

The Page Lifecycle and ViewState

Mohamad Halabi

Microsoft Integration MVP

Ex- ASP.NET MVP

www.pluralsight.com



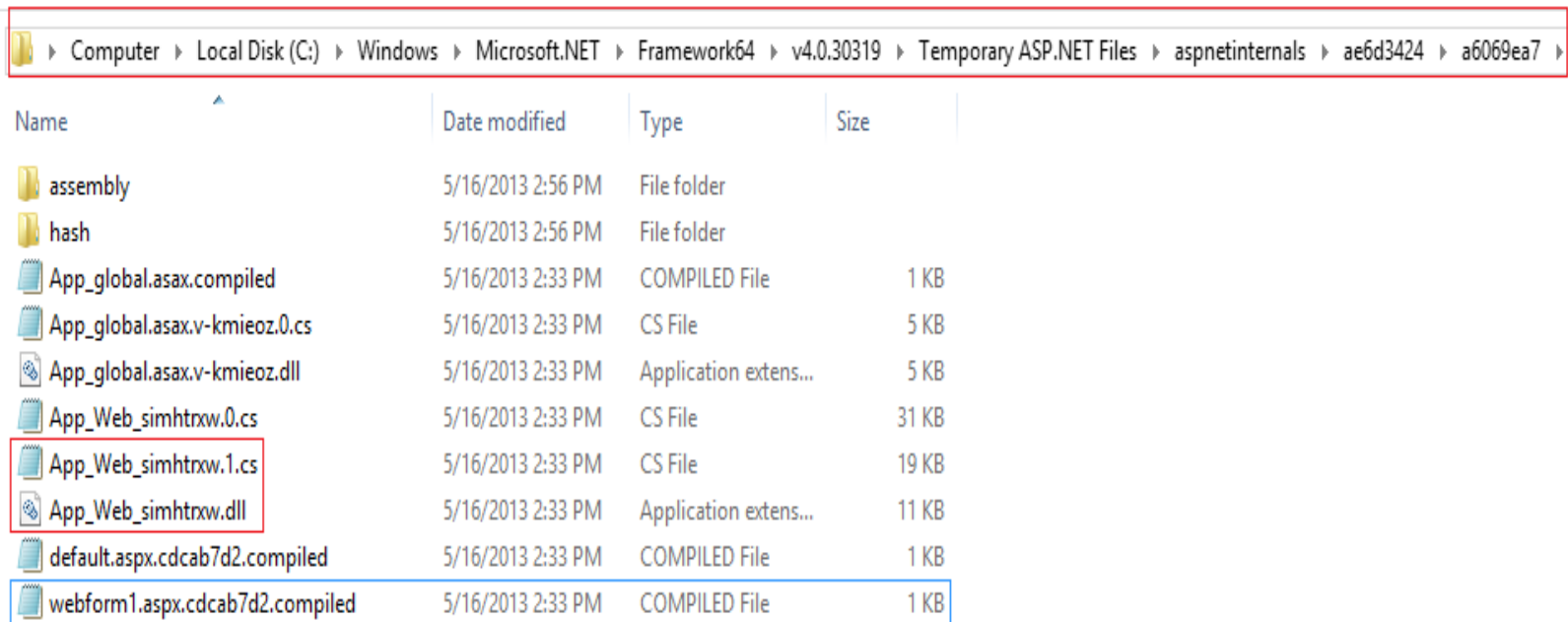
Step0: Generating the Compiled Class

- This step is actually done during the Pre-Page Execution Request Architecture, as shown in Module 2
- Class contains programmatic definition of controls defined in the aspx page
- Class generated and stored in ASP.NET Temporary Files folder upon first request, and updated on page modification or application restart.
- Consider the following aspx:

```
<form id="form1" runat="server">
<table>
  <tr>
    <td><asp:Label runat="server" ID="lblUsername" Text="Username"></asp:Label></td>
    <td><asp:TextBox runat="server" ID="txtUsername"></asp:TextBox></td>
  </tr>
  <tr>
    <td><asp:Label runat="server" ID="lblPassword" Text="Password"></asp:Label></td>
    <td><asp:TextBox runat="server" ID="txtPassword" TextMode="Password"></asp:TextBox></td>
  </tr>
  <tr>
    <td colspan="2"><asp:Button runat="server" ID="btn" Text="Submit" /></td>
  </tr>
</table>
</form>
```

Step0: Generating the Compiled Class, Cont'd

- Upon first request, the generated class is stored in the Temporary Asp.net Files folder, along with the application DLL



Computer > Local Disk (C:) > Windows > Microsoft.NET > Framework64 > v4.0.30319 > Temporary ASP.NET Files > aspnetinternal > ae6d3424 > a6069ea7

Name	Date modified	Type	Size
assembly	5/16/2013 2:56 PM	File folder	
hash	5/16/2013 2:56 PM	File folder	
App_global.asax.compiled	5/16/2013 2:33 PM	COMPILED File	1 KB
App_global.asax.v-kmieoz.0.cs	5/16/2013 2:33 PM	CS File	5 KB
App_global.asax.v-kmieoz.dll	5/16/2013 2:33 PM	Application extens...	5 KB
App_Web_simhtrw.0.cs	5/16/2013 2:33 PM	CS File	31 KB
App_Web_simhtrw.1.cs	5/16/2013 2:33 PM	CS File	19 KB
App_Web_simhtrw.dll	5/16/2013 2:33 PM	Application extens...	11 KB
default.aspx.cdca7d2.compiled	5/16/2013 2:33 PM	COMPILED File	1 KB
webform1.aspx.cdca7d2.compiled	5/16/2013 2:33 PM	COMPILED File	1 KB

- Page will be re-generated upon page update or application restart

Step0: Generating the Compiled Class, Cont'd

- Below is an illustration of the generated class (formatted for demo purposes)

```
public class webform1_aspx : System.Web.IHttpHandler {

    global::System.Web.UI.HtmlControls.HtmlForm @__ctrl;
    @__ctrl.ID = "form1";

    global::System.Web.UI.HtmlControls.HtmlTitle @__ctrl;
    global::System.Web.UI.HtmlControls.HtmlHead @__ctrl;

    global::System.Web.UI.WebControls.Label @__ctrl;
    @__ctrl.ID = "lblUsername";
    @__ctrl.Text = "Username";

    global::System.Web.UI.WebControls.TextBox @__ctrl;
    @__ctrl.ID = "txtUsername";

    global::System.Web.UI.WebControls.Label @__ctrl;
    @__ctrl.ID = "lblPassword";
    @__ctrl.Text = "Password";

    global::System.Web.UI.WebControls.TextBox @__ctrl;
    @__ctrl.ID = "txtPassword";
    @__ctrl.TextMode = System.Web.UI.WebControls.TextBoxMode.Password;

    global::System.Web.UI.WebControls.Button @__ctrl;
    @__ctrl.ID = "btn";
    @__ctrl.Text = "Submit";

    this.__BuildControlTree(this);
}
```

Step1: Page Initialization

■ PreInit

- Entry point of page lifecycle. Available only for Page.
- Only place where programmatic access to master pages and themes is allowed
- Not recursive (not for child controls)

■ Init

- Fired for page and all child controls
 - recall **BuildControlTree** in Step 0?
- Fired bottom-to-up (starting bottom of the control hierarchy)
 - Ex: OnInit will fire on user control X, before page Y which is hosting X

Step1: Page Initialization, Cont'd

■ InitComplete

- End of initialization phase. For the Page only.
- Page.TrackViewState() is called. Starting here, any change in ViewState keys are marked as "**Dirty**", **thus is stored in __VIEWSTATE during the SaveViewState method later in the life cycle**
- That is why, in the previous module, item was marked as dirty in Page_Load but NOT in Page_Init. Because, Load event is fired after InitComplete.
- Note: You can explicitly call Page.TrackViewState() during init or PreInit

Step2: Loading Page Data

■ LoadControlState

- What is Control State?
 - Before ASP.NET 2.0, behavioral state for controls was part of ViewState, so you have to keep it on all the time. This can have severe performance implications (wondering why? You'll get to see this later)
 - Introduced in ASP.NET 2.0. Stores behavioral state information in a separate state. Cannot be turned off.
 - Data also stored in the __VIEWSTATE field
- Only at postbacks
- Not an event that you can subscribe to. It's a virtual protected method which you can override
- Control State saved in the __VIEWSTATE field (via SaveControlState – discussed later), is loaded and then populated in the control hierarchy

Step2: Loading Page Data, Cont'd

- **LoadViewState**

- Only at postbacks
- Not an event that you can subscribe to. It's a virtual protected method which you can override
- Page ViewState which saved in the __VIEWSTATE field (via SaveViewState – discussed later), is loaded and then populated in the control hierarchy
- At this stage, page and controls have restored their previous state courtesy of LoadControlState and LoadViewState

Step2: Loading Page Data, Cont'd

- **LoadPostData**

- Also only at postbacks
- Called by the Page for each control that implements IPostBackDataHandler and returns true if control has changed post data
 - Ex: Textbox text change / LoadPostData loads new value and returns true
 - Ex: DropDownList index not changed / LoadPostData returns false
- Posted form data is loaded into the control hierarchy
 - Ever wondered why data entered in a textbox is preserved upon postbacks? Or why a drop down list selection is preserved upon postbacks?
 - Wish I had a nickel for every time a developer thought this was ViewState!

Step3: Loading the Page

- **PreLoad:**

- Signals end of system level initialization. Raised for the page only.

- **Load**

- Recursive event. However, its top-down rather than down-top.
 - Recall that at this stage, page has restored its previous (pre-postback) state courtesy of LoadViewState and LoadPostData methods

- **RaisePostDataChangedEvent**

- Fired only at postbacks. Also method of IPostBackDataHandler
 - Called for every control that its call of LoadPostData returned true
 - Each control provides its own implementation. For example, TextBox raises its TextChanged event, and DropDownList raises its SelectedIndexChanged event.

Step3: Loading the Page, Cont'd

- **RaisePostBackEvent**

- Fired only at postbacks
- Page determines control that implement IPostBackEventHandler that caused postback. Page calls its RaisePostBackEvent method. Control then raises the appropriate postback event.
 - Ex: Button control raises the onclick event,

- **LoadComplete**

- Page-only and signals end of page loading, now rendering starts...

Step4: Rendering the Page

- **Prerender**

- allows for last updates before the page is rendered
- Recursive top-down

- **PrerenderComplete**

- Signals the end of Prerender

- **SaveControlState**

- Saves the Control State (which is then loaded by LoadControlState in the next postback)
- Called also during first page load
- Recursive

- **SaveViewState**

- Saves the ViewState (which is then loaded by LoadViewState in the next postback)
- Called during first page load / Recursive
- At this stage, both ControlState and ViewState are serialized into __VIEWSTATE hidden field

Step4: Rendering the Page, Cont'd

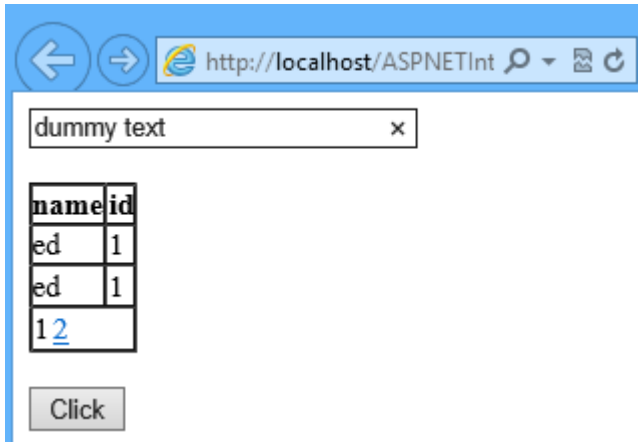
- **Render**

- Recursive event / Top-down
- HTML returned to the client

- **Unload**

- Recursive event / Bottom-up
- Gives chance to perform cleanup before page is disposed

A Full Lifecycle



```
public partial class TestViewState : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //Recall: Dynamic property changes are stored in viewstate
        //Recall: At this stage ViewState tracking is enabled
        if (!IsPostBack)
            txt.Width = 200;
    }
}
```

A Full Lifecycle, Page Load

- **PreInit**
- **Init**
- **InitComplete:**
Page.TrackViewState() is called
- **LoadControlState**
- **LoadViewState**
- **LoadPostData**
- **PreLoad**
- **Load:** Textbox width is marked as "Dirty"
- **RaisePostDataChangedEvent**
- **RaisePostBackEvent**
- **LoadComplete**
- **Prerender**
- **PrerenderComplete**
- **SaveControlState:** stores GridView paging behavior in __VIEWSTATE
- **SaveViewState:** stores Textbox width in __VIEWSTATE
- **Render**
- **Unload**

A Full Lifecycle, Post Back

- **PreInit**
- **Init**
- **InitComplete:**
Page.TrackViewState() is called
- **LoadControlState:** GridView
paging behavior saved in the previous
SaveControlState is loaded back to
the GridView
- **LoadViewState:** TextBox width
saved in the SaveViewState is loaded
back to TextBox
- **LoadPostData:** Postback data
loaded back into TextBox
- **PreLoad**
- **Load**
- **RaisePostDataChangedEvent:**
TextBox TextChanged event is raised
- **RaisePostBackEvent:** Button
Click event is raised
- **LoadComplete**
- **Prerender**
- **PrerenderComplete**
- **SaveControlState**
- **SaveViewState**
- **Render**
- **Unload**