IBEHAVIORINFO INTERFACE

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

IBehaviorInfo defines the configuration for a behavior. It is typically accessed in the behavior constructor via `this._info`.

Methods

SetName(name)

Set the name of the addon. Typically this is read from the language file.

SetDescription(description)

Set the description of the addon. Typically this is read from the language file.

SetVersion(version)

Set the version string of the addon, in A.B.C.D form. Typically this is set to the `BEHAVIOR_VERSION` constant.

SetCategory(category)

Set the category of the addon. Typically this is set to the `BEHAVIOR_CATEGORY` constant. It must be one of `"attributes"`, `"general"`, `"movements"`, `"other"`.

SetAuthor(author)

Set a string identifying the author of the addon.

SetHelpUrl(url)

Set a string specifying a URL where the user can view help and documentation resources for the addon.

SetIcon(url, type)

Set the addon icon URL and type. By default the URL is `"icon.svg"` and the type is `"image/svg+xml"`. It is recommended to leave this at the default and use an SVG icon, since it will scale well to any display size or density. However you can change your addon to load a PNG icon with `SetIcon("icon.png", "image/png")`.

SetIsOnlyOneAllowed(isOnlyOneAllowed)

Set a boolean of whether the behavior is allowed to be added more than once to the same object. The default is `false`, which means the behavior can be added multiple times to the same object. Set to `true` to only allow it to be added once to each object.

SetIsDeprecated(isDeprecated)

Set a boolean of whether the addon is deprecated or not. If you wish to replace your addon with another one, the old one can be deprecated with `SetIsDeprecated(true)`. This makes it invisible in the editor so it cannot be used in new projects; however old projects with the addon already added can continue to load and work as they did before. This discourages use of the deprecated addon without breaking existing projects that use it.

SetCanBeBundled(canBeBundled)

Pass `false` to prevent the addon from being bundled via the *Bundle addons* project property. By default all addons may be bundled with a project, and it is recommended to leave this enabled for best user convenience. However if you publish a commercial addon and want to prevent it being distributed by project-bundling, you may wish to disable this.

SetProperties(propertiesArray)

Set the available addon properties by passing an array of PluginProperty. See Configuring Behaviors for more information.

AddCordovaPluginReference(opts)

Add a dependency on a Cordova plugin, that will be included when using the Cordova exporter. For more information see Specifying dependencies.

AddFileDependency(opts)

Add a dependency on another file included in the addon. For more information see Specifying dependencies.

AddRemoteScriptDependency(url) Not recommended

Add a script dependency to a remote URL (on a different origin). For more information see Specifying dependencies.

SetC3RuntimeScripts(arr)

Pass an array of strings to set the list of runtime scripts the addon uses. The default list is the following, and note that this method entirely replaces it: *"c3runtime/behavior.js", "c3runtime/type.js", "c3runtime/instance.js", "c3runtime/conditions.js", "c3runtime/actions.js", "c3runtime/expressions.js"*.

AddC3RuntimeScript(path)

Add a single runtime script path to the existing list of runtime scripts the addon uses, e.g. "c3runtime/additionalScript.js".

SetRuntimeModuleMainScript(path)

Set the main script that the runtime loads as a module. When this method is called, Construct will only load that script, and it is expected that all your other scripts are imported in the main script. If this method is not called, Construct automatically generates a main script that imports every single runtime script - but note that makes it difficult to use modules properly. See runtime scripts for more information.

SetScriptInterfaceNames(opts)

Use this method to tell Construct the names of your script interface classes. This is necessary to generate the correct TypeScript definition files. `opts` is an object which allows specifying the names for the `instance`, `behaviorType` and `behavior` interface names as necessary, e.g.:

```
this._info.SetScriptInterfaceNames({
        instance: "IBulletBehaviorInstance"
});
```

SetTypeScriptDefinitionFiles(arr)

Specify an array of TypeScript definition files (.d.ts) your addon provides. This should be used to provide full TypeScript definitions of any script interfaces your addon provides, which is necessary for projects using TypeScript with your addon. Example:

```
this._info.SetTypeScriptDefinitionFiles(["c3runtime/IBulletBehaviorInstance.d.ts"]);
```