## Contents

## 1 CGI VS Server Side Scripting

CGI is an abbreviation of Common Gateway Interface, a specification for transferring information between a World Wide Web server and a CGI program. A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic. CGI programs are the most common way for Web servers to interact dynamically with users. Many HTML pages that contain forms, for example, use a CGI program to process the form's data once it's submitted. Another increasingly common way to provide dynamic feedback for Web users is to include scripts or programs that run on the user's machine rather than the Web server. These programs can be Java applets, Java scripts, or ActiveX controls. These technologies are known collectively as client-side solutions, while the use of CGI is a server-side solution because the processing occurs on the Web server.
CGI, on the other hand, is not a programming language but rather an interface to the system which allows for an interface that comes to the system using a script or executable file of which may be C, C++, Perl, and similar languages. So it can actually allow to run server side scripting language, however in an inefficient manner (starting a new process for each request).

## 2 HTTP referer vs location

The HTTP referer (a misspelling of referrer[1]) is an optional HTTP header field that identifies the address of the webpage (i.e. the URI or IRI) which is linked to the resource being requested. By checking the referrer, the new webpage can see where the request originated. The HTTP Location header field is returned in responses from an HTTP server under two circumstances:

1. To ask a web browser to load a different web page (URL redirection). In this circumstance, the Location header should be sent with an HTTP status code of 3xx. It is passed as part of the response by a web server when the requested URI has:

   - Moved temporarily;
   - Moved permanently; or

2. Processed a request, e.g. a POSTed form, and is providing the result of that request at a different URI To provide information about the location of a newly created resource. In this circumstance, the Location header should be sent with an HTTP status code of 201 or 202.[1

## 3 Microsoft RPC vs XML RPC

Microsoft RPC is a modified version of DCE/RPC. Additions include partial support for UCS-2 strings, implicit handles, and complex calculations in the variable-length string and structure paradigms already present in DCE/RPC. XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism. In XML-RPC, a client performs an RPC by sending an HTTP request to a server that implements XML-RPC and receives the HTTP response. A call can have multiple parameters and one result. The protocol defines a few data types for the parameters and result. Some of these data types are complex, i.e. nested. For example, you can have a parameter that is an array of five integers.

## 4 Why would one need an IDL?

An interface definition language (IDL) is used to set up communications between clients and servers in remote procedure calls (RPC). There have been many variations of this such as Sun RPC, ONC RPC, DCE RPC and so on. Basically, you use an IDL to specify the interface between client and server so that the RPC mechanism can create the code stubs required to call functions across the network. RPC needs to create stub functions for the client and a server, using the IDL information.

## 5 What is a binding?

Binding is the process of mapping a name to an address. As an example of naming and binding, we can consider the machine names on the Internet. A name of cs.rutgers.edu may bind to the IP address of 128.6.4.2. In turn, the IP address 128.6.4.2 may bind to the Ethernet address 08:00:20:1f:13:83. The least flexible binding is static binding. This is essentially a hard-coded binding. For example, a program may assume that SMTP mail service is always available on port 25 and simply access port 25 instead of attempting to resolve the binding through other means An alternative to static binding is dynamic binding. One form of dynamic binding is early binding. In this case a binding operation is actually performed, but it is performed some time before the binding is needed. For example, if a program needs to contact a server multiple times during a long period of execution, it might perform a name to IP address binding once at the start for efficiency.

## 6 Websocket Squares Example

### 6.1 Client 1

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=UTF
        -8"/>
    <title>SEND</title>
  </head>
  <body>
    <h1>Square Send</h1><hr/>
    <table id="sqr">
```

```
      <tr><td><input type="button" value="x"/></td><td><input type="button"
         value="x"/></td>
        <td><input type="button" value="x"/></td><td><input type="button" value=
           "x"/></td>
        <td><input type="button" value="x"/></td></tr>
      <tr><td><input type="button" value="x"/></td><td><input type="button"
         value="x"/></td>
        <td><input type="button" value="x"/></td><td><input type="button" value=
           "x"/></td>
        <td><input type="button" value="x"/></td></tr>
      <tr><td><input type="button" value="x"/></td><td><input type="button"
         value="x"/></td>
        <td><input type="button" value="x"/></td><td><input type="button" value=
           "x"/></td>
        <td><input type="button" value="x"/></td></tr>
      <tr><td><input type="button" value="x"/></td><td><input type="button"
         value="x"/></td>
        <td><input type="button" value="x"/></td><td><input type="button" value=
           "x"/></td>
        <td><input type="button" value="x"/></td></tr>
      <tr><td><input type="button" value="x"/></td><td><input type="button"
         value="x"/></td>
        <td><input type="button" value="x"/></td><td><input type="button" value=
           "x"/></td>
        <td><input type="button" value="x"/></td></tr>
    </table>
    <script type="text/javascript" src="square_send.js"></script>
  </body>
</html>
```

```
const socket = new WebSocket("ws:localhost:8765");

const sendField = fieldIndex => {
  if (socket.readyState === 1) {
    socket.send(fieldIndex);
  }
};

window.onbeforeunload = () => {
  console.log("closing ws");
  socket.close();
};

const bts = document.querySelectorAll("input");

bts.forEach((bt, i) => {
  bt.dataset.idx = i;
  bt.addEventListener("click", e => {
    sendField(bt.dataset.idx);
  });
});
```

### 6.2 Client 2

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="application/xhtml+xml;␣charset=UTF
        -8"/>
    <title>RECV</title>
    <link rel="stylesheet" type="text/css" href="bordertable.css"/>
    <style type="text/css">
      .blue {background-color: #0000ff;}
      .white {background-color: #ffffff;}
    </style>
  </head>
  <body>
    <h1>Square Receive</h1><hr/>
    <table id="sqr">
      <tr><td/><td/><td/><td/><td/></tr>
      <tr><td/><td/><td/><td/><td/></tr>
      <tr><td/><td/><td/><td/><td/></tr>
      <tr><td/><td/><td/><td/><td/></tr>
      <tr><td/><td/><td/><td/><td/></tr>
    </table>
    <script type="text/javascript" src="square_recv.js"></script>
  </body>
</html>
```

```
const socket = new WebSocket("ws://localhost:8765");

window.onbeforeunload = () => {
  socket.close();
};

const tds = document.querySelectorAll("td");
const colorTd = fieldIndex => {
  tds.forEach((td, idx) => {
    if (fieldIndex == idx) {
      td.classList.remove("white");
      td.classList.add("blue");
    } else {
      td.classList.remove("blue");
      td.classList.add("white");
    }
  });
};

socket.onmessage = evt => {
  colorTd(evt.data);
};
```

### 6.3 Server

```
import asyncio
import websockets
```

```
sockets = []

async def handler(websocket, path):
    sockets.append(websocket)
    async for message in websocket:
        for socket in sockets:
            await socket.send(message)


start_server = websockets.serve(handler, 'localhost', 8765)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

## 7 Guestbook Example

```
<h1>Guestbook</h1>
<ul>
</ul>
<hr />
<form action="https://vsr.informatik.tu-chemnitz.de/edu/2015/evs/exercises/
    jsajax/guestbook.php" onsubmit="event.preventDefault();␣sendData();" method="
    POST">
  <label for="name">Name</label>
  <input id="name" name="name" type="text" placeholder="Name" />
  <br />
  <label for="text">Text</label>
  <input id="text" name="text" type="text" placeholder="Text" />
  <br />
  <button type="submit">Add entry</button>
</form>

<script>
  const baseUrl = "https://vsr.informatik.tu-chemnitz.de/edu/2015/evs/exercises/
     jsajax/guestbook.php";

  function loadData() {
    const xhr = new XMLHttpRequest();
    xhr.addEventListener("load", displayData);
    xhr.open("GET", baseUrl);
    xhr.send();
  }

  function displayData() {
    res = JSON.parse(this.responseText);
    const ul = document.getElementsByTagName("ul")[0];

    res.forEach((entry) => {
      addEntry(ul, entry);
    });
  }
```

```
function addEntry(ul, entry) {
  let li = document.createElement("li");
  li.id = entry.id;
  li.innerHTML = `<strong>${entry.name}:</strong> ${entry.text} `;
  let a = document.createElement("a");
  a.setAttribute("href", entry.id);
  a.textContent = "(X)";
  li.appendChild(a);
  ul.appendChild(li);
  a.addEventListener("click", function (e) {
    e.preventDefault();
    e.stopPropagation();
    deleteEntry(this.getAttribute("href"));
  });
}

function deleteEntry(id) {
  const xhr = new XMLHttpRequest();
  xhr.open("DELETE", `${baseUrl}?id=${id}`);
  xhr.send();
  xhr.onreadystatechange = function (e) {
    if (xhr.readyState == 4 && xhr.status == 200) {
      res = JSON.parse(this.responseText);
      if (res.message) {
        li = document.getElementById(id);
        li.parentNode.removeChild(li);
      }
    }
  }
}

loadData();

function sendData() {
  const xhr = new XMLHttpRequest();
  const nameEl = document.getElementById("name");
  const textEl = document.getElementById("text");
  const name = nameEl.value;
  const text = textEl.value;
  const params = `name=${name}&text=${text}`;
  nameEl.value = "";
  textEl.value = "";
  xhr.open("POST", baseUrl);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send(params);
  xhr.onreadystatechange = function (e) {
    if (this.readyState == 4 && this.status == 200) {
      res = JSON.parse(this.responseText);
      const ul = document.getElementsByTagName("ul")[0];
      addEntry(ul, res.entry);
    }
    }
  }
</script>
```