# WORLDINSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iworldinstance

---

The `IWorldInstance` script interface represents a single instance of an object type (represented by IObjectClass) that appears in a layout. It derives from the IInstance script interface.

Many objects return a more specific class deriving from `IInstance` or `IWorldInstance` to add APIs specific to the plugin. See the Plugin interfaces reference for more information.

## Getting an IWorldInstance

Instances are typically accessed through IObjectClass methods like `getFirstInstance()`. For example, `runtime.objects.Sprite.getFirstInstance()` will return the first instance of the Sprite object type.

> *Try not to confuse object classes with object instances. A common mistake is to try to use something like `runtime.objects.Sprite.x` to get the X co-ordinate of a Sprite instance. However `runtime.objects.Sprite` is an IObjectClass, which does not have a position. First add another call to get an instance before trying to read instance properties, for example `runtime.objects.Sprite.getFirstInstance().x`.*

## World instance events

The following events can be listened for on any `IWorldInstance` using the `addEventListener` method. See instance event for standard event properties. Note many more kinds of addon-specific events can be fired. See the documentation on each addon's script interfaces for more information.

---

**"hierarchyready"**

Fired for the root instance in a hierarchy after all instances have finished creating. During creation of a hierarchy it is uncertain whether other instances in the hierarchy have been created yet, which can sometimes complicate initializing hierarchies. When this event fires all instances in the hierarchy have been created, including triggering *On created* in event sheets, and so it is a suitable time to perform initialization of an entire hierarchy. As this only fires for the root instance, if you wish to iterate the rest of the instances in the hierarchy, use the `allInstances()` generator method of the root instance.

# General APIs

**layout**

An ILayout interface representing the layout the instance is on.

**layer**

An ILayer interface representing the layer the instance is on.

**x**

**y**

**setPosition(x, y)**

**getPosition()**

The position of this instance, in layout co-ordinates. The methods allow setting or getting both co-ordinates at the same time.

**offsetPosition(dx, dy)**

Adjust the position by adding `dx` to the X co-ordinate and `dy` to the Y co-ordinate.

**zElevation**

The Z elevation of the instance, relative to the layer it is on.

**totalZElevation**

A read-only value indicating the Z elevation of the instance including its layer's Z elevation.

**originX**

**originY**

**setOrigin(originX, originY)**

**getOrigin()**

The current origin point of this instance, represented as a normalized position in the range [0, 1]. For example the origin (0, 0) is in the top-left corner, and (0.5, 0.5) is in the middle regardless of the size of the object. The methods allow setting or getting both co-ordinates at the same time.

> *With Sprite objects, changing the animation frame will also update the origin according to the origin placement in the Animations Editor.*

**width**

**height**

**setSize(width, height)**

**getSize()**

The size of this instance, in layout co-ordinates. The methods allow setting or getting both values at the same time.

---

**angle**

The angle of the instance in radians. If this is changed, `angleDegrees` updates accordingly.

---

**angleDegrees**

The angle of the instance in degrees. If this is changed, `angle` updates accordingly.

---

**getBoundingBox(ignoreMesh = false)**

Return a DOMRect representing the axis-aligned bounding box of the instance in layout co-ordinates. By default this takes in to account any changes from the mesh distortion feature - specifying `false` for the `ignoreMesh` parameter will return a bounding box as if the object did not use mesh distortion.

> *This returns a copy of the bounding box. The returned DOMRect does not change if the instance changes, nor does changing the DOMRect affect the instance.*

---

**getBoundingQuad(ignoreMesh = false)**

Return a DOMQuad representing the bounding quad of the instance in layout co-ordinates. This is always a rectangle, but unlike the bounding box can represent rotation. By default this takes in to account any changes from the mesh distortion feature - specifying `false` for the `ignoreMesh` parameter will return a bounding quad as if the object did not use mesh distortion.

> *This returns a copy of the bounding quad. The returned DOMQuad does not change if the instance changes, nor does changing the DOMQuad affect the instance.*

---

**isVisible**

A boolean indicating whether the instance is visible in the layout.

---

**isOnScreen()**

Returns true if any part of the object's bounding box is within the screen area (performing the same check as the *Is on-screen* condition). This is not affected by the object's visibility or

opacity.

---

**opacity**

The opacity of the instance, as a floating point number in the range [0, 1], where 0 is fully transparent and 1 is fully opaque.

---

**colorRgb**

An array with 3 elements specifying the red, green and blue color filter of the instance, with color values as floats in the 0-1 range.

---

**blendMode**

A string indicating the current blend mode of the instance, controlling how it draws over the background. The allowed strings are the same as accepted by the IRenderer method `setBlendMode()`.

---

**effects**

An array of IEffectInstance representing the effect parameters for each effect on the instance.

## Z order APIs

---

**moveToTop()**

**moveToBottom()**

Move the instance to the top or the bottom of its current layer in the Z order.

---

**moveToLayer(layer)**

Move the instance to the top of a different layer given by its ILayer.

---

**moveAdjacentToInstance(other, isAfter)**

Move the instance adjacent to `other` (another `IWorldInstance`) in the Z order. If necessary this also moves the instance to the same layer as `other`. If `isAfter` is true, it moves it just above the given instance, else just below.

---

**zIndex**

A read-only integer indicating the instance's current index in the Z order on its current layer, starting at 0 for the back of the current layer, and increasing as it moves to the front.

## Collision APIs

See also the ICollisionEngine interface for more collision APIs.

---

**isCollisionEnabled**

Set or get a boolean indicating whether collisions are enabled for this instance. If disabled, the instance will always fail all overlap or collision checks.

---

**containsPoint(x, y)**

Test if a point intersects this instance, using its collision polygon if any, and return a boolean indicating if the point is inside the instance's collision area.

---

**testOverlap(wi)**

Test if this instance overlaps another world instance given by an `IWorldInstance`, returning `true` if they overlap, else `false`. This uses the object's collision polygons if any. If either instance has collisions disabled, this will always return `false`.

---

**testOverlapSolid()**

Test if this instance overlaps any instance with the Solid behavior. This returns the instance interface class for the first instance with the solid behavior that was found to overlap this instance, or `null` if none. This uses the object's collision polygons if any and respects solid collision filtering.

> *The return value of this method is truthy when an overlap is found and falsey when not, so this can be used directly in an `if` statement.*

## Mesh distortion APIs

---

**createMesh(hsize, vsize)**

Create a mesh for deforming the appearance of the object with the given number of mesh points horizontally and vertically. The minimum size is 2.

---

**releaseMesh()**

Releases any mesh that has been created, reverting back to default rendering of the object with no mesh distortion. Ignored if no mesh created.

---

**setMeshPoint(col, row, opts)**

Alter a given point in a created mesh given by its zero-based column and row. `opts` is an object that may specify the following properties:

- `mode`: a string of `"absolute"` (default) or `"relative"`, determining how to interpret the `x`, `y`, `u` and `v` options.

- `x` and `y`: the mesh point position offset, in normalized co-ordinates [0, 1] across the object size. These are allowed to go outside the object bounds. In relative mode these are added to the mesh point's current position.

- `u` and `v`: the texture co-ordinate for the mesh point, in normalized co-ordinates [0, 1]. These are not allowed to go outside the object bounds. These can be omitted, or in absolute mode be set to -1, to indicate not to change the texture co-ordinate from the default.

- `zElevation`: the Z elevation of the mesh point, allowing for distortion in 3D. Similarly to Z elevation of entire objects, this moves the mesh point up and down on the Z axis.

---

**getMeshPoint(col, row)**

Return an object describing the currently set mesh point at its zero-based column and row. The returned object has the same properties as the `opts` argument of `setMeshPoint()` uses, except for `mode`. The returned values are all absolute values (as relative mode is only relevant when applying changes).

---

**getMeshSize()**

Return the size of the mesh as `[hsize, vsize]` (corresponding to the size passed to `createMesh()`) if one is created. If no mesh has been created, returns `[0, 0]`.

## Scene graph APIs

---

**getParent()**

Return the parent `IWorldInstance` of this instance in the scene graph hierarchy if any, else `null`.

---

**getTopParent()**

Return the top parent of this instance in the scene graph hierarchy (which by definition has no parent itself) if any, else `null`.

---

***parents()**

A generator method that can be used to iterate all the instance's parents, up to the top parent.

### getChildCount()

Returns the number of children that have been added to this instance in the scene graph hierarchy.

### getChildAt(index)

Of the children that have been added to this instance, return the child instance at the given zero-based index. If the index is out of bounds, returns `null`.

### *children()

A generator method that can be used to iterate all the instance's added children.

### *allChildren()

A generator method that can be used to iterate all the instance's children recursively, i.e. including children of children, down to the bottom of the scene graph hierarchy.

### addChild(wi, opts)

Add another world instance given by an `IWorldInstance` as a child of this instance in the scene graph hierarchy. This instance becomes its parent in the scene graph hierarchy. The child will move, scale and rotate with this instance according to the provided options specified in the object `opts`, which supports the following properties:

- `transformX`: move the child with this instance's X position
- `transformY`: move the child with this instance's Y position
- `transformWidth`: scale the child with this instance's width
- `transformHeight`: scale the child with this instance's height
- `transformAngle`: rotate the child with this instance's angle
- `transformZElevation`: move the child with this instance's Z elevation
- `transformOpacity`: change the child's opacity according to the parent's opacity
- `transformVisibility`: make the child invisible if the parent is also invisible

- `destroyWithParent`: automatically destroy the child if this instance is destroyed

Each option is a boolean which defaults to `false` if omitted, so only `true` properties need to be specified.

> *Instances can only have one parent. If the given instance is already added as a child of something else, this method will have no effect.*

---

### getHierarchyOpts()

Return an object with properties representing the options specified for this child when it was added to its parent. The returned object has the same properties as the `opts` argument of `addChild()` uses, such as `transformX`, with boolean values indicating whether they are enabled. Therefore the returned object can be passed directly to another call to `addChild()` to re-use the same options.

---

### removeChild(wi)

Remove an existing child given by an `IWorldInstance` that was previously added with `addChild()`. The child is detached from the scene graph hierarchy and this instance will no longer act as its parent. The removed child still keeps its own children, if it has any.

---

### removeFromParent()

Shorthand method for `wi.getParent().removeChild(wi)`, i.e. removes this instance from its parent if it has any. If the instance has no parent, the method has no effect.