Effectively, global.asax allows you to write code that runs in response to "system level" events, such as the application starting, a session ending, an application error occuring, without having to try and shoe-horn that code into each and every page of your site.

You can use it by by choosing Add > New Item > Global Application Class in Visual Studio. Once you've added the file, you can add code under any of the events that are listed (and created by default, at least in Visual Studio 2008):

* Application\_Start
* Application\_End
* Session\_Start
* Session\_End
* Application\_BeginRequest
* Application\_AuthenticateRequest
* Application\_Error

The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level and session-level events raised by ASP.NET or by HTTP modules.

The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by HttpModules.

The Global.asax file resides in the root directory of an ASP.NET-based application. The Global.asax file is parsed and dynamically compiled by ASP.NET.

The Global.asax file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

The Global.asax file does not need recompilation if no changes have been made to it. There can be only one Global.asax file per application and it should be located in the application's root directory only.

The Global.asax contains two types of events those are

**Events which are fired for every request**

**Events which are not fired for every request**

Now I will explain about

**Events which are fired for every request**

**Application\_BeginRequest()** – This event raised at the start of every request for the web application.

**Application\_AuthenticateRequest** – This event rose just before the user credentials are authenticated. We can specify our own authentication logic here to provide custom authentication.

**Application\_AuthorizeRequest()** – This event raised after successful completion of authentication with user’s credentials. This event is used to determine user permissions. You can use this method to give authorization rights to user.

**Application\_ResolveRequestCache()** – This event raised after completion of an authorization request and this event used in conjunction with output caching. With output caching, the rendered HTML of a page is reused without executing its code.

**Application\_AcquireRequestState()** – This event raised just before session-specific data is retrieved for the client and is used to populate Session Collection for current request.

**Application\_PreRequestHandlerExecute()** – This event called before the appropriate HTTP handler executes the request.

**Application\_PostRequestHandlerExecute()** – This event called just after the request is handled by its appropriate HTTP handler.

**Application\_ReleaseRequestState()** – This event raised when session specific information is about to serialized from the session collection.

**Application\_UpdateRequestCache()** – This event raised just before information is added to output cache of the page.

**Application\_EndRequest()** – This event raised at the end of each request right before the objects released.

Now we will see

**Events which are not fired for every request**

**Application\_Start()** – This event raised when the application starts up and application domain is created.

**Session\_Start()** – This event raised for each time a new session begins, This is a good place to put code that is session-specific.

**Application\_Error()** – This event raised whenever an unhandled exception occurs in the application. This provides an opportunity to implement generic application-wide error handling.

**Session\_End()** – This event called when session of user ends.

**Application\_End()** – This event raised just before when web application ends.

**Application\_Disposed()** – This event fired after the web application is destroyed and this event is used to reclaim the memory it occupies.

<script runat="server">

protected void Application\_Start(object sender, EventArgs e)

{

// Code that runs on application startup

WriteFile("Application Starting");

}

protected void Application\_End(object sender, EventArgs e)

{

//  Code that runs on application shutdown

WriteFile("Application Ending");

}

protected void Application\_Error(object sender, EventArgs e)

{

// Code that runs when an unhandled error occurs

string strError;

strError = Server.GetLastError().ToString();

if(Context!=null)

{

Context.ClearError();

Response.Write("Application\_Error" + "<br/>");

Response.Write("<b>Error Msg:</b>" + strError + "<br/>"+"<b>End Error Msg<b/>");

}

}

protected void Session\_Start(object sender, EventArgs e)

{

// Code that runs when a new session is started

Response.Write("Session\_Start" + "<br/>");

}

protected void Session\_End(object sender, EventArgs e)

{

// Code that runs when a session ends.

// Note: The Session\_End event is raised only when the sessionstate mode

// is set to InProc in the Web.config file. If session mode is set to StateServer

// or SQLServer, the event is not raised.

Response.Write("Session\_End"+"<br/>");

}

protected void Application\_BeginRequest(object sender,EventArgs e)

{

Response.Write("Application\_BeginRequest" + "<br/>");

}

protected void Application\_EndRequest(object sender, EventArgs e)

{

Response.Write("Application\_EndRequest" + "<br/>");

}

protected void Application\_AcquireRequestState(object sender, EventArgs e)

{

Response.Write("Application\_AcquireRequestState" + "<br/>");

}

protected void Application\_AuthenticateRequest(object sender, EventArgs e)

{

Response.Write("Application\_AuthenticateRequest" + "<br/>");

}

protected void Application\_AuthorizeRequest(object sender, EventArgs e)

{

Response.Write("Application\_AuthorizeRequest" + "<br/>");

}

protected void Application\_PostRequestHandlerExecute(object sender, EventArgs e)

{

Response.Write("Application\_PostRequestHandlerExecute" + "<br/>");

}

protected void Application\_PreRequestHandlerExecute(object sender, EventArgs e)

{

Response.Write("Application\_PreRequestHandlerExecute" + "<br/>");

}

protected void Application\_PreSendRequestContent(object sender, EventArgs e)

{

Response.Write("Application\_PreSendRequestContent" + "<br/>");

}

protected void Application\_PreSendRequestHeaders(object sender, EventArgs e)

{

Response.Write("Application\_PreSendRequestHeaders" + "<br/>");

}

protected void Application\_ReleaseRequestState(object sender, EventArgs e)

{

Response.Write("Application\_ReleaseRequestState" + "<br/>");

}

protected void Application\_ResolveRequestCache(object sender, EventArgs e)

{

Response.Write("Application\_ResolveRequestCache" + "<br/>");

}

protected void Application\_UpdateRequestCache(object sender, EventArgs e)

{

Response.Write("Application\_UpdateRequestCache" + "<br/>");

}

protected void Application\_Disposed(object sender, EventArgs e)

{

Response.Write("Application\_Disposed"+"<br/>");

}

public void WriteFile(string strText)

{

StreamWriter strWriter = new StreamWriter(@"C:test.txt", true);

string str = DateTime.Now + " " + strText;

strWriter.WriteLine(str);

strWriter.Close();

}

</script>