Player Management: Identification

This guide explores the question *"who is this player?"*, including the specifics around: who performed an action, querying attributes of a player, and checking if two players are the same. The guide describes the details of the player id and player index systems.

Account Identifier

Every player in Horizon Worlds has a global account identifier that never changes (regardless of what world they are in or if they change their username . Horizon Worlds uses this identifier internally to manage travel, follow, mute, block, and more. It is not currently possible to query a player's account identifier, however it can be referenced indirectly using the persistence system.

Persistence

When you set or get a player persistent variable or manage achievements and leaderboards, Horizon Worlds uses account identifiers under the hood.

Example: a player earns access to a VIP lounge, which is stored in a player persistent variable; when that same player later returns to that world, even if in a different world instance, you can access that previous value that was set.

You cannot currently use player persistence to "imitate" global identifiers. Currently no notion of "world persistence", but if there were, you could create a counter that increases for every new player. But as November 2022 this is not possible.

Player Ids player

When a player travels to a world instance they are assigned a player id. Once that player id is assigned, no other player will receive that player id in that world instance. However, if that player leaves that world instance and later returns to that same world instance they will get another new player id.

There are two ways to get access to a player id: as a parameter in an event handlers and using a player index (described below).

Player-Related Event Handlers `when... player`

You can run scripts in response to player-related events via a large assortment of event handlers (the code blocks that start with the word "when"). There are events for when a player enters / exits a world instance, grabs / releases an object, enters / exits a trigger gizmo, presses a button on their controller, unlocks an achievement, and many more. For all of the available events, see the reference guide for the Events tab of the script UI.

Example Player Id Lifecycle

1. A player travels to a world instance and is assigned the player id "3".
2. All objects with an attached script containing the `when player enters the world` code block have that handler run with the player id "3" passed in.
3. Later on that player grabs an item. The code block `when object is grabbed by player` runs on that object, with the player id "3" passed in.
4. The player later leaves the world instance. The `when player exits the world` code block is run on all relevant scripted objects, with the player id "3" passed as the parameter.
5. The player later returns to the same world instance. They are assigned a new player id, perhaps "8".

Note: if one player id *equals* another player id then you are *certain* they refer to the same player. If the player ids are *not equal* then they are either 1) different player OR 2) the same player that left the world instance and then later returned.

When a player leaves a world instance the player id remains valid for a short amount of time, so be sure to use it immediately if you need to. For example, if you want to persist the number of minutes that the player was in the world instance then you should do so immediately in the `when player exits the world` handler.

Server Player server player

There is a special player id that no player is ever assigned. It represents an "omnipresent" and "imaginary" player represented by the Horizon Worlds server. This is the player id returned when the owner of object code block is passed an object owned by the server. You can access this player id directly with the server player code block. You can use that block, along with the transfer ownership of object to player block, to return ownership of an object to the server.

Note that the server player and an invalid/null player id are *not* the same thing (see the next section).

Invalid / Null Player Id

An invalid (or "null") player id is a player id that represents no one (including the omnipresent magical server player). There are a number of ways to end up with an invalid player id:

1. The player that the player id represented has exited the current world instance.
2. The player id is a script variable that has yet to be given a value (meaning the set to code block has not been used on it).
3. The player id came from querying for a player index that isn't currently in use (see below).

# 2 and #3 represent a "null" player id whereas #1 represents a "stale" player id . Comparing one null player id with another null will return true. Comparing a null with a stale will return false. Comparing a stale with a stale will return true only if they represented the same player, who has since left.

Player Index get player index / get player from index

The player index system works like a parking lot where everyone wants to park as close to the building as possible, taking the nearest *empty* space.

The rules of the player index system are as follows:

1. There are as many indices as the maximum number of players, numbered from 0 to one less than the maximum number of players. For example, if a world is set to allow a maximum of 6 players, then there are 6 player indices, numbered as 0, 1, 2, 3, 4, and 5.
2. When a player enters a world instance they are assigned the *lowest* available player index. For example, if two players travel to a new world instance, the first to arrive will be assigned index 0 and then the other will get index 1.
3. When a player exits a world instance the player index that they had becomes available again. For example, assume there are two players with indices 0 and 1 and that the player with player index 0 leaves. If another player enters then they will get player index 0, since it is available; the next player to arrive would get player index 2, since the player with player index 1 is still there.

You can query the current player index for a player with the `get player index` code block. If the player id is invalid then -1 is returned.

You can query which player has a specific player index with the `get player from index` code block. If no player currently has that index then an invalid player id is returned.

Player Data

The last way to identify a player is by the attributes of that player. Player persistent variables were mentioned above. This section briefly discusses the other data that is available.

Name

The player name comes from the underlying Meta account. The name is unique; no one else can have that same name. However it isn't *stable*; a person can change their name to something new at any time (though there is a limit on how often you can do so).

Pose Data

You can query the position and rotation of a player's head, torso, left hand, right hand, and feet. There is currently no way to query data about a player's finger poses, facial expression, emotes, or social data (such as their followers, block list, etc).

Abilities

There are code blocks for managing and querying per-player state, separate from player persistent variables such as who is able to grab a given object, who is able to see a given object, and more.