unity Ads

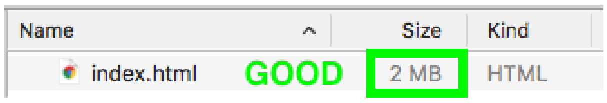
Playable Ads on Unity

Best practises and guides

V.1.0

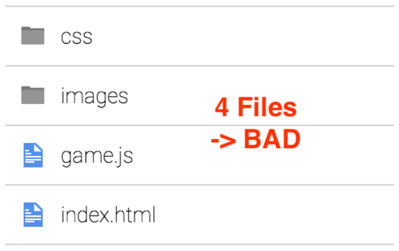
Technical Basics

Playable Ad MUST be contained in a single HTML index file
When a device wants to show an ad, it downloads only one file (for a Playable Ad, this would be the index.html file, for a video it would be an mp4 file) using a link to our CDN. This means, that we can't have a playable consisting of multiple files, simply because the device will only download one file and one file only.



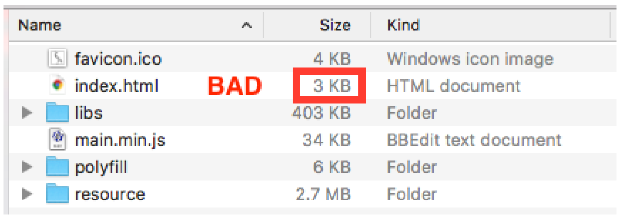
Name		Size	Kind
index.html	GOOD	2 MB	HTML

Check the picture below. There is the index.html file, but there is also a game.js file and two folders of which one contains images.
So, if the device now downloads only the index.html file, it will be missing all the other stuff.



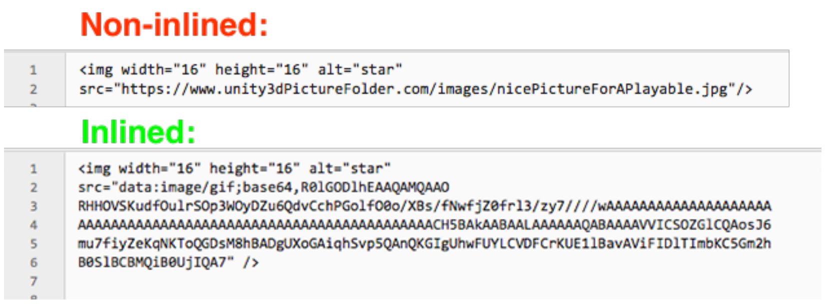
css
images
game.js
index.html

In the screenshot below, you can see that there are multiple files and folders. Also, the index.html file is only 3 KB in size, so all the actual data is contained somewhere else (here the main part is in "resource").



Name		Size	Kind
favicon.ico		4 KB	Windows icon image
index.html	BAD	3 KB	HTML document
libs		403 KB	Folder
main.min.js		34 KB	BBEdit text document
polyfill		6 KB	Folder
resource		2.7 MB	Folder

The single index.html file MUST be inlined and SHOULD be minified



Non-inlined:

```
<img width="16" height="16" alt="star"
src="https://www.unity3dPictureFolder.com/images/nicePictureForAPlayable.jpg"/>
```

Inlined:

```
<img width="16" height="16" alt="star"
src="data:image/gif;base64,R0lGODlhEAAQAMQAAO
RHHOVSKudfOulrSOp3WOyDZu6QdvCchPGolfO0o/XBs/fNwfjZ0frl3/zy7////wAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACH5BAkAABAALAAAAAAQABAAAAVVICSOZGlCQAosJ6
mu7fiyZeKqNKToQGDsM8hBADgUXoGAiqhSvp5QAnQKGIgUhwFUYLCVDFCrKUE1lBavAViFIDlTImbKC5Gm2h
B0SlBCBMQiB0UjIQA7" />
```

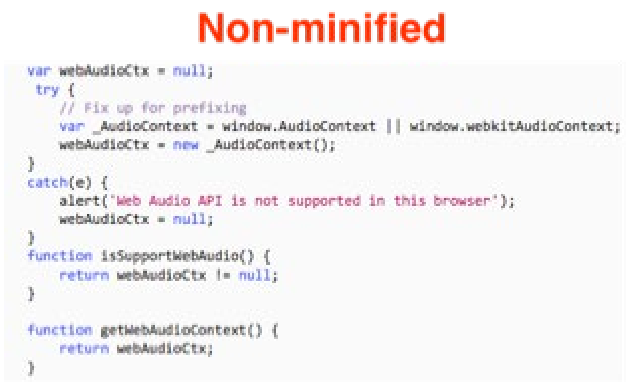
In the “BAD” screenshots from above the index.html file contains links to the other files and folders, meaning that when the device has only downloaded the index.html file and starts showing the playable, it actually needs to download some more stuff from those links, so the ad is not really fully cached to the device.

Also, these links are only relative links like /myParentFolder/resources/images/picture2.png and the device won’t find anything using this path because there is nothing there. Same problem when the links are pointing to some URL on the internet where e.g. the images are stored - caching does not work. So, instead of giving a link to an external source, we tell the browser/ad unit right there in the code what it is supposed to display.

Inlining is basically, a super long string of gibberish characters. It’s not gibberish to the browser in our ad unit though of course. This data is interpreted as the type of file you are saying it is - so as an image (or in this case actually a gif).

The same thing can be done with sound or video files. It is a rare practice, because for the normal usage of an html file (for a website for example), this is often unnecessary. If you simply want to update an image, you would need to change the source code of the website every time. However, for caching the entire ad to the device memory beforehand, it has to be done this way.

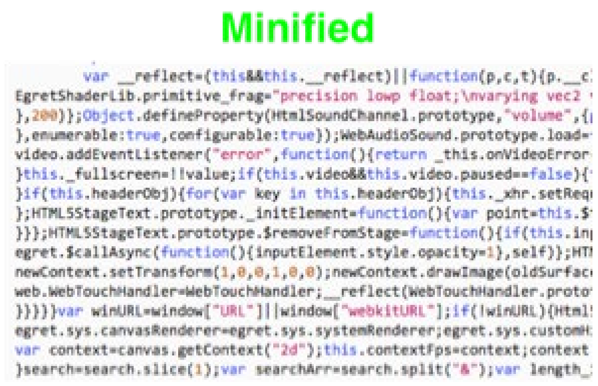
Minifying is a way of reducing the size of software code. It removes empty spaces and shortens variable names etc. It is still perfectly well readable for a machine, but nearly impossible to decipher for a human.



Non-minified

```
var webAudioCtx = null;
 try {
    // Fix up for prefixing
    var _AudioContext = window.AudioContext || window.webkitAudioContext;
    webAudioCtx = new _AudioContext();
}
catch(e) {
    alert('Web Audio API is not supported in this browser');
    webAudioCtx = null;
}
function isSupportWebAudio() {
    return webAudioCtx != null;
}

function getWebAudioContext() {
    return webAudioCtx;
}
```

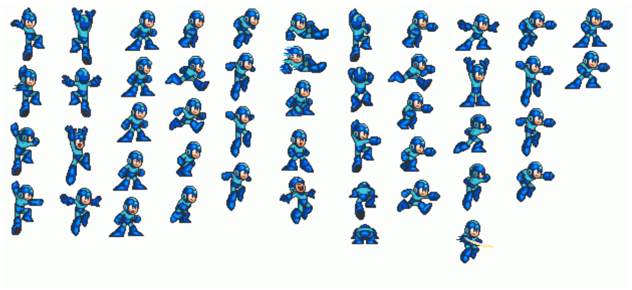


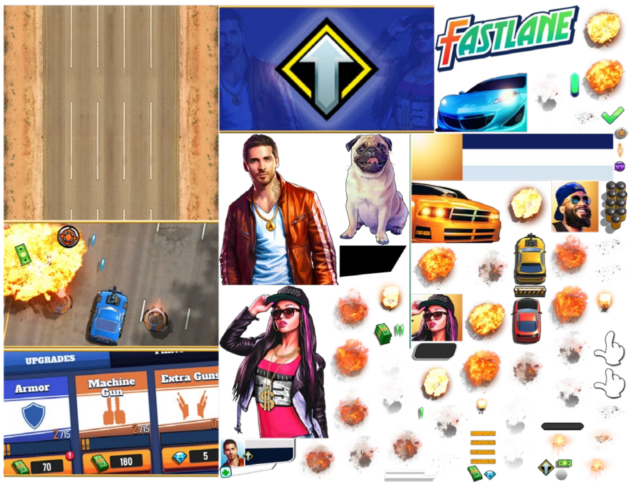
Minified

```
      var __reflect=(this&&this.__reflect)||function(p,c,t){p.__c
EgretShaderLib.primitive_frag="precision lowp float;\nvarying vec2
},200)};Object.defineProperty(HtmlSoundChannel.prototype,"volume",{
},enumerable:true,configurable:true});WebAudioSound.prototype.load=
video.addEventListener("error",function(){return _this.onVideoError
}this._fullscreen=!!value;if(this.video&&this.video.paused==false){
}if(this.headerObj){for(var key in this.headerObj){this._xhr.setReq
};HTML5StageText.prototype._initElement=function(){var point=this.$
}}};HTML5StageText.prototype.$removeFromStage=function(){if(this.in
egret.$callAsync(function(){inputElement.style.opacity=1},self)};HT
newContext.setTransform(1,0,0,1,0,0);newContext.drawImage(oldSurfac
web.WebTouchHandler=WebTouchHandler;__reflect(WebTouchHandler.proto
}}}})var winURL=window["URL"]||window["webkitURL"];if(!winURL){Html
egret.sys.canvasRenderer=egret.sys.systemRenderer;egret.sys.customH
var context=canvas.getContext("2d");this.contextFps=context;context
}search=search.slice(1);var searchArr=search.split("&");var length_
```

Additional notes:

Art assets:
Use spritemaps to contain the art assets. These are sprite maps/sheets/databases:





It can be used to show the movements of a character, essentially it can be used to animate.

A sprite sheet is a bitmap image file that contains several smaller graphics in a tiled grid arrangement. By compiling several graphics into a single file, you enable Animate and other applications to use the graphics while only needing to load a single file.

Browser testing:
Make the final build to work without mraid as well, so that you can successfully run and complete it in desktop browser. Tapping download button in desktop browser should go to the app store page of your choice.

Download button functionality:
Make sure that download button has visual tap / click response, so users understand it was clicked.

Linking to the Appstores should be simple mraid.open calls:
case “Android”:mraid.open(“https://play.google.com/store/apps/details?id=yourgame”); break;
case “iOS”:mraid.open(“https://itunes.apple.com/us/yourgame?mt=8”);

Filesize limitation:
Final filesize of the ad should be under 5 Mb.

Audio:
If your playable is using sound, it must use Web Audio API by default and fallback to HTMLAudioElement or HTML DOM Audio Object. Most modern game frameworks and sound libraries (Howler.js) are working this way. Make sure you do too.

Web Audio API is currently the only solution that respects the state of the mute/ringer switch on iOS. Also, use `viewableChangeHandler` to mute or play sound if the app goes background on Android.

MRAID Basics:

Bootstrapping:

```
// Wait for the SDK to become ready
if (mraid.getState() === 'loading') {
   mraid.addEventListener('ready', onSdkReady);
} else {
   onSdkReady();
}

function viewableChangeHandler(viewable) {
   // start/pause/resume gameplay, stop/play sounds
   if(viewable) {
      showMyAd();
   } else {
      // pause
   }
}

function onSdkReady() {
   mraid.addEventListener('viewableChange', viewableChangeHandler);

   // Wait for the ad to become viewable for the first time
   if (mraid.isViewable()) {
     showMyAd();
   }
}
function showMyAd() {
   ...
}
```

Download Button in Playable Ad - How to open the store:

```
// Detect platform from user agent
var userAgent = navigator.userAgent || navigator.vendor;

var url = '<replace with iTunes store>';
var android = '<replace with Google Play Store link>';

if (/android/i.test(userAgent)) {
   url = android;
}

mraid.open(url);
```

UnityAds MRAID requirements

Mobile Rich Media Ad Interface Definitions

Mraid is injected by our webview and you can just use the mraid methods.
Unity Ads is supporting MRAID v2.0 with the following exceptions:

Support only for full-screen interstitial ads, expand() and resize() will not be available
Custom close buttons will not be available
No support for: sms, tel, calendar and storePicture
inlineVideonext is supported from SDK 2.0.7 onwards

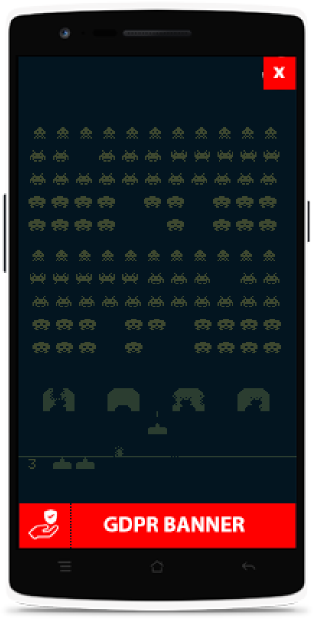
Requirements outside the MRAID specification:

- Advertisement must be contained in a single HTML file. All assets must be inlined
- Device and OS versions expected to be supported must be provided by the implementer.
- Unity Ads SDK supports Android 4.4+ and iOS 8.0+
- Skippability of the advertisement is controlled by Unity Ads SDK
- Advertisements should be designed to support both portrait and landscape orientations
- Advertisements should be designed not to need any network requests (XHR), but for example analytics calls to trusted provides might be allowed
- Advertisement should go directly to the app store/play store using mraid.open() when designing the click-through, any start/view/click attribution will be handled server-side
- There will be a Unity Ads loading screen overlay from SDK side that will be shown when the AdUnit starts
- This loading screen will be shown for min 2 sec and the goal is not only to cover up initialization time but also to prepare the player for a playable ad to come
- Start initialization when receiving the MRAID “ready” event
- Wait for the MRAID “viewableChange” event before starting the actual playable content

Gameplay and visual layout

Entire ad duration should be designed around 30 seconds.
Aim to have the endscreen splash appear in about 27 seconds for optimal time to suggest a download, instead of the close button.

For non-rewarded ads, the close button will appear after 5 seconds.

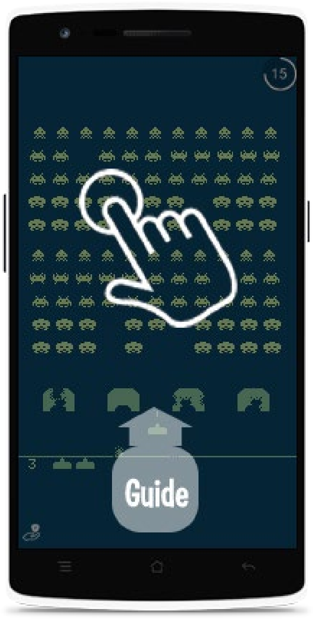


FIXED ELEMENTS
These come pre-built from Unity

Do not place any functionality to these areas
Countdown timer in the top right corner
Timer turns into a close button when ad is closeable

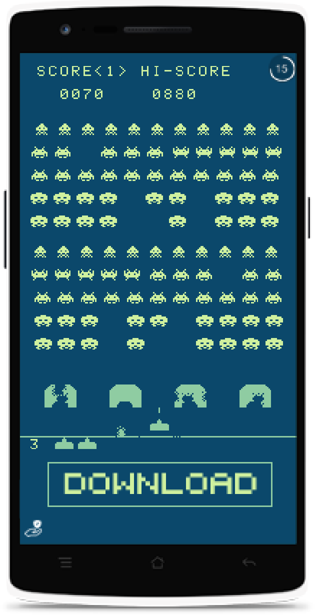
In the EU region, GDPR banner on the bottom
The banner is visible for the first time an ad is shown
The banner takes about 100 pixels in height

Only the privacy icon will be shown on the bottom left corner if:
- Not in the EU region
- Consent given for personalized ads
- GDPR banner was seen by the user once



START SCREEN
A short guide for the gameplay

· Gameplay should start immediately to catch attention
· Fade or animate into gameplay 20 seconds if no input is given
· Show initial interaction by examples
· Show more, tell less



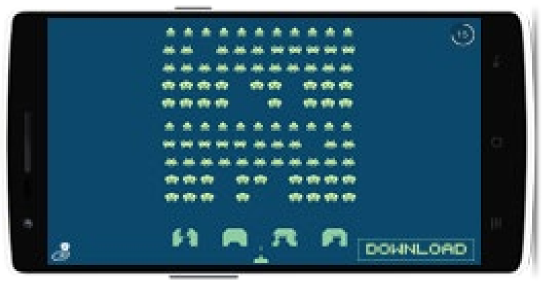
GAMEPLAY

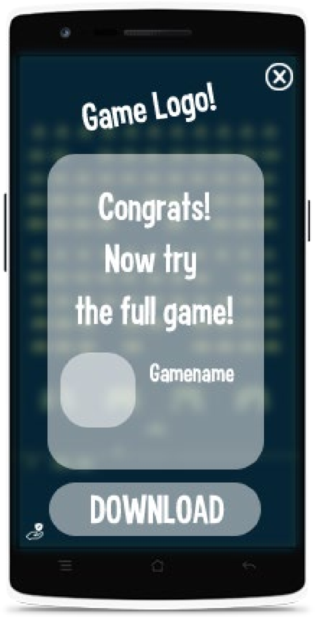
Allow the player to finish the game or fail at it

· Create a Download button to fit the game
· Make sure that the game also works in both orientations
  Design the gameplay to work atleast comparatively well
  on both.

You can fill the excess space with a wallpaper or move the Download button there for example.

Ads get served to both portrait and landscape locked games so forcing the user to turn the device is a bad ad experience





END SCREEN

Either fail message or congratulations

· Include basic app data, game icon, name, logo etc.
· Animate the download button to give it more attention,
  but do not use a strobe effect. Apple does not allow it on
  their platform.

unity Ads

Unity MRAID Testing apps

We have a testing app that you can use to test playables in our real ad unit instead of in a browser.

To try out the playable ad properly, you can download our testing app:

For iOS:
Click the link described in the email you also received with this document with your iPhone/iPad. Or, enter the following in the Safari browser on the iOS device:
(link starting from and including itms-services...)

itms-services://?action=download-manifest&url=https://s3.amazonaws.com/applifier_deployment/ios/enterprise/uads-creative.plist

Safari will ask if you'd like to open iTunes, tap yes.
iOS will present a modal asking you to install 'UnityAdsExample', tap yes.
The app will appear to install as any other iOS app, tapping it before trusting the enterprise certificate will present a modal asking you to do that.
Open the Settings app, tap General, tap Profiles & Device Management, select the Applifier profile and approve it - the app will now be able to open as expected.

For Android:
Download and install the attached APK you also received with this document.
Android OS might ask you to trust web/download sources to allow the install.

Usage:
Upload and host the playable .html file somewhere it can be accessed with an URL.
Please then use any QR code generator (e.g. http://www.qr-code-generator.com/) to generate a QR code from the URL of the playable.

You can then go to the test app and use the QR code reader to read the URL and initialise the ad.

Note: For iOS, the URL that hosts the playable has to use secure http**s** instead of just http. due to Apple platform security.

When you are finished and are submitting the ad:

This might sound self explanatory, but make sure the playable actually loads/plays correctly. Play it a few times to catch any potential bugs or functionality.

Make sure that the playable is all inlined into one HTML file and is under 5 Megabytes.

When submitting the playable to us, please also report the platform and minimum OS Version requirements to avoid unnecessary testing.
(e.g. “this is for iOS 9+ and Android 5.0+”)

FAQs

When testing, make sure, the URL you are hosting the playable in is using httpS not only http. iOS requires https due to the platform holders security concerns.

Check the setting for CORS headers on the server where the file is hosted:
allow origin needs to be set to *****

Make sure the filesize of the completed ad is under 5 Mb.

Some of the playables won’t work in a browser, try it with the Unity Test App and on several different devices if possible, from old Androids to the latest iPhone.

Some of the playables will work in a browser but fail to implement MRAID correctly and thus fail in our ad unit. In that case it is hard for you to spot that and we can check it in the final QA. When you use our testing apps, you can spot many issues beforehand.

The SDK supports iOS 8+ and Android 4.4.1+ for playables
Common Android checks Android 4.4, 5.0 - generally polyfill issues (object.assign, etc.)

Things to look for:
Performance throughout the whole playable.
Install button leads to correct store and game.
(be sure to check an actual iOS device as simulator runtime doesn’t have an App Store)

Duration (how long does it take to reach the end screen and CTA)
Orientation Changes - making sure game still functions/looks okay when orientation is switched from portrait to landscape and vice versa.

Sound Issues
For example, if a user locks the screen, the sound should stop.
If the screen is unlocked, the sound should restart.
Check that the physical mute button is respected.