MULTIPLAYER SCRIPT INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/multiplayer

The `IMultiplayerObjectType` interface derives from IObjectClass to provide APIs specific to the Multiplayer object.

Note this class derives from the object class interface, not the instance interface. Typically it is accessed with `runtime.objects.Multiplayer`.

Designing multiplayer games is a complex topic. There is more documentation about the way the Multiplayer object works in the Multiplayer object manual entry. There is further learning material in the Online multiplayer in Construct tutorial series. This documentation covers only the scripting APIs to access the various multiplayer features Construct provides.

API organisation

The multiplayer APIs broadly fall in to two categories:

1. Signalling APIs, which make use of the signalling server to meet other peers and establish connections to them
2. The main multiplayer APIs, which are used for peer-to-peer communication once connections have been established.

To clearly separate these usages, signalling APIs are available on a dedicated signalling interface at `runtime.objects.Multiplayer.signalling`, whereas the remaining multiplayer APIs are available on the main multiplayer interface at `runtime.objects.Multiplayer`.

There are also some statistics under `runtime.objects.Multiplayer.stats`, but usage of those is optional.

Examples

See the Multiplayer scripting example for a demonstration of using these multiplayer APIs in JavaScript code.

Signalling events

These events are fired on `runtime.objects.Multiplayer.signalling`.

"connected"

Triggered when a connection to the signalling server has successfully been established. The event object includes the properties:

- `myId` : the peer ID assigned to the local user
- `serverVersion` : a string with the signalling server software version
- `serverName` : a string of the signalling server name
- `serverOperator` : a string identifying the signalling server operator
- `serverMOTD` : a "message of the day" string chosen by the server operator

"login"

Fired upon successfully logging in to the signalling server. The event object `myAlias` property provides the alias assigned to the local user (which will be the alias requested, unless the alias is already in use, in which case the signalling server will have assigned a different alias derived from the requested one).

"join"

Fired upon successfully joining a room on the signalling server. The event object includes the properties:

- `isHost` : a boolean indicating whether the local user is the host of the room. The first peer to join a room is assigned the host.
- `hostId` : the peer ID of the room's host. This only needs to be referred to when `isHost` is false (since if the local user is the host, the host ID is their own peer ID).
- `hostAlias` : the alias of the room's host.
- `room` : a string of the room name that was joined. (This can only be different to the requested room name when auto-joining a room.)

"leave"

Fired upon successfully leaving a room on the signalling server.

"disconnected"

Fired when lost connection to the signalling server.

"kicked"

Fired when forcibly removed from the current room on the signalling server. This is similar to the `"leave"` event, but the cause of leaving the room was server-initiated rather than peer-initiated.

"error"

Fired if an error occurs while using the signalling server. The event object `message` property provides an error message (if any).

Signalling APIs

These methods are available on `runtime.objects.Multiplayer.signalling`.

addEventListener(eventName, callback)

removeEventListener(eventName, callback)

Add or remove a callback function for a signalling event. See *Signalling events* above for more information.

async connect(url = "wss://multiplayer.construct.net")

Initiate a connection to a signalling server. The default URL is the official Scirra-hosted signalling server. The method can be awaited and resolves at the same time the signalling `"connected"` event fires, resolving with an object with the same properties as the event object.

disconnect()

Disconnect from the signalling server. This can be done once peer-to-peer connections are established if the signalling server is no longer necessary, but note that will prevent any new peers from joining the game late.

isConnected

A read-only boolean indicating whether a connection to the signalling server is currently active.

addICEServer(url, username, credential)

Add a custom Interactive Connectivity Establishment (ICE) server used by WebRTC to establish connections between peers. These can include STUN and TURN servers. A username and credential can also be optionally provided if the server requires them. This method should be called on startup, before any connections are made.

async login(alias)

Attempt to log in to the signalling server and request to use the provided alias. The method can be awaited and resolves at the same time as the signalling `"login"` event, resolving with an object with the same properties as the event object.

isLoggedIn

A boolean indicating whether the user is currently logged in on the signalling server.

async joinRoom(game, instance, room, maxPeers = 0)

Join a specific room in the given game instance. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. `maxPeers` can be used to limit the number of peers that join. Only the host's value is used. If the room is full, subsequently joining peers will receive a "room full" error. The peer count includes the host, so 2 is the minimum value, or it can be left as 0 to allow an unlimited number of peers to join. The method can be awaited and resolves at the same time as the signalling `"join"` event, resolving with an object with the same properties as the event object.

async autoJoinRoom(game, instance, room, maxPeers = 2, isLocking = true)

Join the first available room with the given game, instance and first room name. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. When rooms are full, the signalling server will create a new room. For example if "myroom" is full, it will try "myroom2", "myroom3", etc. This effectively arranges all joining peers in to games of a particular size. If `isLocking` is true, then the room is locked when full. In that case late-joiners are not allowed; if left unlocked and a peer leaves after the game starts, a newly joining peer may be added back to the game to top it up to `maxPeers` again. This method works similarly to `joinRoom()` in that it can be awaited and resolves when the `"join"` event fires.

async leaveRoom()

Request to leave the current room on the signalling server. This method can be awaited and resolves at the same time the signalling `"leave"` event fires.

async requestGameInstanceList(game)

Request a list of active game instances within the given game. A promise is returned which resolves when the response is received with an array of objects describing each game instance, with the object properties:

- `name`: the game instance name

- `peerCount`: the total number of peers in that game instance

---

async requestRoomList(game, instance, type = "all")

Request a list of active rooms within a given game instance. The returned rooms depends on the `type`: `"all"` includes all rooms; `"unlocked"` includes only rooms which are unlocked; and `"available"` includes only rooms which are available to join (unlocked and not full). A promise is returned which resolves when the response is received with an array of objects describing each room, with the object properties:

- `name`: the room name
- `peerCount`: the number of peers in the room
- `maxPeerCount`: the maximum number of peers allowed in the room, or 0 for unlimited
- `state`: the room state, one of `"available"`, `"locked"` or `"full"`.

Multiplayer events

These events are fired on `runtime.objects.Multiplayer`.

"peerconnect"

Fired when a peer joins the same room. It also fires once per peer already in the room when joining an existing room, including the host. The event object includes the properties:

- `peerId`: the ID of the connected peer
- `peerAlias`: the alias of the connected peer

"peerdisconnect"

Fired when an existing peer disconnects from the room. The event object includes the properties:

- `peerId`: the ID of the disconnected peer
- `peerAlias`: the alias of the disconnected peer
- `leaveReason`: a string with an optional reason provided for the peer disconnecting

---

"message"

Fired when a message is received over the network. Note the order messages are received, or whether a sent message is received at all, depends on the reliability mode chosen when the message was originally sent. The event object includes the properties:

- `fromId` : the ID of the peer the message was sent by
- `fromAlias` : the alias of the peer the message was sent by
- `message` : the content of the message. This is either a string, JSON data, or an ArrayBuffer for binary content, depending on the type of the message sent.
- `transmissionMode` : the transmission mode the message was sent with.

"kicked"

Fired if kicked from the current room. This can occur if the host quits, the connection to the host could not be established, or the host otherwise decides to forcibly remove you from the room. After this fires the player is no longer in the room and must re-join a room to be able to participate in a game.

Multiplayer APIs

These methods and properties are available on `runtime.objects.Multiplayer` .

signalling

Provides the signalling interface - see *Signalling APIs* above.

stats

Provides the statistics interface - see *Statistics APIs* below.

isHost

A read-only boolean indicating if the current peer is the room host.

myId
myAlias

Read-only strings with the peer ID and alias of the local user.

hostId
hostAlias

Read-only strings with the peer ID and alias of the room host (which will be the same as `myId` and `myAlias` if `isHost` is true).

currentGame

currentGameInstance

currentRoom

Read-only strings identifying the current game, game instance, and room.

peerCount

Read-only number of connected peers, including the local user.

getAllPeers()

Return an array of `IMultiplayerPeer` representing every peer in the room, including the local user. See *Peer APIs* below.

getPeerById(peerId)

Return a `IMultiplayerPeer` for a peer in the current room by their peer ID, or returns `null` if they don't exist. See also *Peer APIs* below.

sendPeerMessage(peerId, message, transmissionMode = "o")

Send a message over the network to a peer in the same room identified by their peer ID. The message can be a string, an object for JSON transmission (which must be convertible to a string), or an ArrayBuffer for binary content. The transmission mode can be one of `"o"` for reliable ordered, `"r"` for reliable unordered, or `"u"` for unreliable (see the Multiplayer object documentation for more details about reliability modes). When received the `"message"` event will be fired.

hostBroadcastMessage(fromId, message, transmissionMode = "o")

This is similar to `sendPeerMessage()` but can only be called by the host, and the provided message will be sent to every other peer in the room. `fromId` can optionally be set to another peer ID to make it appear that the message is from that peer, which is useful when relaying messages through the host; if left empty it will use the host ID.

disconnectRoom()

Disconnects from any peers in the current room and also leaves the room on the signalling server. If the current user is the room host, all other peers are kicked.

simulateLatency(latency, pdv, loss)

Simulate latency, PDV and packet loss on all inbound and outbound messages. This can be useful for making local testing more realistic, since unlike the Internet latency is effectively non-existent. For local testing it is only necessary to simulate latency on the host, since that guarantees every message in the game will have delay added; it is not necessary to also simulate latency on the peers. The latency for an individual message is calculated as the latency plus a random value from zero to the PDV. The packet loss indicates the chance an unreliable message is lost entirely, or in the case of reliable messages that retransmission is necessary and the latency is multiplied.

Peer APIs

The `IMultiplayerPeer` interface represents a connected peer in the same room. It is returned by methods like `getAllPeers()` and `getPeerById()`.

id

The peer ID for this peer.

alias

The alias for this peer.

isHost

A boolean indicating if this peer is the room host.

isMe

A boolean indicating if this peer represents the local user.

latency

pdv

Get the measured latency and packet delay variation (PDV) on the network connection to this peer. Note peers can only use this to get the stats for the host, since that is the only connection they have, but the host can use it for any peer.

send(message, transmissionMode = "o")

This is a shorthand for calling `sendPeerMessage()` with this peer's ID.

Statistics APIs

These properties are available under `runtime.objects.Multiplayer.stats`.

inboundBandwidth

inboundBandwidth

outboundBandwidth

Read-only numbers with the total estimated inbound and outbound bandwidth for all data transmission over the network in bytes per second. When automatic data compression is in use, this measures the compressed size of the data actually sent over the network.

inboundDecompressedBandwidth

outboundDecompressedBandwidth

Read-only numbers with the total estimated decompressed inbound and outbound bandwidth for all data sent and received via the Multiplayer object in bytes per second. When automatic data compression is in use, this measures the size of the decompressed messages, which may be significantly larger than the data actually sent over the network.

inboundCount

outboundCount

Read-only numbers of the total number of separate inbound and outbound messages sent and received. This includes internally-used messages for things like ping and synchronisation; generally the bandwidth is the more practically useful statistic.