SPECIFYING DEPENDENCIES

View online: https://www.construct.net/en/make-games/manuals/addon-sdk/reference/specifying-dependencies

Plugins and behaviors can specify dependencies on additional files, or Cordova plugins for inclusion with the Cordova exporter. Dependencies are added using the `AddFileDependency(opts)` and `AddCordovaPluginReference(opts)` methods on both IPluginInfo and IBehaviorInfo. Remote scripts can also be added with `AddRemoteScriptDependency`, but this is not recommended.

File dependencies

A file dependency refers to another file in the addon. Note the file must be bundled with the addon; you cannot refer to URLs elsewhere on the Internet. There are several kinds of file dependency, which correspond to the `type` property in the options object:

copy-to-output

This simply causes the file to be copied to the output folder when exporting. The file is also available in preview mode. This is useful for bundling additional resources, such as an image file that needs to be loaded at runtime, or a script that is dynamically loaded.

external-dom-script

A script dependency that is included via the addition of an extra `<script>` tag in the exported HTML file. The `scriptType` option can be set to `"module"` to load the script as a module (see below). Note in worker mode the script is loaded in the DOM, so is not directly available to the runtime code in the worker. The script is not minified on export. This is suitable for large external libraries that the addon references.

external-runtime-script

A script dependency that is included via the addition of an extra `<script>` tag in the exported HTML file, or loaded on the worker with `importScripts()` in worker mode. This means the script is always directly available to runtime code. However the dependency must be designed to work in a Web Worker, e.g. not assuming the DOM is present. The script is not minified on export.

external-css

A stylesheet dependency that is included via the addition of an extra `<link rel="stylesheet">` tag in the exported HTML file, in case the addon needs to specify CSS styles.

wrapper-extension

A DLL to be bundled for a wrapper extension. See Wrapper extensions for more details.

To add a file dependency, call `AddFileDependency` with an options object, such as in this example:

```
this._info.AddFileDependency({
        filename: "mydependency.js",
        type: "external-dom-script"
});
```

The options object uses the following properties.

filename

Name of the dependency file in the addon. This must be bundled with the addon; it cannot refer to a URL. It is recommended to bundle the script with your addon, but if you must use a URL, see the section *Remote script dependencies*. The file path is relative to the root (the location of addon.json).

> *For developer mode addons, make sure the dependency file is also included in the `"file-list"` key. For more information see the section on Developer mode addons in Addon metadata.*

type

One of the types described above, e.g. `"external-dom-script"`.

fileType

When `type` is `"copy-to-output"`, this must specify the MIME type of the file. For example if including `"image.png"` as a `"copy-to-output"` dependency, the `fileType` must be set to `"image/png"`.

scriptType

Currently only supported when `type` is `"external-dom-script"`. By default external DOM scripts are loaded as "classic" scripts. This property can be set to the string `"module"` to instead load the external DOM script as a module (i.e. with `<script src="filename.js" type="module"></script>`).

platform

When `type` is `"wrapper-extension"`, this specifies the platform architecture of the DLL. The supported options are `"windows-x86"` (for 32-bit Windows), `"windows-x64"`, `"windows-arm64"`, `"xbox-uwp-x64"` (for Xbox UWP export option only), `"macos-universal"` (for macOS WKWebView, with a universal binary including both Intel and Apple Silicon code), `"linux-x64"`, `"linux-arm"` (for 32-bit ARM) and `"linux-arm64"`.

Cordova plugin dependencies

Addons can specify dependencies on Cordova plugins. These only apply to the Cordova exporter, which covers both Android and iOS. When exporting a Cordova project, the additional Cordova plugin dependencies are automatically included in the exported **config.xml**. This allows convenient integration of a Construct addon with a Cordova plugin.

To add a Cordova plugin dependency, call `AddCordovaPluginReference` with an options object, such as in this example:

```
this._info.AddCordovaPluginReference({
	id: "cordova-plugin-inappbrowser"
});
```

If you wish to provide variables to the Cordova plugin, use the `variables` property of the options object to pass an array of `[variableName, pluginProperty]` pairs. In this case the plugin must also be passed in the `plugin` property. An example is shown below.

```
const property = new SDK.PluginProperty("integer", "test-property", 0);

this._info.SetProperties([
	property
]);

this._info.AddCordovaPluginReference({
	id: "cordova-plugin-inappbrowser",
	plugin: this,
	variables: [
		["MY_VAR", property]
	]
});
```

Cordova plugins that require variables will not compile if the variable is omitted from config.xml.

See Cordova plugin variables for more information.

The options object uses the following properties.

id

The ID of the Cordova plugin to reference.

version Optional

A version spec for the Cordova plugin, e.g. `"1.0.4"`. If this is not specified, the latest version will be used.

platform Optional

Specify a specific platform the Cordova plugin applies to. By default this is `"all"` meaning it will be used in both Android and iOS exports. However it can be set to `"android"` or `"ios"` to only be included when exporting to a specific platform. This is useful to switch between different Cordova plugins on different platforms.

variables Optional

Specify variables to be used with the Cordova plugin as an array of `[variableName, pluginProperty]` pairs. The variable name is bound to a `SDK.PluginProperty`. When the project is exported a variable is added under the plugin reference in config.xml with the given name and a value taken from the specified property. When variables are specified, the `plugin` property must also be set.

plugin Optional

Used to specify the plugin when using variables. Normally this should be set to `this`.

Note for security reasons the Construct mobile app build service does not allow arbitrary Cordova plugins to be used. The build service uses an allowlist of allowed Cordova plugins. If you'd like a Cordova plugin to be added to the allowlist, please file an issue on the Construct issue tracker. Please note we cannot guarantee that Cordova plugins will be allowed, and approval is subject to a security review. Other build systems, including compiling with the Cordova CLI, do not impose this restriction.

Cordova resource file dependencies

Addons can further specify dependencies on additional resource files for Cordova exports. When exporting a Cordova project, the additional Cordova resource file dependencies are automatically included in the exported **config.xml** as `<resource file src="..." target="...">` tags.

To add a Cordova resource file dependency, call `AddCordovaResourceFile` with an options object, such as in this example:

```
this._info.AddCordovaResourceFile({
        src: "myfile.txt"
});
```

This will insert `<resource-file src="myfile.txt">` to the exported config.xml.

More information about how resource files are used in Cordova can be found in the Cordova documentation.

The options object uses the following properties.

src

The `src` attribute of the `resource-file` tag. Location of the file relative to config.xml.

target Optional

The `target` attribute of the `resource-file` tag. Path to where the file will be copied.

platform Optional

Specify a specific platform the Cordova plugin applies to. By default this is `"all"` meaning it will be used in both Android and iOS exports. However it can be set to `"android"` or `"ios"` to only be included when exporting to a specific platform.

Remote script dependencies

Plugins and behaviors may also specify remote script dependencies. These are loaded from a cross-origin URL, e.g. `https://example.com/api.js`.

Avoid using remote script dependencies where possible. They have some drawbacks:

- *Construct 3 games work offline. However remote scripts cannot be cached for offline use, so will fail to load when the user is offline.*
- *Remote scripts can also fail to load due to unreliable connections or service outages.*
- *In some native apps, e.g. Cordova on Android/iOS, the native platform may block any access to URLs that are not on an allowlist of allowed origins. This can cause the script to fail to load unless the user does additional configuration of their app.*

Prefer using a file dependency instead, and bundle the script with your addon. If you must use a remote script, ensure your addon gracefully handles the case the remote script fails to load.

Use `AddRemoteScriptDependency` to add a remote URL to load a script from. The second parameter can optionally be set to `"module"` to load the script as a module. For example:

```
// Load remote "classic" script
this._info.AddRemoteScriptDependency("https://example.com/api.js");

// Load remote module script
this._info.AddRemoteScriptDependency("https://example.com/api-module.js", "module");
```

This will produce the following tags on export, loaded before the runtime:

```
<!-- Classic script -->
<script src="https://example.com/api.js"></script>

<!-- Module script -->
<script src="https://example.com/api-module.js" type="module"></script>
```

*The script URL must **not** use `http:` in its URL. On the modern web this will often be blocked from secure sites as mixed content. You must either use secure HTTPS, or a same-protocol URL like `//example.com/api.js`.*