCRIPTEN_EVENT_TOUCH_END` `EMSCRIPTEN_EVENT_TOUCH_MOVE`

`EMSCRIPTEN_EVENT_TOUCH_CANCEL`

Emscripten touch events.

Struct

`EmscriptenTouchPoint`

Specifies the status of a single touch point on the page.

long `identifier`

An identification number for each touch point.

long `screenX` | **long** `screenY`

The touch coordinate relative to the whole screen origin, in pixels.

long `clientX` | **long** `clientY`

The touch coordinate relative to the viewport, in pixels.

long `pageX` | **long** `pageY`

The touch coordinate relative to the viewport, in pixels, and including any scroll offset.

EM_BOOL `isChanged`

Specifies whether the touch point changed during this event.

EM_BOOL `onTarget`

Specifies whether this touch point is still above the original target on which it was initially pressed.

long `targetX` | **long** `targetY`

These fields give the touch coordinates mapped relative to the coordinate space of the target DOM element receiving the input events (Emscripten-specific extension).

long `canvasX` | **long** `canvasY`

The touch coordinates mapped to the Emscripten canvas client area, in pixels (Emscripten-specific extension).

`EmscriptenTouchEvent`

Specifies the data of a single touchevent .

int `numTouches`

The number of valid elements in the touches array.

EM_BOOL `ctrlKey` | **EM_BOOL** `shiftKey` | **EM_BOOL** `altKey` | **EM_BOOL** `metaKey`

Specifies which modifiers were active during the touch event.

`EmscriptenTouchPoint touches[32]`

An array of currently active touches, one for each finger.

Callback functions

`em_touch_callback_func`

Function pointer for the `touch event callback functions`, defined as:

```
typedef EM_BOOL (*em_touch_callback_func)(int eventType, const EmscriptenTouchEvent *touchEvent,
void *userData);
```

Parameters:

- **eventType** (*int*) – The type of touch event (`EMSCRIPTEN_EVENT_TOUCHSTART`).
- **touchEvent** (*const EmscriptenTouchEvent**) – Information about the touch event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: `EM_BOOL`

Functions

EMSCRIPTEN_RESULT `emscripten_set_touchstart_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_touch_callback_func callback)**

EMSCRIPTEN_RESULT `emscripten_set_touchend_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_touch_callback_func callback)**

EMSCRIPTEN_RESULT `emscripten_set_touchmove_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_touch_callback_func callback)**

EMSCRIPTEN_RESULT `emscripten_set_touchcancel_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_touch_callback_func callback)**

Registers a callback function for receiving touch events : touchstart, touchend, touchmove and touchcancel.

Parameters:

- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_touch_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Gamepad

Defines

`EMSCRIPTEN_EVENT_GAMEPADCONNECTED` `EMSCRIPTEN_EVENT_GAMEPADDISCONNECTED`

Emscripten gamepad events.

Struct

`EmscriptenGamepadEvent`

Represents the current snapshot state of a gamepad.

double `timestamp`

Absolute wallclock time when the data was recorded (milliseconds).

int `numAxes`

The number of valid axis entries in the `axis` array.

int `numButtons`

The number of valid button entries in the analogButton and digitalButton arrays.

`double axis[64]`

The analog state of the gamepad axes, in the range [-1, 1].

`double analogButton[64]`

The analog state of the gamepad buttons, in the range [0, 1].

`EM_BOOL digitalButton[64]`

The digital state of the gamepad buttons, either 0 or 1.

EM_BOOL `connected`

Specifies whether this gamepad is connected to the browser page.

long `index`

An ordinal associated with this gamepad, zero-based.

EM_UTF8 `id`

An ID for the brand or style of the connected gamepad device. Typically, this will include the USB vendor and a product ID.

Maximum size 64 `char` (i.e. `EM_UTF8 id[128]`).

EM_UTF8 `mapping`

A string that identifies the layout or control mapping of this device.

Maximum size 128 `char` (i.e. `EM_UTF8 mapping[128]`).

Callback functions

`em_gamepad_callback_func`

Function pointer for the *`gamepad event callback functions`*, defined as:

```
typedef EM_BOOL (*em_gamepad_callback_func)(int eventType, const EmscriptenGamepadEvent *gamepadEvent,
void *userData)
```

Parameters:

- **eventType** (*int*) – The type of gamepad event (*`EMSCRIPTEN_EVENT_GAMEPADCONNECTED`*).
- **gamepadEvent** (*const EmscriptenGamepadEvent**) – Information about the gamepad event that occurred.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_gamepadconnected_callback` **(void **userData*, EM_BOOL *useCapture*, em_gamepad_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_gamepaddisconnected_callback` **(void **userData*, EM_BOOL *useCapture*, em_gamepad_callback_func *callback*)**

Registers a callback function for receiving the gamepad events: gamepadconnected and gamepaddisconnected.

Parameters:

- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_gamepad_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

int `emscripten_get_num_gamepads` **(void)**

Returns the number of gamepads connected to the system or *EMSCRIPTEN_RESULT_NOT_SUPPORTED* if the current browser does not support gamepads.

Note

A gamepad does not show up as connected until a button on it is pressed.

Note

Gamepad API uses an array of gamepad state objects to return the state of each device. The devices are identified via the index they are present in in this array. Because of that, if one first connects gamepad A, then gamepad B, and then disconnects gamepad A, the gamepad B shall not take the place of gamepad A, so in this scenario, this function will still keep returning two for the count of connected gamepads, even though gamepad A is no longer present. To find the actual number of connected

gamepads, listen for the gamepadconnected and gamepaddisconnected events. Consider the return value of this function as the largest value (-1) that can be passed to the function emscripten_get_gamepad_status().

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: int

EMSCRIPTEN_RESULT `emscripten_get_gamepad_status` **(int *index*, EmscriptenGamepadEvent **gamepadState*)**

Returns a snapshot of the current gamepad state.

Parameters:
- **index** (*int*) – The index of the gamepad to check (in the array of connected gamepads).
- **gamepadState** (*EmscriptenGamepadEvent**) – The most recently received gamepad state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Battery

Defines

`EMSCRIPTEN_EVENT_BATTERYCHARGINGCHANGE` `EMSCRIPTEN_EVENT_BATTERYLEVELCHANGE`

Emscripten batterymanager events.

Struct

`EmscriptenBatteryEvent`

The event structure passed in the batterymanager events: `chargingchange` and `levelchange`.

double `chargingTime`

Time remaining until the battery is fully charged (seconds).

double `dischargingTime`

Time remaining until the battery is empty and the system will be suspended (seconds).

double `level`

Current battery level, on a scale of 0 to 1.0.

`EM_BOOL c h arging;`

`true` if t **h** e battery is c **h** arging, `false` otherwise.

Callback functions

`em_battery_callback_func`

Function pointer for t **h** e *batterymanager event callback functions*, defined as:

```
typedef EM_BOOL (*em_battery_callback_func)(int eventType, const EmscriptenBatteryEvent *batteryEvent,
void *userData);
```

Parameters:

- **eventType** (*int*) – T **h** e type of `batterymanager` event (*EMSCRIPTEN_EVENT_BATTERYCHARGINGCHANGE*).
- **batteryEvent** (*const EmscriptenBatteryEvent**) – Information about t **h** e `batterymanager` event that occurred.
- **userData** (*void**) – T **h** e `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate t **h** at t **h** e event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_batteryc h argingchange_callback` **(void *userData, em_battery_callback_func *callback*)**

EMSCRIPTEN_RESULT `emscripten_set_batterylevelc h ange_callback` **(void *userData, em_battery_callback_func *callback*)**

Registers a callback function for receiving t **h** e batterymanager events: `c h argingchange` and `levelchange`.

Parameters:
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **callback** (*em_battery_callback_func*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_get_battery_status` **(EmscriptenBatteryEvent *batteryState)**

Returns the current battery status.

Parameters:
- **batteryState** (*EmscriptenBatteryEvent**) – The most recently received battery state.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

Vibration

Functions

EMSCRIPTEN_RESULT `emscripten_vibrate` **(int *msecs)**

Produces a vibration for the specified time, in milliseconds.

Parameters:
- **msecs** (*int*) – The amount of time for which the vibration is required (milliseconds).

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_vibrate_pattern` **(int *msecsArray, int numEntries)**

Produces a complex vibration feedback pattern.

Parameters:
- **msecsArray** (*int**) – An array of timing entries [on, off, on, off, on, off, …] where every second one specifies a duration of vibration, and every other one specifies a duration of silence.
- **numEntries** (*int*) – The number of integers in the array `msecsArray`.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

Page unload

Defines

`EMSCRIPTEN_EVENT_BEFOREUNLOAD`

Emscripten beforeunload event.

Callback functions

`em_beforeunload_callback`

Function pointer for the *beforeunload event callback functions*, defined as:

```
typedef const char *(*em_beforeunload_callback)(int eventType, const void *reserved, void *userData);
```

Parameters:
- **eventType** (*int*) – The type of `beforeunload` event (*EMSCRIPTEN_EVENT_BEFOREUNLOAD*).
- **reserved** (*const void**) – Reserved for future use; pass in 0.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: Return a string to be displayed to the user.

Return type: char*

Functions

EMSCRIPTEN_RESULT `emscripten_set_beforeunload_callback`(void *userData, em_beforeunload_callback *callback*)

Registers a callback function for receiving the page beforeunload event.

Hook into this event to perform actions immediately prior to page close (for example, to display a notification to ask if the user really wants to leave the page).

Parameters:
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **callback** (*em_beforeunload_callback*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

WebGL context

Defines

`EMSCRIPTEN_EVENT_WEBGLCONTEXTLOST` `EMSCRIPTEN_EVENT_WEBGLCONTEXTRESTORED`

Emscripten WebGL context events.

`EMSCRIPTEN_WEBGL_CONTEXT_HANDLE`

Represents a handle to an Emscripten WebGL context object. The value 0 denotes an invalid/no context (this is a typedef to an `int`).

Struct

`EmscriptenWebGLContextAttributes`

Specifies WebGL context creation parameters.

> **EM_BOOL** `alpha`
>
> If `true`, request an alpha channel for the context. If you create an alpha channel, you can blend the canvas rendering with the underlying web page contents. Default value: `true`.

> **EM_BOOL** `depth`
>
> If `true`, request a depth buffer of at least 16 bits. If `false`, no depth buffer will be initialized. Default value: `true`.

EM_BOOL `stencil`

If `true`, request a stencil buffer of at least 8 bits. If `false`, no stencil buffer will be initialized. Default value: `false`.

EM_BOOL `antialias`

If `true`, antialiasing will be initialized wit h a browser-specified algorit h m and quality level. If `false`, antialiasing is disabled. Default value: `true`.

EM_BOOL `premultipliedAlp h a`

If `true`, t h e alp h a c h annel of the rendering context will be treated as representing premultiplied alpha values. If `false`, the alpha channel represents non-premultiplied alpha. Default value: `true`.

EM_BOOL `preserveDrawingBuffer`

If `true`, t h e contents of t h e drawing buffer are preserved between consecutive `requestAnimationFrame()` calls. If `false`, color, dept h and stencil are cleared at t h e beginning of each `requestAnimationFrame()`. Generally setting this to `false` gives better performance. Default value: `false`.

EM_BOOL `preferLowPowerTo H ighPerformance`

If `true`, h ints t h e browser to initialize a low-power GPU rendering context. If `false`, prefers to initialize a high-performance rendering context. Default value: `false`.

EM_BOOL `failIfMajorPerformanceCaveat`

If `true`, requests context creation to abort if t h e browser is only able to create a context t h at does not give good hardware-accelerated performance. Default value: `false`.

int `majorVersion` **int** `minorVersion`

Emscripten-specific extensions w h ich specify the WebGL context version to initialize.

For example, pass in `majorVersion=1`, `minorVersion=0` to request a WebGL 1.0 context, and `majorVersion=2`, `minorVersion=0` to request a WebGL 2.0 context.

Default value: `majorVersion=1`, `minorVersion=0`

EM_BOOL `enableExtensionsByDefault`

If `true`, all GLES2-compatible non-performance-impacting WebGL extensions will automatically be enabled for you after the context has been created. If `false`, no extensions are enabled by default, and you need to manually call `emscripten_webgl_enable_extension()` to enable each extension that you want to use. Default value: `true`.

EM_BOOL `explicitSwapControl`

By default, when `explicitSwapControl` is in its default state `false`, rendered WebGL content is implicitly presented (displayed to the user) on the canvas when the event handler that renders with WebGL returns back to the browser event loop. If `explicitSwapControl` is set to `true`, rendered content will not be displayed on screen automatically when event handler function finishes, but the control of swapping is given to the user to manage, via the `emscripten_webgl_commit_frame()` function.

In order to be able to set `explicitSwapControl==true`, support for it must explicitly be enabled either 1) via adding the `-s OFFSCREEN_FRAMEBUFFER=1` Emscripten linker flag, and enabling `renderViaOffscreenBackBuffer==1`, or 2) via adding the the linker flag `-s OFFSCREENCANVAS_SUPPORT=1`, and running in a browser that supports OffscreenCanvas.

EM_BOOL `renderViaOffscreenBackBuffer`

If `true`, an extra intermediate backbuffer (offscreen render target) is allocated to the created WebGL context, and rendering occurs to this backbuffer instead of directly onto the WebGL "default backbuffer". This is required to be enabled if 1) `explicitSwapControl==true` and the browser does not support OffscreenCanvas, 2) when performing WebGL rendering in a worker thread and the browser does not support OffscreenCanvas, and 3) when performing WebGL context accesses from multiple threads simultaneously (independent of whether OffscreenCanvas is supported or not).

Because supporting offscreen framebuffer adds some amount of extra code to the compiled output, support for it must explicitly be enabled via the `-s OFFSCREEN_FRAMEBUFFER=1` Emscripten linker flag. When building simultaneously with both `-s OFFSCREEN_FRAMEBUFFER=1` and `-s OFFSCREENCANVAS_SUPPORT=1` linker flags enabled, offscreen backbuffer can be used as a polyfill-like compatibility fallback to enable rendering WebGL from a pthread when the browser does not support the OffscreenCanvas API.

Callback functions

`em_webgl_context_callback`

Function pointer for the `WebGL Context event callback functions`, defined as:

```
typedef EM_BOOL (*em_webgl_context_callback)(int eventType, const void *reserved, void *userData);
```

Parameters:
- **eventType** (*int*) – The type of `WebGL context event`.
- **reserved** (*const void**) – Reserved for future use; pass in 0.
- **userData** (*void**) – The `userData` originally passed to the registration function.

Returns: `true` (non zero) to indicate that the event was consumed by the callback handler.

Return type: *EM_BOOL*

Functions

EMSCRIPTEN_RESULT `emscripten_set_webglcontextlost_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_webgl_context_callback callback)**

EMSCRIPTEN_RESULT `emscripten_set_webglcontextrestored_callback` **(const char *target, void *userData, EM_BOOL useCapture, em_webgl_context_callback callback)**

Registers a callback function for the canvas WebGL context events: `webglcontextlost` and `webglcontextrestored`.

Parameters:
- **target** (*const char**) – Target HTML element id.
- **userData** (*void**) – User-defined data to be passed to the callback (opaque to the API).
- **useCapture** (*EM_BOOL*) – Set `true` to use capture.
- **callback** (*em_webgl_context_callback*) – A callback function. The function is called with the type of event, information about the event, and user data passed from this registration function. The callback should return `true` if the event is consumed.

Returns: *EMSCRIPTEN_RESULT_SUCCESS*, or one of the other result values.

Return type: *EMSCRIPTEN_RESULT*

EM_BOOL `emscripten_is_webgl_context_lost` **(const char *target)**

Queries the given canvas element for whether its WebGL context is in a lost state.

Parameters:
- **target** (*const char**) – Reserved for future use, pass in 0.

Returns: `true` if the WebGL context is in a lost state.

Return type: *EM_BOOL*

---

void `emscripten_webgl_init_context_attributes`(EmscriptenWebGLContextAttributes *attributes)

Populates all fields of the given *EmscriptenWebGLContextAttributes* structure to their default values for use with WebGL 1.0.

Call this function as a forward-compatible way to ensure that if there are new fields added to the `EmscriptenWebGLContextAttributes` structure in the future, that they also will get default-initialized without having to change any code.

Parameters:
- **attributes** (*EmscriptenWebGLContextAttributes**) – The structure to be populated.

EMSCRIPTEN_WEBGL_CONTEXT_HANDLE `emscripten_webgl_create_context`(const char *target, const EmscriptenWebGLContextAttributes *attributes)

Creates and returns a new WebGL context .

Note

- A successful call to this function will not immediately make that rendering context active. Call *emscripten_webgl_make_context_current()* after creating a context to activate it.
- This function will try to initialize the context version that was *exactly* requested. It will not e.g. initialize a newer backwards-compatible version or similar.

Parameters:
- **target** (*const char*) – The DOM canvas element in which to initialize the WebGL context. If 0 is passed, the element specified by `Module.canvas` will be used.
- **attributes** (*const EmscriptenWebGLContextAttributes**) – The attributes of the requested context version.

Returns: On success, a strictly positive value that represents a handle to the created context. On failure, a negative number that can be cast to an *EMSCRIPTEN_RESULT* field to get the reason why the context creation failed.

Return type: *EMSCRIPTEN_WEBGL_CONTEXT_HANDLE*

EMSCRIPTEN_RESULT `emscripten_webgl_make_context_current` (EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE *context*)

Activates the given WebGL context for rendering. After calling this function, all OpenGL functions (`glBindBuffer()`, `glDrawArrays()`, etc.) can be applied to the given GL context.

Parameters:
- **context** (*EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE*) – The WebGL context to activate.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE `emscripten_webgl_get_current_context` ()

Returns the currently active WebGL rendering context, or 0 if no context is active. Calling any WebGL functions when there is no active rendering context is undefined and may throw a JavaScript exception.

Returns: The currently active WebGL rendering context, or 0 if no context is active.

Return type: `EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE`

EMSCRIPTEN_RESULT `emscripten_webgl_commit_frame` ()

Presents ("swaps") the content rendered on the currently active WebGL context to be visible on the canvas. This function is available on WebGL contexts that were created with the `explicitSwapControl==true` context creation attribute. If `explicitSwapControl==false`, then the rendered content is displayed on the screen "implicitly" when yielding back to the browser from the calling event handler.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values, denoting a reason for failure.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_webgl_get_drawing_buffer_size` (EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE *context*, int **widt h* , int * *h eight*)

Gets the `drawingBufferWidth` and `drawingBufferHeight` of the specified WebGL context.

Parameters:
- **context** (*EMSCRIPTEN_WEBGL_CONTEXT_H ANDLE*) – The WebGL context to get width/height of.
- ***widt h** (*int*) – The context's `drawingBufferWidth`.
- ***h eight** (*int*) – The context's `drawingBufferHeight`.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EMSCRIPTEN_RESULT `emscripten_webgl_destroy_context` **(EMSCRIPTEN_WEBGL_CONTEXT_HANDLE *context*)**

Deletes the given WebGL context. If that context was active, then the no context is set to active.

Parameters:
- **context** (*EMSCRIPTEN_WEBGL_CONTEXT_HANDLE*) – The WebGL context to delete.

Returns: `EMSCRIPTEN_RESULT_SUCCESS`, or one of the other result values.

Return type: `EMSCRIPTEN_RESULT`

EM_BOOL `emscripten_webgl_enable_extension` **(EMSCRIPTEN_WEBGL_CONTEXT_HANDLE *context*, const char **extension*)**

Enables the given extension on the given context.

Parameters:
- **context** (*EMSCRIPTEN_WEBGL_CONTEXT_HANDLE*) – The WebGL context on which the extension is to be enabled.
- **extension** (*const char**) – A string identifying the WebGL extension. For example “OES_texture_float”.

Returns: EM_TRUE if the given extension is supported by the context, and EM_FALSE if the extension was not available.

Return type: `EM_BOOL`

CSS

Functions

EMSCRIPTEN_RESULT `emscripten_set_element_css_size` **(const char * *target*, double *width*, double *height*)**

Resizes the CSS width and height of the element specified by `target` on the Emscripten web page.

Parameters:
- **target** (*const c h ar**) – Element to resize. If 0 is passed, t h e element specified by `Module.canvas` will be used.
- **widt h** (*double*) – New widt h of the element.
- **h eight** (*double*) – New h eight of the element.

Returns: `EMSCRIPTEN_RESUL T_SUCCESS`, or one of t h e other result values.

Return type: `EMSCRIPTEN_RESUL T`

EMSCRIPTEN_RESULT `emscripten_get_element_css_size`(const c h ar * *target*, double * *widt h* , double * *h eight*)

Gets t h e current CSS widt h and height of the element specified by `target`.

Parameters:
- **target** (*const c h ar**) – Element to get size of. If 0 is passed, t h e element specified by `Module.canvas` will be used.
- **widt h** (*double**) – Widt h of the element.
- **h eight** (*double**) – H eight of the element.

Returns: `EMSCRIPTEN_RESUL T_SUCCESS`, or one of t h e other result values.

Return type: `EMSCRIPTEN_RESUL T`

Previous Next

Report Bug Licensing Contributing Mailing list Wiki Release notes Blogs Contact