Events: On-Update Event Reference

Execution Client

Every script in a Horizon world has an **execution client**; this is the device that is currently running the script. For a Default script the execution client is a server somewhere, for a local script the execution client is one of the player's headsets (based on whoever currently has ownership, or it is the server if ownership has not been transferred to a player, or has been transferred back).

Render Frame

Each execution client repeatedly needs to: run scripts, calculate physics updates, move the player (e.g. for locomotion), update particle effects, etc. An execution client does all these updates over and over many times a second. We'll refer to each set of updates as a "**render frame**" or just a "frame".

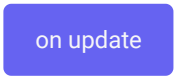
When the execution client is the server, there are about 60 execution frames per second and when the execution client is a player's headset, it runs at the device's framerate (e.g. 72 for Quest 2). For example, this means that physics data updates and scripts run 72 times a second for a script with player-ownership. See below for more details on the order updates occur in.

on update code block reference

Events ➤ Events ➤ on update

Run code blocks every frame, specifying to run them before or after the physics engine updates.

Appearance in Library



Description

The “On Update” code block allows you to specify whether you want to receive the event before the physics system and scripted events run (“PrePhysics”) or after scripted events (“Default”).

Why does execution order matter? When building interactive experiences it's important to think about the execution order of scripting, in relation to the physics engine. For example, if you want a “halo” to follow a user's head then you want that script to happen *after* the physics engine runs (note that the physics engine includes updating the players) so that the halo follows the player. But if you are making a car you want to move the car *before* the player is moved so that the player follows the car.

PrePhysics Update

If you want players or objects to move in response to an object or player *X* in *the same frame* then you should update *X* inside of a PrePhysics update.

For example if you want to implement a vehicle, a moving platform, or anything that moves the player, then you want it to run before the player updates. Likewise if you wanted a cart to crash through a stack of barrels, you would move the cart in a pre-update so that car moves *before* the physics update runs.

Moving an object during PrePhysics update will allow scripts to read the position of that object on the same frame (the script and Default updates), if the scripts are on the same client.

Default Update

If you want a player or object *X* to move in response to other objects that moved in the *same frame* then you should update *X* inside of a Default update.

This allows you to write code that reacts to (or reads data from) player or object movement that occurred due to physics, locomotion, and script events (e.g. from “send event”, “when object is grabbed”, etc) previously in the frame. If you wanted a laser gun to follow a player in a “holster” then you would want to move the laser gun after the player is moved.

The Default update option updates after script events. If a script moves an object, Default update can find the position of the object in the same frame, if they're on the same client.

Parameters

stage	**PrePhysics** The PrePhysics setting specifies that this code block, and those nested within it, should run every frame and that it should run *before* the frame's physics update (where physical objects, players, and recorded animations update). --- **Default** The Default setting specifies that this code block, and those nested within it, should run every frame and that it should run *after* the frame's physics update (where physical objects, players, and recorded animations update).
# deltaTime	The elapsed time in seconds since the last frame. The value is calculated as how much time passed between the start of this frame and the start of the previous frame. This value does not change during a frame (it is cached at frame-start). This value is useful for moving objects at a consistent speed, no matter the framerate. To move an object 1 meter per second in the upwards direction, move it by (0, 1, 0) * deltaTime each frame, which you can now easily do with the “On Update” event. Likewise you can take the position of an object across two frames, get the difference, and divide by the deltaTime to get its instantaneous velocity in meters per second.

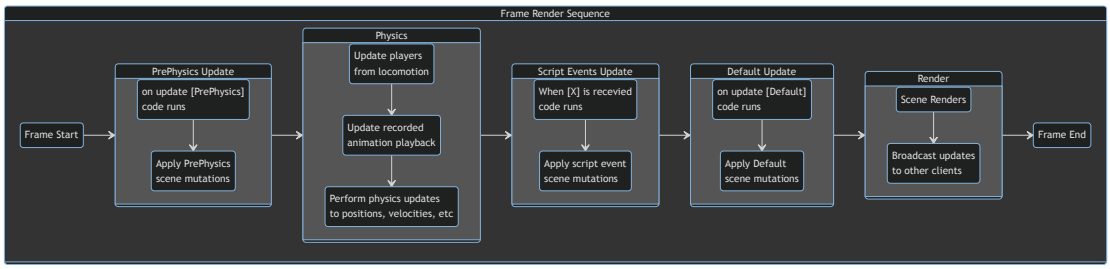
Execution Order

Clients send data out to other clients at the end of each frame. Each execution client runs the scripting system every frame. At the end of a frame, data is broadcast out to other execution clients. This data needs to go across the network so there will be inherent variance in how long the data takes to send and arrive. So for example, if a local script modifies the position of an object, then at the end of that frame it will send out that change and all other execution clients will receive that information shortly thereafter.

Script update stages don't apply mutations to the world until the end of the stage and sent events are held until the next frame. In one frame there are three script stages (PrePhysics, Script Events, and Default). When a stage runs, the top-level event handlers are run (such as "when player enters trigger" or "on update" or "when [myEvent]"). If these handlers do "send event" then those events are stored until the "script events" stage of the next frame. Likewise "send event with delay" will get executed in the "script events" stage of some future frame after the delay has passed. *There is no way to "send event" and have that event run in the same frame".* If you move an object with "moveTo" then that object will not actually have that new position until the end of that script event.

Physics operations are only applied in the physics update. If you push or spin an object, update the player's velocity, etc then those updates will occur in the next physics update. If those updates were in the "PrePhysics" stage then the physics update will be that same frame, otherwise they will end up in the next frame (since they are happening after this frame's physics update has already occurred).

Here's the execution order of an entire frame (each execution client runs this order every frame):



Every “send event” and code block that runs contribute to the per-frame limits (e.g. in all stages combined you cannot execute more than 2048 script events on a single execution client).

There are some key observations about the above sequence:

- **“On update” events within the same update have no guaranteed order.** If multiple scripts respond to the PrePhysics event on the same execution client, then can execute in any order. One script can also have multiple “On Update PrePhysics” blocks as well; the order they execute in is also unspecified.

- **“Move player” is a scene mutation and not a physics event.** Player movement from the joystick is applied during the Physics stage. Player movement from forces / velocity updates are also applied during the Physics stage. However the “move player to/by” code block is not a physics block and operates just like “move to”, meaning that you can use that code block during any scripting stage and the data will be updated at the end of that scripting stage. E.g. if you “move player” in the Script Events stage (perhaps due to a “trigger enter” event) then the player will have that new position at the end of that stage, and that position will be

seen by “position of player” in the following Default stage of the same frame.

- **Physics is only updated once in a frame.** If you apply physics (push / spin / add velocity / etc) then that update will occur in the next Physics update, whenever that is. If you are in the PrePhysics update then that will be in the next stage, otherwise it will wait until the next frame’s Physics stage.
- **Scene mutations done in the PrePhysics update are not “visible” until the script events update on the same client and after a network sync for other clients.** If you modify an object’s position in a PrePhysics Event you will not see that new position in any other code that runs in the PrePhysics event on the same frame. So if you do “move to” and then in the very next line you get the “position of” you will not see the new value until the script events update of that frame. Other clients won’t see this data until it is synchronized across the network at the end of the frame.
- **Scene mutations done in the script events update are not “visible” until the Default update on the same client and after a network sync for other clients.** If you modify an object’s position in a script event (e.g. from “send event”, “send event with delay”, “when object is grabbed”, etc) then that client won’t see that data until the Default update in that frame, and other clients won’t see it until they eventually get the network event.
- **Scene mutations done in the Default update are not “visible” until the next frame on the same client and after a network sync for other clients.** If you modify an object’s position in a Default event, that client won’t see that data until the next frame, and other clients won’t see it until they eventually get the network event. This is how events have always worked in Horizon, independent of the On Update block.

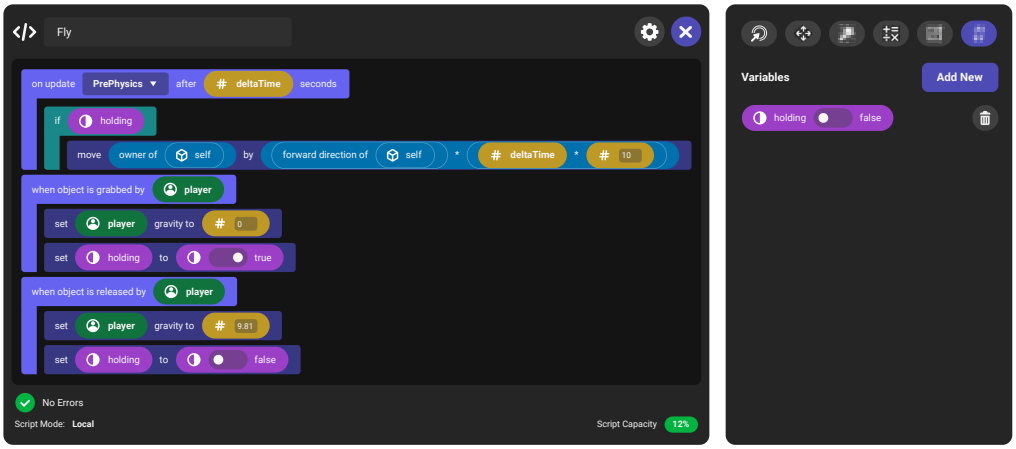
Examples

Pull player forward with held object

What it does: This example moves a player in the forward direction of an object that they are holding so that they can fly.

How it works: We want to look at where the held object is and then use that to move the player, so we want to run the script before the player update occurs. Thus the PrePhysics update is used to send the “move player” command.

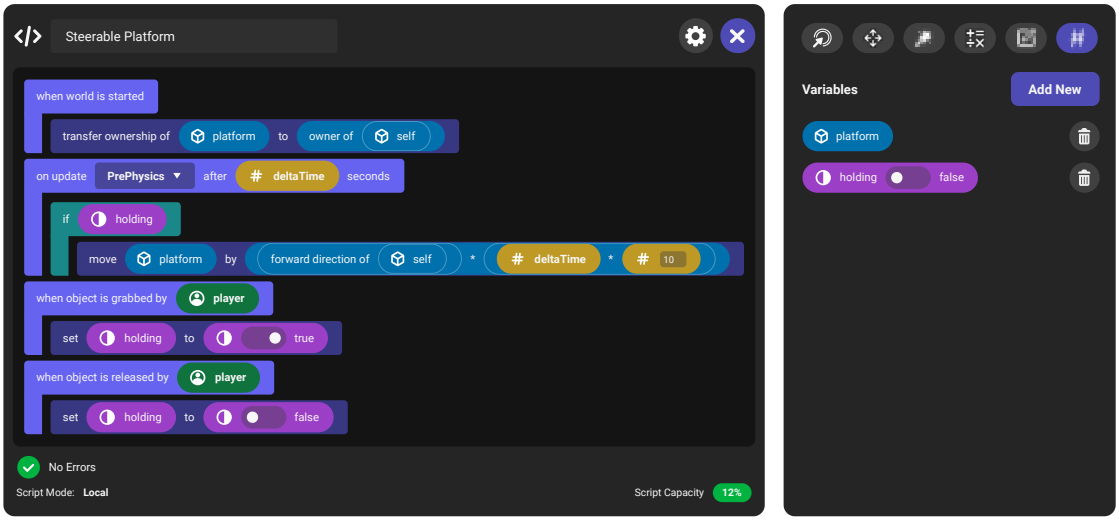
If Default update was used instead, the grabbed object would lag behind. PrePhysics updates the player position before the object position so that the object can accurately move to the player's hand. We set the player's gravity to 0 while they are holding the object so that the script can take control of their movement (until they let go of the object).



Ride on a moving platform

What it does: Grab the wand and point it in the direction you want the platform to go.

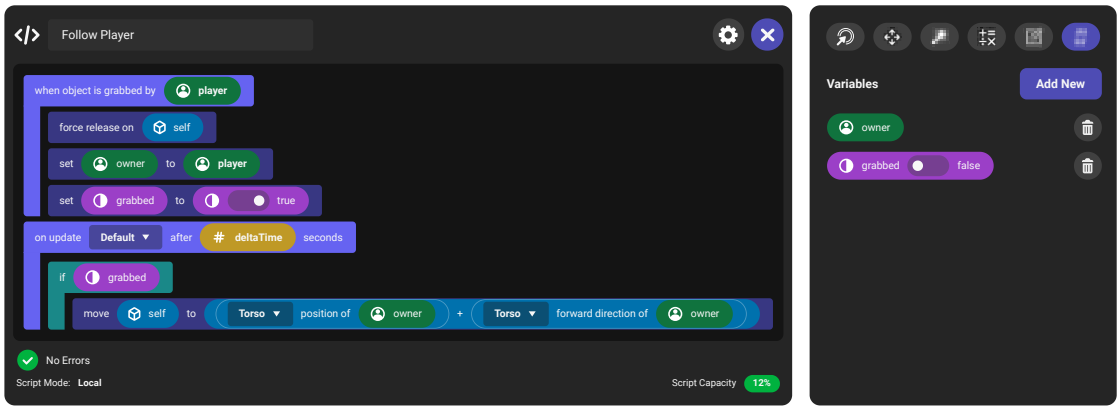
How it works: The PrePhysics update is used to move a player on a platform. In this case we want to move the platform and then have the player moved to match the platform. Since the player will be updated during the Physics update we need to move the platform before that, so we choose PrePhysics. If the Default update was used then we would move the platform after the player has already moved for that frame and the platform would always be one frame “ahead” of the player which will create jitter (because the player needs to “snap” back onto the platform every frame). PrePhysics updates the platform position before updating the player's position.


Steerable Platform
when world is started
transfer ownership of platform to owner of self
on update PrePhysics after deltaTime seconds
if holding
move platform by forward direction of self * deltaTime * 10
when object is grabbed by player
set holding to true
when object is released by player
set holding to false
No Errors
Script Mode: Local
Script Capacity 12%
Variables
Add New
platform
holding false

Follow Player

What it does: The object follows the player, always right in front of them, so that it is easy to reach and grab.

How it works: Default update is used to holster an object in front of a player. The Default update is used so that we can get the position the player is at this frame (which is computed after the PrePhysics update runs). If PrePhysics was used instead then we would be getting the torso position and direction from the previous frame and the holstered gun would lag behind the player by one frame. Default updates after player motion, allowing it to get the exact position of the player on that frame.



Follow Object

What it does: The *firstFollower* object follows *self*. The *secondFollower* follows the *firstFollower*.

How it works: FirstFollower is moving in an event loop, while SecondFollower is moving in a Default update event. Since Default events run after normal script events, this allows SecondFollower to move to the exact position of FirstFollower.

If SecondFollower was instead moving to the position of FirstFollower inside the event loop, it would lag behind by a frame. This is because the position of FirstFollower isn't updated until after the loop event completes.