

Lesson 3-7: Understand Request and Response

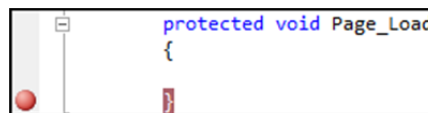
On 'plain' HTML web pages, a web browser sends a request to a web server and the web server responds by sending back the page's HTML code.

ASP.NET works in-between receiving the request from the browser and sending the response back to them, and you can see this in action by inspecting the *Page.Request* and *Page.Response* objects.

- 1 Open *CSharpTest* from your sample files folder.
- 2 Open *requestresponse.aspx.cs* (The code-behind file of *requestresponse.aspx*).
- 3 Add a breakpoint to the *Page_Load* event handler.

You might be wondering how to add a breakpoint to an event handler that doesn't have any code in it; if you try to set a breakpoint on a blank line, it won't let you.

Instead, set the breakpoint on the line with the } sign on it.

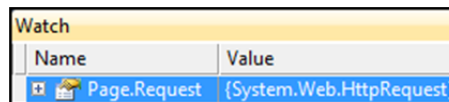


- 4 Run *requestresponse.aspx* in Debug mode.

Your code should be paused almost immediately, as the breakpoint is reached as soon as the page loads.

- 5 Clear any existing watches and add a watch for *Page.Request*.

This was covered in: *Lesson 3-6: Understand the Page object*.



- 6 Inspect *Page.Request*.

The *Page.Request* object contains all of the information that was sent by the visitor to the page.

1. Expand *Page.Request* and find *UserAgent* in the properties list.

The *UserAgent* property tells you which browser the visitor is using.

Url	{http://localhost:50085/requestresponse.aspx}
UrlReferrer	null
UserAgent	"Mozilla/4.0 (compatible; MSIE 8.0; Window
UserHostAddress	"127.0.0.1"
UserHostName	"127.0.0.1"
UserLanguages	{string[1]}

In a slightly cryptic way the text informs me that I am using Microsoft Internet Explorer 8 (MSIE 8.0). This will be different if you are using a different browser.

tip

Shortcuts to Page properties

If you were to add a watch for *Request* instead of *Page.Request*, you'd see exactly the same results.

This is because the page's code-behind file is considered to be part of the page, so it can access all of the properties of the *Page* object directly.

This book shows the full path to avoid any confusion.

- Look at the other properties of *Page.Request*.

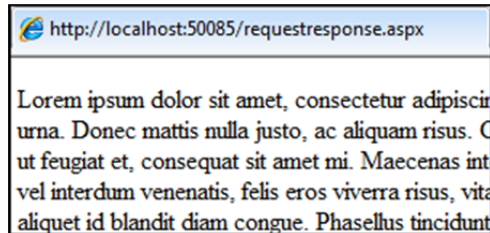
UserHostAddress shows the IP address of the visitor (127.0.0.1 or ::1 means it is your own machine).

Url shows the web address the visitor entered to get to this page.

7 Manipulate *Page.Response*.

- Stop debugging and remove the breakpoint.
- Run *requestresponse.aspx* in Debug mode.

You will see that it is filled with 'Lorem Ipsum' dummy text.



- Stop debugging and return to the code-behind file of *requestresponse.aspx*.
- Add the following code to the *Page_Load* event handler:

```
Page.Response.Write("Hello world!");
Page.Response.End();
```

```
protected void Page_Load(object sender, EventArgs e)
{
    Page.Response.Write("Hello world!");
    Page.Response.End();
}
```

- Run *requestresponse.aspx* in Debug mode.

You'll see that the big block of text that was on the page has disappeared and been replaced by *Hello World!*.



This happened because you ended the page's response to the visitor early. If you hadn't ended the response, the web server would have gone on to add the page's content to the response.

By using *Page.Response.Write*, as you did in this example, you can add content directly to the web server's response without it being on the .aspx page at all. This is useful for testing and debugging purposes, but it is better practice to use a control on the page to display content.

ASP.NET allows you to 'listen' to the user's *Request* and modify the web server's *Response* appropriately. In practice you will rarely need to access the *Page.Request* and *Page.Response* objects directly.

8 Close Visual Studio.

note

The page lifecycle

You might still be a little confused about why there's no content on the page because you stopped the response during *Page_Load*.

This is because ASP.NET doesn't add the page's content until it *Renders* the page (converts the ASP controls into HTML).

Page rendering happens after *Page_Load*. Because you ended the response before the page had a chance to render, the remainder of the page remained blank.

Microsoft's documentation contains a full list of the order in which events are executed when a page is requested (the page lifecycle).