## Introduction

This article talks about the ASP.NET application life cycle and Page life cycle. We will try to see the what all events are usually of the importance for an ASP.NET developer and what needs to be done in these events.

## Background

As an ASP.NET developer, it is crucial to understand the ASP.NET application life cycle and Page life cycle. With the ease of development provided by Visual Studio, sometimes new programmers get started with writingASP.NET pages without understanding the Application and Page life cycle.

From an end user's perspective, a request for a web page is made to the web server and web server will return the page to the user. For a little bit technical users, we can also state that the web server will receive the request, perform some server side activities like reading from database and tailor the HTML output and push it back to the user. From an ASP.NET developer's perspective this is just the tip of the ice berg. As an ASP.NET developer, one need to understand how this request is being processed i.e. the Application life cycle and how the web page is being processed and getting served to the user i.e. the Page life cycle.

### Using the code

Let us start our discussion with how the ASP.NET process the request. Then we will look at the Application life cycle to take better control over the request and response. Finally, we will see the page life cycle to understand what need to be done when to serve the request in a proper manner.

#### Understanding the Application Life Cycle

Whenever the user requests for a web page the request goes to IIS. IIS then checks the ISAPI (Internet Server Application Programming Interface) extension of the request to check how to process the request. If the request is for an .aspx page then the request will be redirected to the ASP.NET.

When the ASP.NET engine receives the request it check whether an Application Domain exist for this request to run. If yes, it will use that application domain, if not it will create one and pass on the request to that Application domain.

**Note:** Application domain provides the actual isolation levels so that various website hosted in the same IISservers will not interfere with each other.

Once the Application domain has been created the objects required to serve the incoming requests and generate the proper response are created. Following objects are created to achieve the same HttpContext,HttpRequest, and HttpResponse. The HttpContext object contains the reference to the HTTPRequest andHTTPResponse(these are the object containing the information about the current request). The HTTPRequestobject specifically contains the information about the current request i.e. browser related info and existing cookies. The HTTPResponse object contains the information about the response that will be sent from the server to the client i.e. cookies to be written etc.

Once these objects are created the object of HTTPApplication will be created. This object contains the methods and events that are common for the application(the current application domain). The reason this object is of particular interest is that this object raises events that could be useful for the developer. If we have aglobal.asax file in our application, we can handle all the events raised by this object and perform our application specific operations (We will see the events in details later).

Once the HTTPApplication object is done with the request initialization, authentication and authorization events, it passes the request to the Page. This is the point typically where we write most of our code. We handle all the page life cycle events to tailor the page as per the user request(We will see the Page events and life cycle shortly).

Once the page processing is done, the HTTPApplication will execute the cleanup and end request events and then the response will be sent to the user. SO if we try to visualize the above process in form of an algorithm.

Hide   Copy Code

1. User initiates the request.

2. Request is received by IIS and checked with ISAPI

2a. If the request is for an .aspx, pass it on to the ASP.NET

3. Check for the AppDmain.

3a. If application domain exists, use it.

3b. If AppDomain does not exist create it.

4. Create the core objects for HTTPContext, HTTPRequest and HTTPResponse.

5. Check if the HTTPApplication object exist.

5a. If it exist pass on the core objects to it.

5b. If it doesn't create it and then pass the core objects.

6.HTTPApplication object will process the request.

6a. HTTPApplication begin, authentication and authorization events will run.

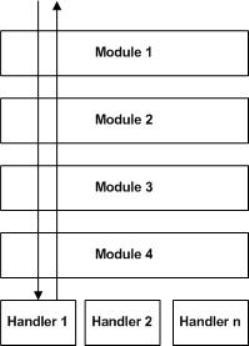
7. The request will then be passed to the Page.

7a. Page events and complete Page life cycle will run.

8. The HTTPApplication object will run the cleanup and end events and pass on the response to the user.

#### A Note on HTTPHandlers and HTTPModules

HTTPHandlers are used by ASP.NET to handle the specific requests based on extensions. HTTPModule, on the other hand, is used if we want to have our own functionality working along with the default ASP.NET functionality. There is one Handler for a specific request but there could be N number of modules for that.



Why is this important? This is important because once the HTTPApplication object get hold of the core request and response object, all the HTTPModules will be created and the init() method for them will be called. So if we have modules in our application, we could also handle the Application events there along with theglobal.asax file.

And for the HTTPHandlers, they will be called after the Application events(both in global.asax and modules) are executed. If we have a handler registered for any particular extension then it will always be called after the Application begin, authentication and authorization events.

So the HTTPModules in our application can also be used to handle application events i.e. 6 and 8 in our algorithm. Where as the HTTPHandlers will come in picture once the Application begin, authentication and authorization events are executed i.e. between 6 and 7 in our algorithm.

#### A Deeper look into Application Events

Now we have understood the basic Application life cycle. the next important thing for us is to look at the Application events' sequence so that we can have our custom logic written either in global.asax file or in a module. Understanding the sequence of events is important because we should know what could and should be done in each event.

Note: We will only look at some important and frequently used Application events. For a comprehensive list of events MSDN is a good place to look into.

| **Event** | **What should be done** |
| --- | --- |
| BeginRequest | This event will be called in every request. This event indicates the beginning of a new request. |
| AuthenticateRequest | This event is fired when the ASP.NET is ready for authentication. All the code that is for the authentication of the user should come here. |
| AuthorizeRequest | This event is fired when the ASP.NET is ready for authorization. All the code that is for the authorization of the user should come here. |
| AcquireRequestState | This event is important because at this point the AS.NET is ready to accept and play with Session variables. If we need to initialize some session variable, it could be done in here or here onwards any event. |
| ProcessRequest | Here the Required HTTPHandler will get executed. If it is an aspx page the respective handler will get executed and the request will be passed on to the page. |
| PAGE EVENTS WILL COME HERE | |
| ReleaseRequestState | This event is important because this is the last point play with Session variables. |
| EndRequest | This event will be fired just before the response will be sent to the user. |

#### Understanding the Page Life Cycle

Having talked about the application life cycle and application events, It the the right time to discuss the Page life cycle and Page events. A page life cycle basically consist of:

1. Start
2. Initialize
3. Load
4. Validate
5. Event Handling
6. Render
7. Unload

The Start phase is not associated with any event. It is just to indicate that the request has been passed on to the page. Similarly the Validate and Event handling is not associated with any predefined event. Its just that the validation will occur once the Load is done and the user defined controls' events will be fired after validation and before Render.

For rest of the phases i.e. Initialize, Load, Render and Unload there are predefined events associated with them. For Unload there is only one event but for rest of them there are 3 events associated with each. One fired before the phase starts, second during the phase and last one after the work is done. So following list shows the typical events sequence for the Page.

1. PreInit
2. Init
3. InitComplete
4. Preload
5. Load
6. LoadComplete
7. PreRender
8. PreRenderComplete
9. Unload

One more event that has significance in the Page life cycle is SaveStateComplete. It will be called afterPreRenderComplete and this indicates that from this point onwards changes in ViewState will not be preserved i.e. ViewState has been saved for the page already.

So if we now try to draw a matrix on sequence of events and what should be done in each event.

| **Event** | **Description** |
| --- | --- |
| PreInit | All the Dynamic Controls, Master pages, Profiles and themes should be set in this event. |
| Init | This should be used to set the initial value of the Control properties. |
| InitComplete | This should be used to have custom ViewState data. This is the first place where ViewState has been loaded and can be changed. |
| Preload | This can be used to set the properties of the controls. |
| Load | All the Database connections and Data Binding can be performed here. Before this event finishes up all the validations will be done. Once the event is finished, the events for all the controls will execute before calling the next event in this list. |
| LoadComplete | This event can be used for activities on controls that require them to be fully loaded.s |
| PreRender | This is the last page where the visual properties of the controls can be changed before getting them displayed on the page. |
| PreRenderComplete | This will be called when the page is ready and no changes in visual elements can be made. All data binding are done at this point. |
| SaveStateComplete | View State has been saved and from this point onwards changes in ViewState will not be preserved i.e.ViewState has been saved for the page already. |
| Unload | The Page processing is done now. This is the last event that will be called. |

The important thing to consider here is that all these events will call the respective events of their child controls. The Page will also keep the events of all the child controls in sync with its own events. That is why is is a good idea to add dynamic controls in PreInit only. If we add dynamic controls later like in Load then the page will wait till all the added controls reach the load state and fully catch up with the page events.

Let us now go ahead and see all these sequence of events in a toy application. The sample application contains all the events handled on an empty single page website. All the events have a sequence number and the event name message getting printed on the debug console to illustrate how these events are getting fired. Here is a couple of sample functions from the code:

void Application\_BeginRequest(object sender, EventArgs e)

{

System.Diagnostics.Debug.WriteLine("MyLog: 1. Application\_BeginRequest");

}

void Application\_AuthenticateRequest(object sender, EventArgs e)

{

System.Diagnostics.Debug.WriteLine("MyLog: 2. Application\_AuthenticateRequest");

}

And the output for this looks like:

The red highlighted area represent the phase when the respective handler gets called and in this case which would call the aspx page.