ISDKUTILS ADDON SDK INTERFACE

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

The `ISDKUtils` interface provides general APIs intended for use with the Addon SDK. It is normally accessed via `runtime.sdk`.

ISDKUtils APIs

addLoadPromise(promise)

Only valid while the project is still loading. Add a promise that the runtime will wait to resolve before starting the first layout. This is useful if you want to make sure your addon loads an asynchronous resource before the game starts.

> *Single-global plugins create their instance before loading, so this method can be used in single-global instance constructors. Otherwise it can only be used in the plugin or behavior constructor.*

updateRender()

By default, Construct does not render a new frame unless something has changed. If something in your addon is changed which will affect the way it draws, call this method to ensure Construct draws a new frame to reflect the changes. Avoid calling this method unnecessarily - only call it when something has actually changed (be sure not to call it if e.g. setting something to the same value).

isWrapperExtensionAvailable(wrapperComponentId)

Returns a boolean indicating whether a wrapper extension with the specified ID 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. This is similar to the corresponding ISDKInstanceBase method, but allows checking for any wrapper extension rather than just the one associated with the plugin. In general it is recommended to use the instance-specific methods as cases where you need to message a different wrapper extension to your own are rare.

sendWrapperExtensionMessage(wrapperComponentId, messageId, params)

sendWrapperExtensionMessageAsync(wrapperComponentId, messageId, params)

These methods are equivalent to those in ISDKInstanceBase, but allow specifying an arbitrary wrapper component ID. In general it is recommended to use the instance-specific

methods as cases where you need to message a different wrapper extension to your own are rare.

---

getObjectClassBySid(sid)

Returns an IObjectClass with the given SID (Serialization ID), or null if none found. This can be used to look up the `IObjectClass` for plugin "object" properties, which pass a SID at runtime for the property value.

---

isAutoSuspendEnabled

Set or get a boolean indicating whether the runtime automatically suspends when it detects the page or app going in to the background. This is enabled by default; by turning it off you can then use `setSuspended()` to control when the runtime suspends and resumes. Note however that even if automatic suspending is disabled, browsers may effectively suspend background pages anyway through other means, such as throttling timers and callbacks. Therefore disabling automatic suspending does not necessarily mean the runtime actually is able to continue running in the background. This API exists mainly to comply with the requirements of platforms that provide their own suspend and resume events that are required to be used.

---

setSuspended(isSuspended)

Pass `true` to suspend the runtime, and `false` to resume it. When suspended, the runtime stops ticking and drawing anything, and remains inactive. Make sure that suspend and resume calls are paired one-to-one: for example do not suspend once but resume twice.

> *The current suspended state can be read via the property* `runtime.isSuspended`.

---

constructVersionCode

A read-only number representing the Construct release the project has been previewed or exported with. A version code is the release number multiplied by 100, plus the patch number. For example r123.4 has the version code 12304. This can be useful for compatibility checks where feature detection (such as checking if methods exist) is not sufficient.

---

createLoopingConditionContext(loopName)

This method is intended specifically for looping conditions (where `"isLooping": true` is specified in the condition definition). It allows "retriggering" the event, which executes all subsequent conditions, actions and sub-events in one call; a loop is implemented by repeatedly retriggering the event. The loop name is optional but allows passing the name to the system *loopindex* condition to retrieve the index in nested loops. It returns an ILoopingConditionContext. An example looping condition implementation is shown below.

```
// Sample looping condition method
TestLoop(count)
{
        const loopCtx = this.runtime.sdk.createLoopingConditionContext();

        for(let i = 0; i < count; ++i)
        {
                loopCtx.retrigger();

                if (loopCtx.isStopped)
                        break;
        }

        loopCtx.release();
}
```