N7 PROGRAMMING LANGUAGE EXTENSION

P2P Library – API Reference

Introduction

This library allows you to build “serverless” online multiplayer games (meaning no dedicated external game server is required). One player creates a lobby and acts as the host/server, while other players connect as clients. All network traffic is automatically relayed through the host behind the scenes.

Lists of active game lobbies can optionally be published to and fetched from NaaLaa's server, including public and local IP addresses so clients can discover and join games.

> **Note on Hosting & UPnP:** When creating a lobby (`P2P_CreateLobby`), the library automatically attempts to open the required port on the host's router using **UPnP** (Universal Plug and Play) so that clients over the internet can connect. This works out of the box on most home routers, though the host may occasionally need to enable UPnP in their router settings if external players are unable to connect. Clients joining a game do not need UPnP or port forwarding enabled.

Lifecycle & Connection Management

```
(number) P2P_Init()
```

Initializes the P2P library. Must be called before any other P2P function.

- **Returns:** `true` on success, `false` otherwise.

```
P2P_Terminate()
```

Shuts down and cleans up the P2P library. Optional to call manually, as the extension automatically cleans up on program exit.

```
(number) P2P_CreateLobby((string) game, (string) user, (number) extra, (number) public)
```

Creates a lobby and optionally registers it on NaaLaa's server. Listed lobbies are automatically removed upon timeout or when calling `P2P_Disconnect` .

- `game` – Identifier for your game/application (choose something unique).
- `user` – Identifier or display name for the host user.
- `extra` – Integer flag or metadata value (e.g., player count or a “full” flag), visible to clients via `P2P_FindLobbies` when `public` is enabled. Can be updated later with `P2P_SetLobbyExtra` .
- `public` – Set to `true` to publish the lobby to NaaLaa's server so other clients can find it.
- **Returns:** `true` on success, `false` otherwise.

```
P2P_SetLobbyExtra((number) extra)
```

Updates the extra value stored on NaaLaa's server for a lobby created with `public = true`.

- `extra` – New integer value/flag for the lobby.

```
(array) P2P_FindLobbies((string) game)
```

Fetches active lobbies for the specified game identifier from NaaLaa's server. Returns an empty array if no lobbies are found.

- `game` – Game identifier passed to `P2P_CreateLobby`.
- **Returns:** An array of lobby objects, each containing:
  - `ip` – Public IP address (standard target for `P2P_Connect`).
  - `pip` – Local/private IP address (useful for LAN testing).
  - `user` – Name of the user hosting the lobby.
  - `extra` – Extra value/flag set by the host.

```
(number) P2P_Connect((string) server, (string) user)
```

Connects a client to a host at the specified IP address. Note that this call blocks execution for a few seconds while connecting.

- `server` – IP address of the host/server to connect to.
- `user` – Name or identifier of the connecting client.
- **Returns:** `true` on success, `false` on failure.

```
P2P_Disconnect()
```

Closes the current connection.

- **As a client:** Disconnects from the server. The server and remaining clients receive a `P2P_USER_DISCONNECTED` message via `P2P_Receive` with your unique user ID in `usr`.
- **As a server:** Closes the lobby/session, causing connected clients to receive a `P2P_CONNECTION_LOST` message.

```
P2P_Kick((number) usr)
```

Force-disconnects a client from the lobby. Only works when called by the server.

- `usr` – Unique user ID of the client to kick.

Client Discovery (Server Mode)

```
(number) P2P_SearchClient()
```

Starts listening for an incoming client connection when acting as the server. Poll `P2P_Receive` afterward to handle events. When a client connects, you receive a `P2P_USER_CONNECTED` message with the username in `msg` and the client's unique integer ID in `usr`. Call `P2P_SearchClient` again if you want to accept additional clients.

- **Returns:** `true` on success, `false` otherwise.

```
P2P_CancelSearch()
```

Cancels an active client search started with `P2P_SearchClient`.

```
(number) P2P_IsSearching()
```

Checks whether the server is currently searching for an incoming client.

- **Returns:** `true` if actively searching, `false` otherwise.

Main Loop & Event Polling

```
P2P_Update()
```

Updates network state and message handling. Must be called once per frame/tick (typically 60 times per second) in your main loop once a lobby is created or joined; otherwise `P2P_Send`, `P2P_Receive`, and related functions will not work.

```
(number) P2P_Receive((table) pkg)
```

Polls for incoming messages and network events. Call after `P2P_Update`.

- `pkg` – Table/object populated with message data when a message is available:
  - `id` – Message identifier (custom ID `1..100` or a built-in event constant).
  - `usr` – Unique non-zero ID of the user who sent the message.
  - `msg` – Message payload/text.
- **Returns:** `true` if a message was received, `false` otherwise.

Predefined Event ID	Description
`P2P_USER_CONNECTED`	A client connected (`msg` contains the username).
`P2P_USER_DISCONNECTED`	A client disconnected (`msg` contains the username).
`P2P_CONNECTION_LOST`	Connection to the server was lost.

Sending Messages

```
P2P_Send((number) id, (string) msg)
```

Broadcasts a text message to all other connected peers (server/clients).

- `id` – Message type identifier in the range `[1..100]`.
- `msg` – Message string to send.

```
P2P_SendTo((number) rcv, (number) id, (string) msg)
```

Sends a text message to a specific user.

- `rcv` – Target user's unique ID.
- `id` – Message type identifier in the range `[1..100]`.
- `msg` – Message string to send.

```
(number) P2P_Buffer((number) usr)
```

Returns the available send-buffer space (in bytes).

- **In client mode:** Only one buffer exists (for sending to the server, which relays to other clients). Pass `0` for `usr`.
- **In server mode:** Each connected client has its own send buffer. Pass the target client's ID as `usr`.
- **Returns:** Available buffer space in bytes.

Structured Packages (Multi-Value Messages)

Use these functions when sending messages composed of multiple typed values (such as a player's `x` and `y` coordinates in a 2D game).

```
P2P_ComposePackage((number) id)
```

Begins building a structured message package.

- `id` – Message type identifier.

```
P2P_AddInt((number) value)
```

Appends an integer value to the package currently being composed.

```
P2P_AddFloat((number) value)
```

Appends a floating-point value to the package currently being composed.

```
P2P_AddString((string) value)
```

Appends a string value to the package currently being composed.

```
P2P_SendComposed()
```

Finalizes and broadcasts the composed package to all connected users.

```
P2P_SendComposedTo((number) rcv)
```

Finalizes and sends the composed package to a specific user.

- `rcv` – Target user's unique ID.

```
P2P_Decompose((string) msg)
```

Prepares a received complex message (`pkg.msg` from `P2P_Receive`) for reading. Call `P2P_GetInt`, `P2P_GetFloat`, and `P2P_GetString` in the exact same order the fields were added when composing the package.

```
(number) P2P_GetInt()
```

Reads and returns the next integer value from the decomposed package.

```
(number) P2P_GetFloat()
```

Reads and returns the next floating-point value from the decomposed package.

```
(string) P2P_GetString()
```

Reads and returns the next string value from the decomposed package.