# CONFIGURING PLUGINS

**View online:** https://www.construct.net/en/make-games/manuals/addon-sdk/guide/configuring-plugins

The main configuration for a plugin is set in **plugin.js**.

## Plugin constants

The following constants are defined in the file-level scope:

```
const PLUGIN_ID = "MyCompany_MyAddon";
const PLUGIN_CATEGORY = "general";
```

The ID and version constants must match the values specified in addon.json.

**PLUGIN_ID**

This is a unique ID that identifies your plugin from all other addons. This must not be used by any other addon ever published for Construct 3. **It must never change** after you first publish your addon. (The name is the only visible identifier of the addon in the Construct 3 editor, so that can be changed any time, but the ID must always be the same.) To ensure it is unique, it is recommended to use a vendor-specific prefix, e.g. `MyCompany_MyAddon`.

**PLUGIN_CATEGORY**

The category for the plugin when displaying it in the *Create New Object Type* dialog. This must be one of `"3d"`, `"data-and-storage"`, `"form-controls"`, `"general"`, `"input"`, `"media"`, `"monetisation"`, `"platform-specific"`, `"web"`, `"other"`.

## Updating plugin identifiers

The main class declaration of the plugin looks like this:

```
const PLUGIN_CLASS = SDK.Plugins.MyCompany_MyAddon = class MyCustomPlugin extends SDK.IPluginBas
```

Be sure to update the identifiers to describe your own plugin, in both the SDK namespace and the `class` name.

### Updating in type.js and instance.js

Likewise in both type.js and instance.js, you must update the following:

- `PLUGIN_CLASS` to refer to your plugin's name

- The `class` name suffixed with `Type` or `Instance`. (For example the Audio plugin uses `AudioPlugin`, `AudioType` and `AudioInstance` as the three names.)

### Optional plugin scripts

With the addon SDK, you may omit the editor script files type.js and instance.js, as well as the runtime script files plugin.js and type.js. If these files are omitted, it uses the default base class with no modifications. To remove these files, be sure to do the following:

1. Delete any unused script files
2. Remove the files from the `"file-list"` array in addon.json
3. Remove any unused editor script files from the `"editor-scripts"` array in addon.json
4. In the editor plugin script, call `this._info.SetC3RuntimeScripts()` with an array of the runtime script files in use, as the default list includes c3runtime/plugin.js and c3runtime/type.js.

## The plugin constructor

The main function of plugin.js is to define a class representing your plugin. In the class constructor, the configuration for the plugin is set via the `this._info` member, which is an IPluginInfo interface. The constructor also reads potentially translated strings from the language subsystem.

For more information about the possible plugin configurations, see the `IPluginInfo` reference.

## Specifying plugin properties

The plugin properties appear in the Properties Bar when instances of the plugin are selected. To set which properties appear, pass an array of PluginProperty to `this._info.SetProperties`. An example is shown below. For more details see the `PluginProperty` reference.

```
this._info.SetProperties([
        new SDK.PluginProperty("integer", "test-property", 0)
]);
```