

# Project Documentation: C# & HTML Bi-Directional Bridge

---

This document explains how to create a real-time communication link between a C# application (like a GTA V Mod) and an external HTML user interface using a self-hosted HTTP server.

**Prerequisites:** Visual Studio (C#), ScriptHookV .NET, NUllight v0.2 and basic knowledge of JavaScript `fetch()` API.

## 1. The Architecture

Since the game and the browser are separate processes, we use the **HTTP Protocol**. The C# script acts as a "Web Server" (Listener), and the HTML page acts as the "Client" (Browser).

## 2. C# Implementation: The Server

We use `System.Net.HttpListener` to host a server inside the game. This avoids the need for external tools like Node.js.

### Key Functions:

- **Initialize:** Start the listener on a specific port (e.g., 3000).
- **HandleRequests:** An asynchronous loop that waits for the browser to "ask" for data.
- **Cleanup:** Closing the port when the script stops to prevent "Address already in use" errors.

```
// C# Server Logic (Snippet)
private async Task ListenForRequests() {
    while (!_listener.IsListening) {
        var context = await _listener.GetContextAsync();
        var request = context.Request;
        var response = context.Response;

        // ENABLE CORS (Allow browser to talk to us)
        response.Headers.Add("Access-Control-Allow-Origin", "*");

        if (request.Method == "POST") {
            // RECEIVING DATA FROM HTML
            using (var reader = new StreamReader(request.InputStream)) {
                string body = await reader.ReadToEndAsync();
                // Process the body (e.g., Parse JSON)
            }
        } else {
            // SENDING DATA TO HTML
            string json = "{\"status\": \"success\"}";
            byte[] buffer = Encoding.UTF8.GetBytes(json);
            await response.OutputStream.WriteAsync(buffer, 0, buffer.Length);
        }
        response.Close();
    }
}
```

### 3. HTML Implementation: The Client

The browser uses the `fetch()` API to communicate. To keep the UI updated, we use `setInterval()` or `requestAnimationFrame()`.

## A. Receiving Data (GET)

```
async function getData() {  
    const response = await fetch('http://localhost:3000/data');  
    const data = await response.json();  
    console.log("Data from Game:", data.money);  
}
```

## B. Sending Data (POST)

To send data back to the game (e.g., clicking a button in HTML to give the player money), use the `POST` method:

```
async function sendAction(actionType) {  
    await fetch('http://localhost:3000/data', {  
        method: 'POST',  
        headers: { 'Content-Type': 'application/json' },  
        body: JSON.stringify({ action: actionType, value: 100 })  
    });  
}
```

# 4. Troubleshooting & Best Practices

### Issue

### Solution

ERR_CONNECTION_REFUSED	Server isn't running or the port is blocked by Windows Firewall.
------------------------	--

---

CORS Policy Error

Missing `Access-Control-Allow-Origin` header in C# code.

---

UI Stuttering

Use CSS transitions or `requestAnimationFrame` for smooth number counting.

---

**Tip :** Ensure your HTML `body` has `background-color: rgba(0,0,0,0);` for full transparency in game.

Documentation for C# / HTML Integration Project © 2026 by ShadowOne.