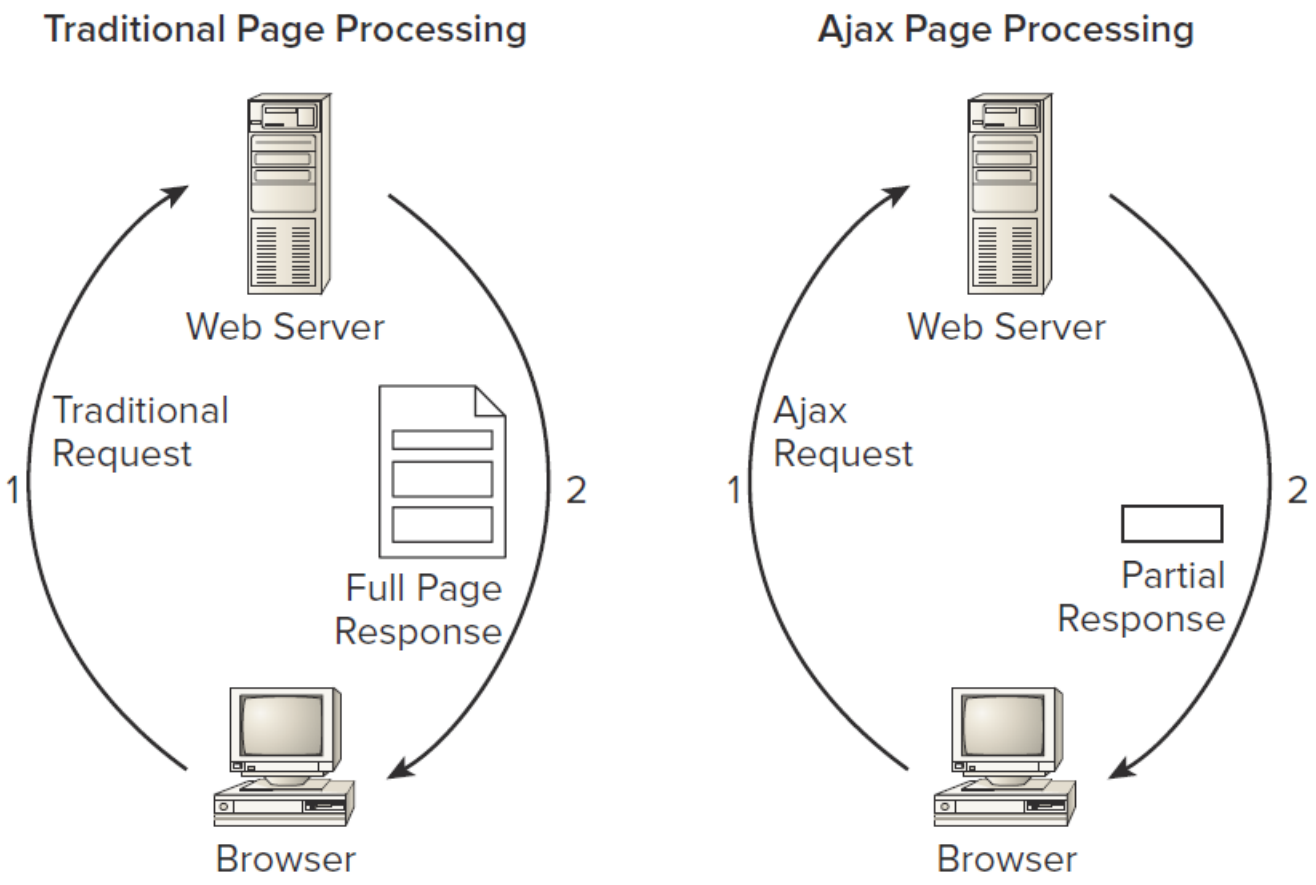
ASP.NET AJAX

Asst. Prof. Dr. Özgü Can

ASP.NET AJAX

- Asynchronous JavaScript And XML
- Enables *client-side* web pages to exchange data with the *server* through asynchronous calls.
- **Flicker-free page**
  - Enables to perform a postback to the server without refreshing the entire page.
- Gives more server controls to create *rich*, *interactive*, and *responsive* user interfaces.

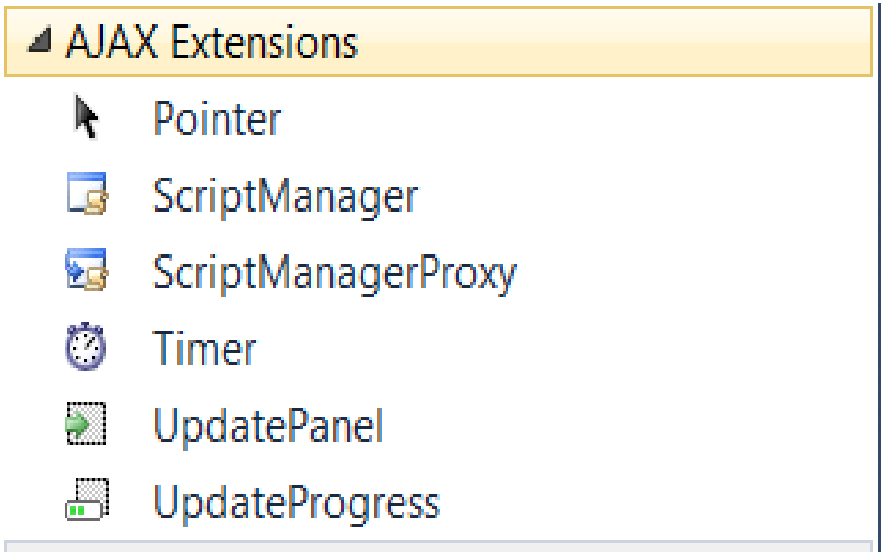
AJAX


Traditional Page Processing
Web Server
Traditional
Request
1
2
Full Page
Response
Browser
Ajax Page Processing
Web Server
Ajax
Request
1
2
Partial
Response
Browser

ASP.NET AJAX

- Enables you to:
  - Create flicker-free pages that enable you to refresh portions of the page without a full reload and without affecting other parts of the page.
  - Provide feedback to your users *during* these page refreshes.
  - Update sections of a page and call server-side code on a scheduled basis *using* a **timer**.
  - Access server-side web services and page methods and work *with* the **data** they return.
  - Use the rich, client-side programming framework to access and modify elements in your page, and get access to a code model and type system that looks similar to that of the .NET Framework.

AJAX Extensions



Creating Flicker-Free Pages

- `UpdatePanel` → To **avoid** full postbacks in ASPX pages and **update** only part of the page.
- ***Whenever*** one of the controls ***within*** the `UpdatePanel` causes a postback to the **server**, only the **content** ***within*** that `UpdatePanel` is **refreshed**.
- For this control to operate correctly;
  - `ScriptManager` → bridge between the client page and the server

ScriptManager

- ***Place*** the `ScriptManager` in the **master** page
  - if you're going to use Ajax functionality in many of your ASPX pages.
    - It's available in all pages that are based on this master.
- You can only have **one** ScriptManager per page
  - ***if*** you add **one** to a **master** page, you **can't** add another one to a **content** page.
- `ScriptManagerProxy` → In order to access a `ScriptManager` control that is defined in a **master** page ***from*** a **content** page.

UpdateProgress

- Postback's advantage → the user can see *something is happening*
- `UpdatePanel` → Users have ***no visual cue*** that something is happening until it has happened.
- `UpdateProgress`
  - Tells users to hold on for a few seconds while their request is being processed.

UpdateProgress

- Connect the `UpdateProgress` control to an `UpdatePanel`:
  - `AssociatedUpdatePanelID` property
    - text such as **"Please wait"** or an animated image to let the user know something is happening

Timer

- Great for executing server-side code on a repetitive basis.
  - *For example*, you can use it to update the contents of an `UpdatePanel` every 5 seconds.
- For more information check out MSDN's web page → http://tinyurl.com/TimerClass

Web Services

- Methods that you can call over the Internet and that can optionally return data to the calling code.
- Ideal for exchanging data between *between* **different** types of ***platforms*** and ***systems***.
  - For example;
    - exchange data between an **ASP.NET web site** *running on* **Microsoft Windows** and a **PHP-based site** *running on* **Linux**.
    - exchange data between an **ASP.NET web site** and a **client browser** *using* **JavaScript**.

Web Services

- To expose a method as a service:
  - `WebMethod` attribute
- An ***attribute*** is like a little tag or label that you can stick on code elements, like methods, properties, and so on, to **mark *that piece of code as something special***.

```
[WebMethod]
public string HelloWorld() {
    return "Hello World";
}
```

Signal to the ASP.NET runtime that you really want to expose this method as a web method that can be **called *from* client-side** code.

Web Services

- This also enables you to create other methods in the same class that are **not** exposed as **web services** automatically, giving you *flexibility in determining what to open up for the outside world*.
- Place this method in a file with an `.asmx` extension and inside a class that inherits from `System.Web.Services.WebService`.

Web Services

- You may want your **client-side** pages to **talk** to a **web service on a different domain**.
  - You need to set up security in the browser to allow this.
- You can also use **web services** to **have two servers** *or* **other applications communicate with each other**. In that case, one application interacts with an ASP.NET web service over the network to exchange data.
- *Outside the scope of this course!*

Configuring the Web Service

- To make a *web service* **visible** *by* **client-side script**:
  - at the `NameService` class in the `App_Code` folder, you see that the template already added the attribute for you, but commented it out:

```
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
```

- Uncomment the line to expose the entire service as a client-script service.

Configuring the ScriptManager

- A required component in almost all Ajax-related operations.
- To expose your web service to client script:
  - In the `ScriptManager` in the **master page**

  or

  - In a **content page** that uses the web service, using the `ScriptManagerProxy` class

Configuring the ScriptManager

- In the `ScriptManager` in the **master page**
- Give the `ScriptManager` control a `<Services>` element that in turn contains one or more `ServiceReference` elements that point to your public services.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />
  </Services>
</asp:ScriptManager>
```

Configuring the ScriptManager

- On a normal page that **doesn't** use a master page with a `ScriptManager` you can simply **add** a `ScriptManager` to the Web Form **directly**.
- *However*, if you are **using** a **master page** that already has its **own** `ScriptManager` you **need** to use a `ScriptManagerProxy` control.

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />
  </Services>
</asp:ScriptManagerProxy>
```

Page Methods

- Similar to web services.
- What's **different** is that page methods are defined directly in an ***existing*** ASPX page **instead of** a ***separate*** ASMX service file.
- You can only call them from script running within that page.
- Ideal for small, simple functionality that is limited in scope to the current page.

Page Methods

- To enable page methods you need to set the property `EnablePageMethods` of the `ScriptManager` control to **`True`**.

Page Methods

- Setting them up and using them is a twostep process:
  1. Create a `public` and `static` method in the Code Behind of the page you're working with. You need to apply the `[WebMethod]` attribute to this method. The method can optionally receive data through its parameters and optionally return some data.
  2. Write the necessary JavaScript to call the page method and work with its result.

ASP.NET AJAX

- For more information please check out MSDN's web site:

  http://tinyurl.com/AboutAjax