ISDKINSTANCEBASE ADDON SDK INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkinstancebase

The `ISDKInstanceBase` interface is used as a runtime base class for instances in the addon SDK. It derives from IInstance.

ISDKInstanceBase APIs

_release()

Optional override for when an instance is released. Clean up any other resources that need releasing in this method. It must also call `super._release()` to release all engine resources.

_getInitProperties()

Call this in the constructor to get an array of the instance's properties to create the instance with.

_trigger(method)

_triggerAsync(method)

Fire a trigger condition. The condition must be declared as a trigger in aces.json. Pass a full reference to the condition method, e.g. `this._trigger(C3.Plugins.MyPlugin.Cnds.MyTrigger)`. The async variant returns a Promise that resolves when the trigger has finished executing, which can be used to support Construct's debugger, as it may wait if it hits a breakpoint inside the trigger.

_addDOMMessageHandler(handler, callback)

_addDOMMessageHandlers(arr)

Add a callback to be run to handle a message posted from a DOM-side script. The handler is a string identifier. The callback receives the posted data as an argument. Note that if the caller in the DOM-side script originally used the `PostToRuntimeAsync` method, the callback may be an `async` function, and the return value is posted back to the DOM-side script. The `_addDOMMessageHandlers` variant accepts an array of `[handler, callback]` which is convenient when adding multiple handlers.

_postToDOM(handler, data)

_postToDOMAsync(handler, data)

Post a message to a DOM-side script. The handler is a string identifier. The `data` must be structurally clonable (since it is posted down a MessageChannel).
The async method returns a promise that resolves with the DOM-side callback's return value. The non-async method does not return a value and the DOM-side callback's return value is discarded (i.e. a "fire and forget" message).

_postToDOMMaybeSync(handler, data)

As with `_postToDOM()`, but when the runtime is in DOM mode, calls the DOM handler synchronously inside the call. When the runtime is in worker mode, this still posts a message which is handled later. Usually this method is not necessary, but it can be used to work around some user input restrictions in some browsers in DOM mode only.

_setTicking(isTicking)

_setTicking2(isTicking)

_isTicking()

_isTicking2()

Utility methods to start or stop the runtime calling the `_tick()` or `_tick2()` methods of your instance every tick, and also to check whether ticking is active. It is recommended to stop ticking whenever the tick method is no longer needed to reduce the performance overhead of ticking. Redundant calls to start or stop ticking are ignored. The first call always takes effect (i.e. calls do not stack - if you make 3 calls to start ticking then 1 call to stop ticking, ticking is stopped).

_tick()

Optional override that is called every tick just **before** events are run after `_setTicking(true)` has been called.

_tick2()

Optional override that is called every tick just **after** events are run after `_setTicking2(true)` has been called.

_getDebuggerProperties()

Override to return properties to display in the debugger. For more information see runtime scripts.

_saveToJson()

Optional override to return a JSON object that represents the state of the instance for savegames.

_loadFromJson(o)

Optional override accepting a JSON object returned by a prior call to `_saveToJson()` that represents the state of an instance to load, for savegames.

Wrapper extension methods

These methods relate to the use of wrapper extensions. Refer to that manual section for more details; for completeness the relevant methods are also included here.

_isWrapperExtensionAvailable()

Returns a boolean indicating whether the corresponding wrapper extension was successfully loaded. If this returns false then no messages sent to the wrapper extension will be received, and async messages will return a promise that never resolves.

_addWrapperExtensionMessageHandler(messageId, callback)

_addWrapperMessageHandlers(list)

Add a callback to be run to handle a message posted from the corresponding wrapper extension. The callback receives the JSON data sent from the wrapper extension as an argument. The `_addWrapperMessageHandlers` variant accepts an array of `[messageId, callback]` which is convenient when adding multiple handlers.

_sendWrapperExtensionMessage(messageId, params)

_sendWrapperExtensionMessageAsync(messageId, params)

Send a message to the wrapper extension. The message ID is used to identify the kind of message. `params` is an optional array of parameters to provide. These may only be boolean, number or string type values. The async variant returns a Promise that resolves when the wrapper extension responds to the message. The promise resolves with the JSON data sent from the wrapper extension.