Following are the page life cycle events:

* **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page\_PreInit handler.
* **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page\_Init handler.
* **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
* **LoadViewState** - LoadViewState event allows loading view state information into the controls.
* **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
* **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page\_PreLoad handler.
* **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page\_Load handler.
* **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page\_LoadComplete handler
* **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
* **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
* **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page\_Render handler.
* **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page\_UnLoad handler.

**PreInit**

**Init**

**InitComplete**

**PreLoad**

**Load**

**ControlEvents**

**LoadComplete**

**PreRender**

**PreRenderComplete:**

**SaveStateComplete**

**RenderComplete**

.**Unload**

**PreInit:**

You can:

* Check for the IsPostBack property to determine whether this is the first time the page is being processed.
* Create or recreate dynamic controls.
* Set master page dynamically.
* Set the Theme property dynamically.
* Read or set profile property values.

If Request is postback:

* The values of the controls have not yet been restored from view state.
* If you set control property at this stage, its value might be overwritten in the next event.

**Init:**

* In the Init event of the individual controls occurs first, later the Init event of the Page takes place.
* This event is used to initialize control properties.

**InitComplete:**

* Tracking of the ViewState is turned on in this event.
* Any changes made to the ViewState in this event are persisted even after the next postback.

**PreLoad:**

* This event processes the postback data that is included with the request.

**Load:**

* In this event the Page object calls the OnLoad method on the Page object itself, later the OnLoad method of the controls is called.
* Thus Load event of the individual controls occurs after the Load event of the page.

**ControlEvents:**

* This event is used to handle specific control events such as a Button control’s Click event or a TextBoxcontrol’s TextChanged event.

In case of postback:

* If the page contains validator controls, the Page.IsValid property and the validation of the controls takes place before the firing of individual control events.

**LoadComplete:**

* This event occurs after the event handling stage.
* This event is used for tasks such as loading all other controls on the page.

**PreRender:**

* In this event the PreRender event of the page is called first and later for the child control.

Usage:

* This method is used to make final changes to the controls on the page like assigning the DataSourceIdand calling the DataBind method.

**PreRenderComplete:**

* This event is raised after each control's PreRender property is completed.

**SaveStateComplete:**

* This is raised after the control state and view state have been saved for the page and for all controls.

**RenderComplete:**

* The page object calls this method on each control which is present on the page.
* This method writes the control’s markup to send it to the browser.

**Unload:**

* This event is raised for each control and then for the Page object.

Usage:

* Use this event in controls for final cleanup work, such as closing open database connections, closing open files, etc.

Creation of ASP.NET Environment

**Step 1:** The user sends a request to IIS. IIS first checks which ISAPI extension can serve this request. Dependingon file extension the request is processed. For instance, if the page is an ‘.ASPX page’, then it will be passed to *aspnet\_isapi.dll*’ for processing.

**Step 2:** If this is the first request to the website, then a class called as ‘ApplicationManager’ creates an application domain where the website can run. As we all know, the application domain creates isolation between two web applications hosted on the same IIS. So in case there is an issue in one app domain, it does not affect the other app domain.  
  
**Step 3:** The newly created application domain creates hosting environment, i.e. the ‘HttpRuntime’ object. Once the hosting environment is created, the necessary core ASP.NET objects like ‘HttpContext’ , ‘HttpRequest’ and ‘HttpResponse’ objects are created.  
  
**Step 4:** Once all the core ASP.NET objects are created, ‘HttpApplication’ object is created to serve the request. In case you have a ‘*global.asax*’ file in your system, then the object of the ‘*global.asax*’ file will be created. Please note *global.asax* file inherits from ‘HttpApplication’ class.  
**Note**: The first time an ASP.NET page is attached to an application, a new instance of ‘HttpApplication’ is created. Said and done to maximize performance, HttpApplication instances might be reused for multiple requests.  
  
**Step 5:** The HttpApplication object is then assigned to the core ASP.NET objects to process the page.  
  
**Step 6:** HttpApplication then starts processing the request by HTTP module events, handlers and page events. It fires the MHPM event for request processing.

The below image explains how the internal object model looks like for an ASP.NET request. At the top level is the ASP.NET runtime which creates an ‘Appdomain’ which in turn has ‘HttpRuntime’ with ‘request’, ‘response’ and ‘context’ objects.

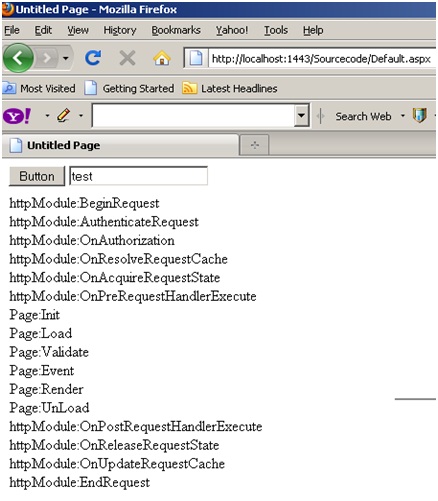
Process Request using MHPM Events Fired

Once ‘HttpApplication’ is created, it starts processing requests. It goes through 3 different sections‘HttpModule’ , ‘Page’ and ‘HttpHandler’. As it moves through these sections, it invokes different events which the developer can extend and add customize logic to the same.  
Before we move ahead, let's understand what are ‘HttpModule’ and ‘HttpHandlers’. They help us to inject custom logic before and after the ASP.NET page is processed. The main differences between both of them are:

* If you want to inject logic based in file extensions like ‘*.ASPX*’, ‘*.HTML*’, then you use ‘HttpHandler’. In other words, ‘HttpHandler’ is an extension based processor.

**Step 1(M: HttpModule):** Client request processing starts. Before the ASP.NET engine goes and creates the ASP.NET HttpModule emits events which can be used to inject customized logic. There are 6 important events which you can utilize before your page object is created BeginRequest, AuthenticateRequest,AuthorizeRequest, ResolveRequestCache, AcquireRequestState and PreRequestHandlerExecute.  
  
**Step 2 (H: ‘HttpHandler’):** Once the above 6 events are fired, ASP.NET engine will invoke ProcessRequestevent if you have implemented HttpHandler in your project.  
  
**Step 3 (P: ASP.NET page):** Once the HttpHandler logic executes, the ASP.NET page object is created. While the ASP.NET page object is created, many events are fired which can help us to write our custom logic inside those page events. There are 6 important events which provides us placeholder to write logic inside ASP.NET pages Init, Load, validate, event, render and unload. You can remember the word SILVER to remember the events S – Start (does not signify anything as such just forms the word) , I – (Init) , L (Load) , V (Validate), E (Event) and R (Render).  
  
**Step4 (M: HttpModule):** Once the page object is executed and unloaded from memory, HttpModule provides post page execution events which can be used to inject custom post-processing logic. There are 4 important post-processing events PostRequestHandlerExecute, ReleaserequestState, UpdateRequestCache andEndRequest.  
The below figure shows the same in a pictorial format.

In What Event Should We Do What?

The million dollar question is in which events should we do what? Below is the table which shows in which 

Zooming ASP.NET Page Events

In the above section, we have seen the overall flow of events for an ASP.NET page request. One of the most important sections is the ASP.NET page, we have not discussed the same in detail. So let’s take some luxury to describe the ASP.NET page events in more detail in this section.

| **Seq** | **Events** | **Controls Initialized** | **View state  Available** | **Form data Available** | **What Logic can be written here?** |
| --- | --- | --- | --- | --- | --- |
| 1 | Init | No | No | No | **Note**: You can access form data etc. by using ASP.NET request objects but not by Server controls.Creating controls dynamically, in case you have controls to be created on runtime. Any settinginitialization.Master pages and them settings. In this section, we do not have access to viewstate , posted values and neither the controls are initialized. |
| 2 | Load view state | Not guaranteed | Yes | Not guaranteed | You can access view state and any synch logic where you want viewstate to be pushed to behind code variables can be done here. |
| 3 | PostBackdata | Not guaranteed | Yes | Yes | You can access form data. Any logic where you want the form data to be pushed to behind code variables can be done here. |
| 4 | Load | Yes | Yes | Yes | This is the place where you will put any logic you want to operate on the controls. Like flourishing a combobox from the database, sorting data on a grid, etc. In this event, we get access to all controls, viewstate and their posted values. |
| 5 | Validate | Yes | Yes | Yes | If your page has validators or you want to execute validation for your page, this is the right place to the same. |
| 6 | Event | Yes | Yes | Yes | If this is a post back by a button click or a dropdown change, then the relative events will be fired. Any kind of logic which is related to that event can be executed here. |
| 7 | Pre-render | Yes | Yes | Yes | If you want to make final changes to the UI objects like changing tree structure or property values, before these controls are saved in to view state. |
| 8 | Save view state | Yes | Yes | Yes | Once all changes to server controls are done, this event can be an opportunity to save control data in to view state. |
| 9 | Render | Yes | Yes | Yes | If you want to add some custom HTML to the output this is the place you can. |
| 10 | Unload | Yes | Yes | Yes | Any kind of clean up you would like to do here. |