PORTING TO ADDON SDK V2

View online: https://www.construct.net/en/make-games/manuals/addon-sdk/guide/porting-addon-sdk-v2

Construct formerly supported a legacy SDK, referred to as the Addon SDK v1. As of Construct 3 r450+, released in 2025, it now only supports a modern replacement with industry-standard encapsulation, referred to as the Addon SDK v2. This guide covers what you need to do to upgrade an existing addon using the Addon SDK v1 to v2.

Only plugins and behaviors need updating to the Addon SDK v2. This is because the main change is to how runtime scripts are written. Effects and themes do not need porting, as they do not use runtime scripts. Further, plugins and behaviors should not need any significant modifications to any of their editor code, DOM-side runtime code, or wrapper extensions; usually only the runtime scripts in the *c3runtime* folder need significant changes.

Porting guide

Follow these steps to update a plugin or behavior from Addon SDK v1 to v2.

Step 1: update SDK version

In addon.json, update the `"sdk-version"` field to 2. If the field is missing, add it with the value 2. This indicates to Construct that the addon is using the Addon SDK v2.

You'll probably want to also increment the `"version"` field as the update to Addon SDK v2 warrants an updated addon version too.

Step 2: update base classes

The base classes used in the Addon SDK v2 are different to v1. In other words, the class after the `extends` keyword needs to be updated.

The following table lists the changes to base classes for plugins.

SDK v1 base class	SDK v2 base class
C3.SDKPluginBase	globalThis.ISDKPluginBase
C3.SDKTypeBase	globalThis.ISDKObjectTypeBase
C3.SDKInstanceBase	globalThis.ISDKInstanceBase
C3.SDKWorldInstanceBase	globalThis.ISDKWorldInstanceBase
C3.SDKDOMPluginBase	globalThis.ISDKDOMPluginBase
C3.SDKDOMInstanceBase	globalThis.ISDKDOMInstanceBase

The following table lists the changes to base classes for behaviors.

SDK v1 base class	SDK v2 base class
C3.SDKBehaviorBase	globalThis.ISDKBehaviorBase
C3.SDKBehaviorTypeBase	globalThis.ISDKBehaviorTypeBase
C3.SDKBehaviorInstanceBase	globalThis.ISDKBehaviorInstanceBase

Step 3: update class constructors

In the Addon SDK v2, the constructors of the above base classes do not use any parameters. For example with the Addon SDK v1 class `C3.SDKInstanceBase`, the constructor used two parameters. With the Addon SDK v2 class `globalThis.ISDKInstanceBase`, the constructor uses no parameters.

Further, for instance classes specifically, instead of the `properties` parameter being passed in the constructor, instead call `this._getInitProperties()` to access the same information. An example of the difference is shown below.

```
// Instance constructor in Addon SDK v1
constructor(inst, properties)
{
	super(inst);

	if (properties)
	{
		// ... read properties ...
	}

	// ... rest of constructor ...
}

// Instance constructor in Addon SDK v2
constructor()
{
	super();

	const properties = this._getInitProperties();
	if (properties)
	{
		// ... read properties ...
	}

	// ... rest of constructor ...
}
```

Further, for DOM or wrapper extension plugins, note that the DOM component ID or wrapper extension ID are now passed as part of an options object in the `super()` call, e.g.:

```
// In ISDKDOMPluginBase or ISDKDOMInstanceBase constructor:
super({ domComponentId: DOM_COMPONENT_ID });

// In ISDKInstanceBase constructor for a wrapper extension:
```

```
super({ wrapperComponentId: "my-extension" });
// (instead of calling SetWrapperExtensionComponentId())
```

Step 4: remove separate script interface

In the Addon SDK v2, runtime script classes *are* the script interface classes: all public properties and methods are also accessible from Construct's scripting feature. Therefore there is no longer any need to define a separate script interface class.

The method `GetScriptInterfaceClass()` can be deleted and the entire script interface class deleted if one was specified.

To ensure backwards compatibility, if you did have a script interface class, make sure the main class now implements all the same properties and methods as the script interface class used to.

Step 5: update property and method names

While all previously documented properties and methods in SDK v1 are still supported in SDK v2, they have been renamed to follow the naming conventions of the rest of the scripting APIs. In some cases some details are different to simplify the SDK or to adapt to the particular requirements of the new architecture.

For example the SDK v1 instance method `GetDebuggerProperties()` is now named `_getDebuggerProperties()`, but otherwise works identically; `Trigger()` is now named `_trigger()` but otherwise works identically; and so on. Refer to the class links in the above table to review the full reference. Note also some features may now be in other base classes; for example the SDK v1 class `SDKInstanceBase` had a property `this._runtime`; the SDK v2 class `ISDKInstanceBase` does not define a property for the runtime, but it inherits from IInstance, which defines the property `this.runtime`.

For consistency, you will likely want to rename any other class methods to follow the new camelCase naming convention. As per the Addon SDK coding conventions, you may wish to use an underscore prefix to indicate methods which should not be called from the scripting feature, but cannot be made private.

Step 6: update remaining code

In some addons, such as if the main purpose is to integrate a third-party service, the addon should not require any further major changes. However if your addon has extensive logic using the runtime APIs from the Addon SDK v1, these will need to be rewritten in terms of the new runtime APIs based on Construct's scripting feature. A full reference of the available APIs can be found in the scripting reference section of Construct's manual.

Recommended architecture

Our recommended architecture is to implement core logic as methods and properties (or setters/getters) on your main instance class. This makes the features accessible to the scripting feature. Then actions, conditions and expressions just call the script APIs, so they work consistently with the scripting feature, and don't have to have any significant logic in ACE methods. Then the debugger methods can use the same methods so the debugger views and changes things consistently with the way actions/expressions or script APIs would. Overall this means event sheets, scripting, and the debugger all share the same implementation, which also makes maintenance easier, as there is only one place code needs to be updated.

Recommended additional changes

The following changes are not required, but are recommended.

Delete PLUGIN_VERSION

In the editor plugin script, delete `PLUGIN_VERSION` and delete the line `this._info.SetVersion(PLUGIN_VERSION);`. This is because with the Addon SDK v2, the addon version is taken from the version specified in addon.json only; the version in the editor plugin script is now ignored. A deprecation warning will appear in the console until these changes are made.

Use globalThis

In the past the global object may have been referred to using `window` or `self`. The modern standardized way to do this is with `globalThis`, so it is recommended to use that instead of any other alternatives.

Configure to use modules

The Addon SDK v2 now supports using JavaScript Modules (i.e. `import` and `export` statements) in your runtime scripts. By default Construct creates a new main script module for your addon so you don't have to set anything else up, but it's good practice to set it up anyway as it's how all modern addons are written, and it's necessary if you want to use `import` or `export` in your runtime scripts. For more details see the section *Configuring use of modules* in Runtime scripts.

Sample diffs

On the Construct Addon SDK GitHub repository, you can find commits that update the SDK samples from the Addon SDK v1 to the Addon SDK v2. The differences, or "diffs", in these commits can serve as a reference of what needs to be changed to update addons. With the sample addons the changes take in to above everything described above, including recommended changes. The following links display the diffs for the updates for each SDK sample.

- Update customImporterPlugin to addon SDK v2
- Update domElementPlugin to addon SDK v2

- Update domMessagingPlugin to addon SDK v2
- Update drawingPlugin to addon SDK v2
- Update editorTextPlugin to addon SDK v2
- Update singleGlobalPlugin to addon SDK v2
- Update wrapperExtensionPlugin to addon SDK v2
- Update sample behavior to addon SDK v2

Publishing

Addon developers should now be publishing updates to their addons using SDK v2. Support for the legacy SDK v1 will only continue until the middle of 2025, although beyond that support for SDK v1 addons will be extended until the end of 2026 with an LTS release.

In the Addons section of the website, there is a checkbox labelled **Uses Addon SDK v2** when editing an addon. When uploading an SDK v2 addon it should automatically detect that your addon uses SDK v2 and check this box for you. It's worth double-checking the checkbox for existing addons or after publishing an update using SDK v2, as this then ensures it is listed when using the *SDKv2 addons only* filtering option, which will eventually be enabled by default and therefore hide all old addons using SDK v1 by default unless this option has been checked.