```
function createUnityInstance(canvas, config, onProgress) {
  onProgress = onProgress || function () {};


  function showBanner(msg, type) {
    // Only ever show one error at most - other banner messages after that should get
ignored
    // to avoid noise.
    if (!showBanner.aborted && config.showBanner) {
      if (type == 'error') showBanner.aborted = true;
      return config.showBanner(msg, type);
    }

    // Fallback to console logging if visible banners have been suppressed
    // from the main page.
    switch(type) {
      case 'error': console.error(msg); break;
      case 'warning': console.warn(msg); break;
      default: console.log(msg); break;
    }
  }

  function errorListener(e) {
    var error = e.reason || e.error;
    var message = error ? error.toString() : (e.message || e.reason || '');
    var stack = (error && error.stack) ? error.stack.toString() : '';

    // Do not repeat the error message if it's present in the stack trace.
    if (stack.startsWith(message)) {
      stack = stack.substring(message.length);
    }

    message += '\n' + stack.trim();

    if (!message || !Module.stackTraceRegExp || !Module.stackTraceRegExp.test(message))
      return;

    var filename = e.filename || (error && (error.fileName || error.sourceURL)) || '';
    var lineno = e.lineno || (error && (error.lineNumber || error.line)) || 0;

    errorHandler(message, filename, lineno);
  }

  function fallbackToDefaultConfigWithWarning(config, key, defaultValue) {
    var value = config[key];

    if (typeof value === "undefined" || !value) {
      console.warn("Config option \"" + key + "\" is missing or empty. Falling back to
default value: \"" + defaultValue + "\". Consider updating your WebGL template to
include the missing config option.");
      config[key] = defaultValue;
    }
  }

  var Module = {
    canvas: canvas,
    webglContextAttributes: {
      preserveDrawingBuffer: false,
      powerPreference: 2,
    },
    cacheControl: function (url) {
      return (url == Module.dataUrl || url.match(/\.bundle/)) ? "must-revalidate" :
"no-store";
    },
    streamingAssetsUrl: "StreamingAssets",
    downloadProgress: {},
    deinitializers: [],
    intervals: {},
    setInterval: function (func, ms) {
      var id = window.setInterval(func, ms);
      this.intervals[id] = true;
      return id;
    },
    clearInterval: function(id) {
      delete this.intervals[id];
      window.clearInterval(id);
    },
    preRun: [],
    postRun: [],
    print: function (message) {
      console.log(message);
    },
    printErr: function (message) {
      console.error(message);

      if (typeof message === 'string' && message.indexOf('wasm streaming compile
failed') != -1) {
        if (message.toLowerCase().indexOf('mime') != -1) {
          showBanner('HTTP Response Header "Content-Type" configured incorrectly on the
server for file ' + Module.codeUrl + ' , should be "application/wasm". Startup time
performance will suffer.', 'warning');
        } else {
          showBanner('WebAssembly streaming compilation failed! This can happen for
example if "Content-Encoding" HTTP header is incorrectly enabled on the server for file
' + Module.codeUrl + ', but the file is not pre-compressed on disk (or vice versa).
Check the Network tab in browser Devtools to debug server header configuration.',
'warning');
        }
      }
    },
    locateFile: function (url) {
      if (url == "build.wasm") return this.codeUrl;
      return url;
    },
    disabledCanvasEvents: [
      "contextmenu",
      "dragstart",
    ],
  };
  // Add fallback values for companyName, productName and productVersion to ensure that
the UnityCache is working.
  fallbackToDefaultConfigWithWarning(config, "companyName", "Unity");
  fallbackToDefaultConfigWithWarning(config, "productName", "WebGL Player");
  fallbackToDefaultConfigWithWarning(config, "productVersion", "1.0");

  for (var parameter in config)
    Module[parameter] = config[parameter];

  Module.streamingAssetsUrl = new URL(Module.streamingAssetsUrl, document.URL).href;

  // Operate on a clone of Module.disabledCanvasEvents field so that at Quit time
  // we will ensure we'll remove the events that we created (in case user has
  // modified/cleared Module.disabledCanvasEvents in between)
  var disabledCanvasEvents = Module.disabledCanvasEvents.slice();

  function preventDefault(e) {
    e.preventDefault();
  }

  disabledCanvasEvents.forEach(function (disabledCanvasEvent) {
    canvas.addEventListener(disabledCanvasEvent, preventDefault);
  });

  window.addEventListener("error", errorListener);
  window.addEventListener("unhandledrejection", errorListener);

  // Safari does not automatically stretch the fullscreen element to fill the screen.
  // The CSS width/height of the canvas causes it to remain the same size in the full
screen
  // window on Safari, resulting in it being a small canvas with black borders filling
the
  // rest of the screen.
  var _savedElementWidth = "";
  var _savedElementHeight = "";
  function webkitFullscreenChangeEventHandler(e) {
    // Safari uses webkitCurrentFullScreenElement and not fullscreenElement.
    var fullscreenElement = document.webkitCurrentFullScreenElement;
    if (fullscreenElement === canvas) {
      if (canvas.style.width) {
        _savedElementWidth = canvas.style.width;
        _savedElementHeight = canvas.style.height;
        canvas.style.width = "100%";
        canvas.style.height = "100%";
      }
    } else {
      if (_savedElementWidth) {
        canvas.style.width = _savedElementWidth;
        canvas.style.height = _savedElementHeight;
        _savedElementWidth = "";
        _savedElementHeight = "";
      }
    }
  }
  // Safari uses webkitfullscreenchange event and not fullscreenchange
  document.addEventListener("webkitfullscreenchange",
webkitFullscreenChangeEventHandler);

  // Clear the event handlers we added above when the app quits, so that the event
handler
  // functions will not hold references to this JS function scope after
  // exit, to allow JS garbage collection to take place.
  Module.deinitializers.push(function() {
    Module['disableAccessToMediaDevices']();
    disabledCanvasEvents.forEach(function (disabledCanvasEvent) {
      canvas.removeEventListener(disabledCanvasEvent, preventDefault);
    });
    window.removeEventListener("error", errorListener);
    window.removeEventListener("unhandledrejection", errorListener);

    document.removeEventListener("webkitfullscreenchange",
webkitFullscreenChangeEventHandler);

    for (var id in Module.intervals)
    {
      window.clearInterval(id);
    }
    Module.intervals = {};
  });

  Module.QuitCleanup = function () {
    for (var i = 0; i < Module.deinitializers.length; i++) {
      Module.deinitializers[i]();
    }
    Module.deinitializers = [];
    // After all deinitializer callbacks are called, notify user code that the Unity
game instance has now shut down.
    if (typeof Module.onQuit == "function")
      Module.onQuit();

  };

  var unityInstance = {
    Module: Module,
    SetFullscreen: function () {
      if (Module.SetFullscreen)
        return Module.SetFullscreen.apply(Module, arguments);
      Module.print("Failed to set Fullscreen mode: Player not loaded yet.");
    },
    SendMessage: function () {
      if (Module.SendMessage)
        return Module.SendMessage.apply(Module, arguments);
      Module.print("Failed to execute SendMessage: Player not loaded yet.");
    },
    Quit: function () {
      return new Promise(function (resolve, reject) {
        Module.shouldQuit = true;
        Module.onQuit = resolve;
      });
    },
    GetMemoryInfo: function () {
      var memInfoPtr = Module._getMemInfo();
      return {
        totalWASMHeapSize: Module.HEAPU32[memInfoPtr >> 2],
        usedWASMHeapSize: Module.HEAPU32[(memInfoPtr >> 2) + 1],
        totalJSHeapSize: Module.HEAPF64[(memInfoPtr >> 3) + 1],
        usedJSHeapSize: Module.HEAPF64[(memInfoPtr >> 3) + 2]
      };
    },
  };


  Module.SystemInfo = (function () {

    var browser, browserVersion, os, osVersion, canvas, gpu;

    var ua = navigator.userAgent + ' ';
    var browsers = [
      ['Firefox', 'Firefox'],
      ['OPR', 'Opera'],
      ['Edg', 'Edge'],
      ['SamsungBrowser', 'Samsung Browser'],
      ['Trident', 'Internet Explorer'],
      ['MSIE', 'Internet Explorer'],
      ['Chrome', 'Chrome'],
      ['CriOS', 'Chrome on iOS Safari'],
      ['FxiOS', 'Firefox on iOS Safari'],
      ['Safari', 'Safari'],
    ];

    function extractRe(re, str, idx) {
      re = RegExp(re, 'i').exec(str);
      return re && re[idx];
    }
    for(var b = 0; b < browsers.length; ++b) {
      browserVersion = extractRe(browsers[b][0] + '[\/ ](.*?)[ \\)]', ua, 1);
      if (browserVersion) {
        browser = browsers[b][1];
        break;
      }
    }
    if (browser == 'Safari') browserVersion = extractRe('Version\/(.*?) ', ua, 1);
    if (browser == 'Internet Explorer') browserVersion = extractRe('rv:(.*?)\\)? ', ua,
1) || browserVersion;

    // These OS strings need to match the ones in
Runtime/Misc/SystemInfo.cpp::GetOperatingSystemFamily()
    var oses = [
      ['Windows (.*?)[;\)]', 'Windows'],
      ['Android ([0-9_\.]+)', 'Android'],
      ['iPhone OS ([0-9_\.]+)', 'iPhoneOS'],
      ['iPad.*? OS ([0-9_\.]+)', 'iPadOS'],
      ['FreeBSD( )', 'FreeBSD'],
      ['OpenBSD( )', 'OpenBSD'],
      ['Linux|X11()', 'Linux'],
      ['Mac OS X ([0-9_\\.]+)', 'MacOS'],
      ['bot|google|baidu|bing|msn|teoma|slurp|yandex', 'Search Bot']
    ];
    for(var o = 0; o < oses.length; ++o) {
      osVersion = extractRe(oses[o][0], ua, 1);
      if (osVersion) {
        os = oses[o][1];
        osVersion = osVersion.replace(/_/g, '.');
        break;
      }
    }
    var versionMappings = {
      'NT 5.0': '2000',
      'NT 5.1': 'XP',
      'NT 5.2': 'Server 2003',
      'NT 6.0': 'Vista',
      'NT 6.1': '7',
      'NT 6.2': '8',
      'NT 6.3': '8.1',
      'NT 10.0': '10'
    };
    osVersion = versionMappings[osVersion] || osVersion;

    // TODO: Add mobile device identifier, e.g. SM-G960U

    canvas = document.createElement("canvas");
    if (canvas) {
      gl = canvas.getContext("webgl2");
      glVersion = gl ? 2 : 0;
      if (!gl) {
        if (gl = canvas && canvas.getContext("webgl")) glVersion = 1;
      }

      if (gl) {
        gpu = (gl.getExtension("WEBGL_debug_renderer_info") && gl.getParameter(0x9246
/*debugRendererInfo.UNMASKED_RENDERER_WEBGL*/)) || gl.getParameter(0x1F01
/*gl.RENDERER*/);
      }
    }

    var hasThreads = typeof SharedArrayBuffer !== 'undefined';
    var hasWasm = typeof WebAssembly === "object" && typeof WebAssembly.compile ===
"function";
    return {
      width: screen.width,
      height: screen.height,
      userAgent: ua.trim(),
      browser: browser || 'Unknown browser',
      browserVersion: browserVersion || 'Unknown version',
      mobile: /Mobile|Android|iP(ad|hone)/.test(navigator.appVersion),
      os: os || 'Unknown OS',
      osVersion: osVersion || 'Unknown OS Version',
      gpu: gpu || 'Unknown GPU',
      language: navigator.userLanguage || navigator.language,
      hasWebGL: glVersion,
      hasCursorLock: !!document.body.requestPointerLock,
      hasFullscreen: !!document.body.requestFullscreen ||
!!document.body.webkitRequestFullscreen, // Safari still uses the webkit prefixed
version
      hasThreads: hasThreads,
      hasWasm: hasWasm,
      // This should be updated when we re-enable wasm threads. Previously it checked
for WASM thread
      // support with: var wasmMemory = hasWasm && hasThreads && new
WebAssembly.Memory({"initial": 1, "maximum": 1, "shared": true});
      // which caused Chrome to have a warning that SharedArrayBuffer requires cross
origin isolation.
      hasWasmThreads: false,
    };
  })();

  function errorHandler(message, filename, lineno) {
    // Unity needs to rely on Emscripten deferred fullscreen requests, so these will
make their way to error handler
    if (message.indexOf('fullscreen error') != -1)
      return;

    if (Module.startupErrorHandler) {
      Module.startupErrorHandler(message, filename, lineno);
      return;
    }
    if (Module.errorHandler && Module.errorHandler(message, filename, lineno))
      return;
    console.log("Invoking error handler due to\n" + message);

    // Support Firefox window.dump functionality.
    if (typeof dump == "function")
      dump("Invoking error handler due to\n" + message);

    if (errorHandler.didShowErrorMessage)
      return;
    var message = "An error occurred running the Unity content on this page. See your
browser JavaScript console for more info. The error was:\n" + message;
    if (message.indexOf("DISABLE_EXCEPTION_CATCHING") != -1) {
      message = "An exception has occurred, but exception handling has been disabled in
this build. If you are the developer of this content, enable exceptions in your project
WebGL player settings to be able to catch the exception or see the stack trace.";
    } else if (message.indexOf("Cannot enlarge memory arrays") != -1) {
      message = "Out of memory. If you are the developer of this content, try
allocating more memory to your WebGL build in the WebGL player settings.";
    } else if (message.indexOf("Invalid array buffer length") != -1  ||
message.indexOf("Invalid typed array length") != -1 || message.indexOf("out of memory")
!= -1 || message.indexOf("could not allocate memory") != -1) {
      message = "The browser could not allocate enough memory for the WebGL content. If
you are the developer of this content, try allocating less memory to your WebGL build
in the WebGL player settings.";
    }
    alert(message);
    errorHandler.didShowErrorMessage = true;
  }
  Module.abortHandler = function (message) {
    errorHandler(message, "", 0);
    return true;
  };

  Error.stackTraceLimit = Math.max(Error.stackTraceLimit || 0, 50);

  function progressUpdate(id, e) {
    if (id == "symbolsUrl")
      return;
    var progress = Module.downloadProgress[id];
    if (!progress)
      progress = Module.downloadProgress[id] = {
        started: false,
        finished: false,
        lengthComputable: false,
        total: 0,
        loaded: 0,
      };
    if (typeof e == "object" && (e.type == "progress" || e.type == "load")) {
      if (!progress.started) {
        progress.started = true;
        progress.lengthComputable = e.lengthComputable;
      }
      progress.total = e.total;
      progress.loaded = e.loaded;
      if (e.type == "load")
        progress.finished = true;
    }
    var loaded = 0, total = 0, started = 0, computable = 0, unfinishedNonComputable =
0;
    for (var id in Module.downloadProgress) {
      var progress = Module.downloadProgress[id];
      if (!progress.started)
        return 0;
      started++;
      if (progress.lengthComputable) {
        loaded += progress.loaded;
        total += progress.total;
        computable++;
      } else if (!progress.finished) {
        unfinishedNonComputable++;
      }
    }
    var totalProgress = started ? (started - unfinishedNonComputable - (total ?
computable * (total - loaded) / total : 0)) / started : 0;
    onProgress(0.9 * totalProgress);
  }

Module.readBodyWithProgress = function() {
  /**
   * Estimate length of uncompressed content by taking average compression ratios
   * of compression type into account.
   * @param {Response} response A Fetch API response object
   * @param {boolean} lengthComputable Return wether content length was given in
header.
   * @returns {number}
   */
  function estimateContentLength(response, lengthComputable) {
    if (!lengthComputable) {
      // No content length available
      return 0;
    }

    var compression = response.headers.get("Content-Encoding");
    var contentLength = parseInt(response.headers.get("Content-Length"));

    switch (compression) {
    case "br":
      return Math.round(contentLength * 5);
    case "gzip":
      return Math.round(contentLength * 4);
    default:
      return contentLength;
    }
  }

  function readBodyWithProgress(response, onProgress, enableStreaming) {
    var reader = response.body ? response.body.getReader() : undefined;
    var lengthComputable = typeof response.headers.get('Content-Length') !==
"undefined";
    var estimatedContentLength = estimateContentLength(response, lengthComputable);
    var body = new Uint8Array(estimatedContentLength);
    var trailingChunks = [];
    var receivedLength = 0;
    var trailingChunksStart = 0;

    if (!lengthComputable) {
      console.warn("[UnityCache] Response is served without Content-Length header.
Please reconfigure server to include valid Content-Length for better download
performance.");
    }

    function readBody() {
      if (typeof reader === "undefined") {
        // Browser does not support streaming reader API
        // Fallback to Respone.arrayBuffer()
        return response.arrayBuffer().then(function (buffer) {
          var body = new Uint8Array(buffer);
          onProgress({
            type: "progress",
            response: response,
            total: buffer.length,
            loaded: 0,
            lengthComputable: lengthComputable,
            chunk: enableStreaming ? body : null
          });

          return body;
        });
      }

      // Start reading memory chunks
      return reader.read().then(function (result) {
        if (result.done) {
          return concatenateTrailingChunks();
        }

        if ((receivedLength + result.value.length) <= body.length) {
          // Directly append chunk to body if enough memory was allocated
          body.set(result.value, receivedLength);
          trailingChunksStart = receivedLength + result.value.length;
        } else {
          // Store additional chunks in array to append later
          trailingChunks.push(result.value);
        }

        receivedLength += result.value.length;
        onProgress({
          type: "progress",
          response: response,
          total: Math.max(estimatedContentLength, receivedLength),
          loaded: receivedLength,
          lengthComputable: lengthComputable,
          chunk: enableStreaming ? result.value : null
        });

        return readBody();
      });
    }

    function concatenateTrailingChunks() {
      if (receivedLength === estimatedContentLength) {
        return body;
      }

      if (receivedLength < estimatedContentLength) {
        // Less data received than estimated, shrink body
        return body.slice(0, receivedLength);
      }

      // More data received than estimated, create new larger body to prepend all
additional chunks to the body
      var newBody = new Uint8Array(receivedLength);
      newBody.set(body, 0);
      var position = trailingChunksStart;
      for (var i = 0; i < trailingChunks.length; ++i) {
        newBody.set(trailingChunks[i], position);
        position += trailingChunks[i].length;
      }

      return newBody;
    }
    return readBody().then(function (parsedBody) {
      onProgress({
        type: "load",
        response: response,
        total: parsedBody.length,
        loaded: parsedBody.length,
        lengthComputable: lengthComputable,
        chunk: null
      });

      response.parsedBody = parsedBody;
      return response;
    });
  }

  return readBodyWithProgress;
}();

Module.fetchWithProgress = function () {
  function fetchWithProgress(resource, init) {
    var onProgress = function () { };
    if (init && init.onProgress) {
      onProgress = init.onProgress;
    }

    return fetch(resource, init).then(function (response) {
      return Module.readBodyWithProgress(response, onProgress,
init.enableStreamingDownload);
    });
  }

  return fetchWithProgress;
}();

  /**
 * @interface RequestMetaData
 * An object with meta data for a request
 *
 * @property {string} url The url of a request
 * @property {string} company The company name
 * @property {string} product The product name
 * @property {number} version The version of the build
 * @property {number} size The company of the build
 * @property {number} accessedAt Timestamp when request was last accessed (Unix
timestamp format)
 * @property {number} updatedAt Timestamp when request was last updated in the cache
(Unix timestamp format)
 */

/**
 * @interface ResponseWithMetaData
 * An object with a cached response and meta data
 * @property {Response} response
 * @property {RequestMetaData} metaData
 */
```

```
Module.UnityCache = function () {
  var UnityCacheDatabase = { name: "UnityCache", version: 4 };
  var RequestMetaDataStore = { name: "RequestMetaDataStore", version: 1 };
  var RequestStore = { name: "RequestStore", version: 1 };
  var WebAssemblyStore = { name: "WebAssembly", version: 1 };
  var indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB ||
window.msIndexedDB;

  function log(message) {
    console.log("[UnityCache] " + message);
  }

  /**
   * A request cache that uses the browser Index DB to cache large requests
   * @property {Promise<void>} isConnected
   * @property {Cache} cache
   */
  function UnityCache() {
    var self = this;

    this.isConnected = this.connect().then(function () {
      return self.cleanUpCache();
    });

    this.isConnected.catch(function (error) {
      log("Error when initializing cache: " + error);
    });
  }

  var instance = null;
  /**
   * Singleton accessor. Returns unity cache instance
   * @returns {UnityCache}
   */
  UnityCache.getInstance = function () {
    if (!instance) {
      instance = new UnityCache();
    }

    return instance;
  }

  /**
   * Destroy unity cache instance. Returns a promise that waits for the
   * database connection to be closed.
   * @returns {Promise<void>}
   */
  UnityCache.destroyInstance = function () {
    if (!instance) {
      return Promise.resolve();
    }

    return instance.close().then(function () {
      instance = null;
    });
  }

  /**
   * Clear the unity cache.
   * @returns {Promise<void>} A promise that resolves when the cache is cleared.
   */
  UnityCache.prototype.clearCache = function () {
    var self = this;

    function deleteCacheEntries(cacheKeys) {
      if (cacheKeys.length === 0) {
        return Promise.resolve();
      }

      var key = cacheKeys.pop();

      return self.cache.delete(key).then(function () {
        return deleteCacheEntries(cacheKeys);
      });
    }

    return this.isConnected.then(function () {
      return self.execute(RequestMetaDataStore.name, "clear", []);
    }).then(function () {
      return self.cache.keys();
    }).then(function (keys) {
      return deleteCacheEntries(keys)
    });
  }

  /**
   * Config for request meta data store
   */
  UnityCache.UnityCacheDatabase = UnityCacheDatabase;
  UnityCache.RequestMetaDataStore = RequestMetaDataStore;
  UnityCache.MaximumCacheSize = 1024 * 1024 * 1024; // 1 GB

  /**
   * Load a request response from cache
   * @param {Request|string} request The fetch request
   * @returns {Promise<ResponseWithMetaData|undefined>} A cached response with meta
data for the request or undefined if request is not in cache.
   */
  UnityCache.prototype.loadRequest = function (request) {
    var self = this;

    return self.isConnected.then(function () {
      return Promise.all([
        self.cache.match(request),
        self.loadRequestMetaData(request)
      ]);
    }).then(function (result) {
      if (typeof result[0] === "undefined" || typeof result[1] === "undefined") {
        return undefined;
      }

      return {
        response: result[0],
        metaData: result[1]
      };
    });
  }

  /**
   * Load a request meta data from cache
   * @param {Request|string} request The fetch request
   * @returns {Promise<RequestMetaData>} Request meta data
   */
  UnityCache.prototype.loadRequestMetaData = function (request) {
    var url = typeof request === "string" ? request : request.url;

    return this.execute(RequestMetaDataStore.name, "get", [url]);
  }

  /**
   * Update meta data of a request
   * @param {RequestMetaData} metaData
   * @returns {Promise<void>}
   */
  UnityCache.prototype.updateRequestMetaData = function (metaData) {
    return this.execute(RequestMetaDataStore.name, "put", [metaData]);
  }

  /**
   * Store request in cache
   * @param {Request} request
   * @param {Response} response
   * @returns {Promise<void>}
   */
  UnityCache.prototype.storeRequest = function (request, response) {
    var self = this;

    return self.isConnected.then(function () {
      return self.cache.put(request, response);
    });
  }

  /**
   * Close database and cache connection.
   * @async
   */
   UnityCache.prototype.close = function () {
    return this.isConnected.then(function () {
      if (this.database) {
        this.database.close();
        this.database = null;
      }

      if (this.cache) {
        this.cache = null;
      }

    }.bind(this));
  }


  /**
   * Create a connection to Cache and IndexedDB for meta data storage
   * @private
   * @async
   * @returns {Promise<void>} A Promise that is resolved when a connection to the
IndexedDB and cache are established.
   */
  UnityCache.prototype.connect = function () {
    var self = this;

    if (typeof indexedDB === "undefined") {
      return Promise.reject(new Error("Could not connect to cache: IndexedDB is not
supported."));
    }

    if (typeof window.caches === "undefined") {
      return Promise.reject(new Error("Could not connect to cache: Cache API is not
supported."));
    }

    var isConnected = new Promise(function (resolve, reject) {
      try {
        // Workaround for WebKit bug 226547:
        // On very first page load opening a connection to IndexedDB hangs without
triggering onerror.
        // Add a timeout that triggers the error handling code.
        self.openDBTimeout = setTimeout(function () {
          if (typeof self.database != "undefined") {
            return;
          }

          reject(new Error("Could not connect to cache: Database timeout."));
        }, 20000);

        function clearOpenDBTimeout() {
          if (!self.openDBTimeout) {
            return;
          }

          clearTimeout(self.openDBTimeout);
          self.openDBTimeout = null;
        }

        var openRequest = indexedDB.open(UnityCacheDatabase.name,
UnityCacheDatabase.version);

        openRequest.onupgradeneeded =  self.upgradeDatabase.bind(self);
        openRequest.onsuccess = function (e) {
          clearOpenDBTimeout();
          self.database = e.target.result;
          resolve();
        };

        openRequest.onerror = function (error) {
          clearOpenDBTimeout();
          self.database = null;
          reject(new Error("Could not connect to database."));
        };
      } catch (error) {
        clearOpenDBTimeout();
        self.database = null;
        self.cache = null;
        reject(new Error("Could not connect to cache: Could not connect to
database."));
      }
    }).then(function () {
      var cacheName = UnityCacheDatabase.name + "_" + Module.companyName + "_" +
Module.productName;

      return caches.open(cacheName);
    }).then(function (cache) {
      self.cache = cache;
    });

    return isConnected;
  }

  /**
   * Upgrade object store if database is outdated
   * @private
   * @param {any} e Database upgrade event
   */
  UnityCache.prototype.upgradeDatabase = function (e) {
    var database = e.target.result;

    if (!database.objectStoreNames.contains(RequestMetaDataStore.name)) {
      var objectStore = database.createObjectStore(RequestMetaDataStore.name, {
keyPath: "url" });
      ["accessedAt", "updatedAt"].forEach(function (index) {
objectStore.createIndex(index, index); });
    }

    if (database.objectStoreNames.contains(RequestStore.name)) {
      database.deleteObjectStore(RequestStore.name);
    }

    if (database.objectStoreNames.contains(WebAssemblyStore.name)) {
      database.deleteObjectStore(WebAssemblyStore.name);
    }
  }

  /**
```

```
   * Execute an operation on the cache
   * @private
   * @param {string} store The name of the store to use
   * @param {string} operation The operation to to execute on the cache
   * @param {Array} parameters Parameters for the operation
   * @returns {Promise} A promise to the cache entry
   */
   UnityCache.prototype.execute = function (store, operation, parameters) {
    return this.isConnected.then(function () {
      return new Promise(function (resolve, reject) {
        try {
          // Failure during initialization of database -> reject Promise
          if (this.database === null) {
            reject(new Error("indexedDB access denied"))
            return;
          }

          // Create a transaction for the request
          var accessMode = ["put", "delete", "clear"].indexOf(operation) != -1 ?
"readwrite" : "readonly";
          var transaction = this.database.transaction([store], accessMode)
          var target = transaction.objectStore(store);
          if (operation == "openKeyCursor") {
            target = target.index(parameters[0]);
            parameters = parameters.slice(1);
          }

          // Make a request to the database
          var request = target[operation].apply(target, parameters);
          request.onsuccess = function (e) {
            resolve(e.target.result);
          };
          request.onerror = function (error) {
            reject(error);
          };
        } catch (error) {
          reject(error);
        }
      }.bind(this));
    }.bind(this));
  }

  UnityCache.prototype.getMetaDataEntries = function () {
    var self = this;
    var cacheSize = 0;
    var metaDataEntries = [];

    return new Promise(function (resolve, reject) {
      var transaction = self.database.transaction([RequestMetaDataStore.name],
"readonly");
      var target = transaction.objectStore(RequestMetaDataStore.name);
      var request = target.openCursor();

      request.onsuccess = function (event) {
        var cursor = event.target.result;
        if (cursor) {
          cacheSize += cursor.value.size;
          metaDataEntries.push(cursor.value);

          cursor.continue();
        } else {
          resolve({
            metaDataEntries: metaDataEntries,
            cacheSize: cacheSize
          });
        }
      };
      request.onerror = function (error) {
        reject(error);
      };
    });
  }

  /**
   * Clean up cache by removing outdated entries.
   * @private
   * @returns {Promise<void>}
   */
  UnityCache.prototype.cleanUpCache = function () {
    var self = this;

    return this.getMetaDataEntries().then(function (result) {
      var metaDataEntries = result.metaDataEntries;
      var cacheSize = result.cacheSize;
      var entriesToDelete = [];
      var newMetaDataEntries = [];

      // Remove cached entries with outdated product version
      for (var i = 0; i < metaDataEntries.length; ++i) {
        if (metaDataEntries[i].version == Module.productVersion) {
          newMetaDataEntries.push(metaDataEntries[i]);
          continue;
        }

        entriesToDelete.push(metaDataEntries[i]);
        cacheSize -= metaDataEntries[i].size;
      }

      // Remove cache entries until cache size limit is met
      newMetaDataEntries.sort(function (a,b) {
        return a.accessedAt - b.accessedAt;
      });

      for (var i = 0; i < newMetaDataEntries.length; ++i) {
        if (cacheSize < UnityCache.MaximumCacheSize) {
          break;
        }

        entriesToDelete.push(newMetaDataEntries[i]);
```

```
        cacheSize -= newMetaDataEntries[i].size;
      }

      function deleteMetaDataEntry(url) {
        return new Promise(function (resolve, reject) {
          var transaction = self.database.transaction([RequestMetaDataStore.name],
"readwrite");
          var target = transaction.objectStore(RequestMetaDataStore.name);
          target.delete(url);

          transaction.oncomplete = resolve;
          transaction.onerror = reject;
        });
      }

      function deleteEntries() {
        if (entriesToDelete.length === 0) {
          return Promise.resolve();
        }

        var entryToDelete = entriesToDelete.pop();
        return self.cache.delete(entryToDelete.url).then(function (deleted) {
          if (deleted) {
            return deleteMetaDataEntry(entryToDelete.url);
          }
        }).then(function () {
          return deleteEntries();
        });
      }

      return deleteEntries();
    });
  }

  return UnityCache;
}();
  Module.cachedFetch = function () {
  var UnityCache = Module.UnityCache;
  var fetchWithProgress = Module.fetchWithProgress;
  var readBodyWithProgress = Module.readBodyWithProgress;

  function log(message) {
    console.log("[UnityCache] " + message);
  }

  function resolveURL(url) {
    resolveURL.link = resolveURL.link || document.createElement("a");
    resolveURL.link.href = url;
    return resolveURL.link.href;
  }

  function isCrossOriginURL(url) {
    var originMatch = window.location.href.match(/^[a-z]+:\/\/[^\/]+/);
    return !originMatch || url.lastIndexOf(originMatch[0], 0);
  }
  function isCacheEnabled(url, init) {
    if (init && init.method && init.method !== "GET") {
      return false;
    }

    if (init && ["must-revalidate", "immutable"].indexOf(init.control) == -1) {
      return false;
    }

    if (!url.match("^https?:\/\/")) {
      return false;
    }

    return true;
  }

  function cachedFetch(resource, init) {
    var unityCache = UnityCache.getInstance();
    var url = resolveURL((typeof resource === "string") ? resource : resource.url);
    var cache = { enabled: isCacheEnabled(url, init) };
    if (init) {
      cache.control = init.control;
      cache.companyName = init.companyName;
      cache.productName = init.productName;
      cache.productVersion = init.productVersion;
    }
    cache.revalidated = false;
    cache.metaData = {
      url: url,
      accessedAt: Date.now(),
      version: cache.productVersion
    };
    cache.response = null;

    function fetchAndStoreInCache(resource, init) {
      return fetch(resource, init).then(function (response) {
        if (!cache.enabled || cache.revalidated) {
          return response;
        }

        if (response.status === 304) {
          // Cached response is still valid. Set revalidated flag and return cached
response
          cache.revalidated = true;

          unityCache.updateRequestMetaData(cache.metaData).then(function () {
            log("'" + cache.metaData.url + "' successfully revalidated and served from
the indexedDB cache");
          }).catch(function (error) {
            log("'" + cache.metaData.url + "' successfully revalidated but not stored
in the indexedDB cache due to the error: " + error);
          });
          return readBodyWithProgress(cache.response, init.onProgress,
init.enableStreamingDownload);
        } else if (response.status == 200) {
          // New response -> Store it and cache and return it
          cache.response = response;
          cache.metaData.updatedAt = cache.metaData.accessedAt;
          cache.revalidated = true;
          var clonedResponse = response.clone();

          return readBodyWithProgress(response, init.onProgress,
init.enableStreamingDownload).then(function (response) {
            // Update cached request and meta data
            cache.metaData.size = response.parsedBody.length;
            Promise.all([
              unityCache.storeRequest(resource, clonedResponse),
              unityCache.updateRequestMetaData(cache.metaData)
            ]).then(function () {
              log("'" + url + "' successfully downloaded and stored in the indexedDB
cache");
            }).catch(function (error) {
              log("'" + url + "' successfully downloaded but not stored in the
indexedDB cache due to the error: " + error);
            });

            return response;
          });
        } else {
          // Request failed
          log("'" + url + "' request failed with status: " + response.status + " " +
response.statusText);
        }

        return readBodyWithProgress(response, init.onProgress,
init.enableStreamingDownload);
      });
    }

    // Use fetch directly if request can't be cached
    if (!cache.enabled) {
      return fetchWithProgress(resource, init);
    }

    return unityCache.loadRequest(url).then(function (result) {
      // Fetch resource and store it in cache if not present or outdated version
      if (!result) {
        return fetchAndStoreInCache(resource, init);
      }

      var response = result.response;
      var metaData = result.metaData;
      cache.response = response;
      cache.metaData.size = metaData.size;
      cache.metaData.updatedAt = metaData.updatedAt;

      if (cache.control == "immutable") {
        cache.revalidated = true;
        unityCache.updateRequestMetaData(metaData).then(function () {
          log("'" + cache.metaData.url + "' served from the indexedDB cache without
revalidation");
        });

        return readBodyWithProgress(response, init.onProgress,
init.enableStreamingDownload);
      } else if (isCrossOriginURL(url) && (response.headers.get("Last-Modified") ||
response.headers.get("ETag"))) {
        return fetch(url, { method: "HEAD" }).then(function (headResult) {
          cache.revalidated = ["Last-Modified", "ETag"].every(function (header) {
            return !response.headers.get(header) || response.headers.get(header) ==
headResult.headers.get(header);
          });
          if (cache.revalidated) {
            unityCache.updateRequestMetaData(metaData).then(function () {
              log("'" + cache.metaData.url  + "' successfully revalidated and served
from the indexedDB cache");
            });

            return readBodyWithProgress(cache.response, init.onProgress,
init.enableStreamingDownload);
          } else {
            return fetchAndStoreInCache(resource, init);
          }
        });
      } else {
        init = init || {};
        var requestHeaders = init.headers || {};
        init.headers = requestHeaders;
        if (response.headers.get("Last-Modified")) {
          requestHeaders["If-Modified-Since"] = response.headers.get("Last-Modified");
          requestHeaders["Cache-Control"] = "no-cache";
        } else if (response.headers.get("ETag")) {
          requestHeaders["If-None-Match"] = response.headers.get("ETag");
          requestHeaders["Cache-Control"] = "no-cache";
        }

        return fetchAndStoreInCache(resource, init);
      }
    }).catch(function (error) {
      // Fallback to regular fetch if and IndexDB error occurs
      log("Failed to load '" + cache.metaData.url  + "' from indexedDB cache due to the
error: " + error);
      return fetchWithProgress(resource, init);
    });
  }

  return cachedFetch;
}();


  function downloadBinary(urlId) {
      progressUpdate(urlId);
      var cacheControl = Module.cacheControl(Module[urlId]);
      var fetchImpl = Module.companyName && Module.productName ? Module.cachedFetch :
Module.fetchWithProgress;
      var url = Module[urlId];
      var mode = /file:\/\//.exec(url) ? "same-origin" : undefined;

      var request = fetchImpl(Module[urlId], {
        method: "GET",
        companyName: Module.companyName,
        productName: Module.productName,
        productVersion: Module.productVersion,
        control: cacheControl,
        mode: mode,
        onProgress: function (event) {
          progressUpdate(urlId, event);
        }
      });

      return request.then(function (response) {
        return response.parsedBody;
      }).catch(function (e) {
        var error = 'Failed to download file ' + Module[urlId];
        if (location.protocol == 'file:') {
          showBanner(error + '. Loading web pages via a file:// URL without a web
server is not supported by this browser. Please use a local development web server to
host Unity content, or use the Unity Build and Run option.', 'error');
        } else {
          console.error(error);
        }
      });
  }

  function downloadFramework() {
      return new Promise(function (resolve, reject) {
        var script = document.createElement("script");
        script.src = Module.frameworkUrl;
        script.onload = function () {
          // Adding the framework.js script to DOM created a global
          // 'unityFramework' variable that should be considered internal.
          // If not, then we have received a malformed file.
          if (typeof unityFramework === 'undefined' || !unityFramework) {
            var compressions = [['br', 'br'], ['gz', 'gzip']];
            for(var i in compressions) {
              var compression = compressions[i];
              if (Module.frameworkUrl.endsWith('.' + compression[0])) {
                var error = 'Unable to parse ' + Module.frameworkUrl + '!';
                if (location.protocol == 'file:') {
                  showBanner(error + ' Loading pre-compressed (brotli or gzip) content
via a file:// URL without a web server is not supported by this browser. Please use a
local development web server to host compressed Unity content, or use the Unity Build
and Run option.', 'error');
                  return;
                }
                error += ' This can happen if build compression was enabled but web
server hosting the content was misconfigured to not serve the file with HTTP Response
Header "Content-Encoding: ' + compression[1] + '" present. Check browser Console and
Devtools Network tab to debug.';
                if (compression[0] == 'br') {
                  if (location.protocol == 'http:') {
                    var migrationHelp = ['localhost',
'127.0.0.1'].indexOf(location.hostname) != -1 ? '' : 'Migrate your server to use
HTTPS.'
                    if (/Firefox/.test(navigator.userAgent)) error = 'Unable to parse '
+ Module.frameworkUrl + '!<br>If using custom web server, verify that web server is
sending .br files with HTTP Response Header "Content-Encoding: br". Brotli compression
may not be supported in Firefox over HTTP connections. ' + migrationHelp + ' See <a
href="https://bugzilla.mozilla.org/show_bug.cgi?id=1670675">https://bugzilla.mozilla.or
g/show_bug.cgi?id=1670675</a> for more information.';
                    else error = 'Unable to parse ' + Module.frameworkUrl + '!<br>If
using custom web server, verify that web server is sending .br files with HTTP Response
Header "Content-Encoding: br". Brotli compression may not be supported over HTTP
connections. Migrate your server to use HTTPS.';
                  }
                }
                showBanner(error, 'error');
                return;
              }
            };
            showBanner('Unable to parse ' + Module.frameworkUrl + '! The file is
corrupt, or compression was misconfigured? (check Content-Encoding HTTP Response Header
on web server)', 'error');
          }

          // Capture the variable to local scope and clear it from global
          // scope so that JS garbage collection can take place on
          // application quit.
          var fw = unityFramework;
          unityFramework = null;
          // Also ensure this function will not hold any JS scope
          // references to prevent JS garbage collection.
          script.onload = null;
          resolve(fw);
        }
        script.onerror = function(e) {
          showBanner('Unable to load file ' + Module.frameworkUrl + '! Check that the
file exists on the remote server. (also check browser Console and Devtools Network tab
to debug)', 'error');
        }
        document.body.appendChild(script);
        Module.deinitializers.push(function() {
          document.body.removeChild(script);
        });
      });
  }

  function loadBuild() {
    downloadFramework().then(function (unityFramework) {
      unityFramework(Module);
    });

    var dataPromise = downloadBinary("dataUrl");
```

```
Module.preRun.push(function () {
      Module.addRunDependency("dataUrl");
      dataPromise.then(function (data) {
        var view = new DataView(data.buffer, data.byteOffset, data.byteLength);
        var pos = 0;
        var prefix = "UnityWebData1.0\0";
        if (!String.fromCharCode.apply(null, data.subarray(pos, pos + prefix.length))
== prefix)
          throw "unknown data format";
        pos += prefix.length;
        var headerSize = view.getUint32(pos, true); pos += 4;
        while (pos < headerSize) {
          var offset = view.getUint32(pos, true); pos += 4;
          var size = view.getUint32(pos, true); pos += 4;
          var pathLength = view.getUint32(pos, true); pos += 4;
          var path = String.fromCharCode.apply(null, data.subarray(pos, pos +
pathLength)); pos += pathLength;
          for (var folder = 0, folderNext = path.indexOf("/", folder) + 1 ; folderNext
> 0; folder = folderNext, folderNext = path.indexOf("/", folder) + 1)
            Module.FS_createPath(path.substring(0, folder), path.substring(folder,
folderNext - 1), true, true);
          Module.FS_createDataFile(path, null, data.subarray(offset, offset + size),
true, true, true);
        }
        Module.removeRunDependency("dataUrl");
      });
    });
  }

  return new Promise(function (resolve, reject) {
    if (!Module.SystemInfo.hasWebGL) {
      reject("Your browser does not support WebGL.");
    } else if (Module.SystemInfo.hasWebGL == 1) {
      var msg = "Your browser does not support graphics API \"WebGL 2\" which is
required for this content.";
      if (Module.SystemInfo.browser == 'Safari' &&
parseInt(Module.SystemInfo.browserVersion) < 15) {
        if (Module.SystemInfo.mobile || navigator.maxTouchPoints > 1)
          msg += "\nUpgrade to iOS 15 or later.";
        else
          msg += "\nUpgrade to Safari 15 or later.";
      }
      reject(msg);
    } else if (!Module.SystemInfo.hasWasm) {
      reject("Your browser does not support WebAssembly.");
    } else {
      Module.startupErrorHandler = reject;
      onProgress(0);
      Module.postRun.push(function () {
        onProgress(1);
        delete Module.startupErrorHandler;
        resolve(unityInstance);
      });
      loadBuild();
    }
  });
```

```
}
```