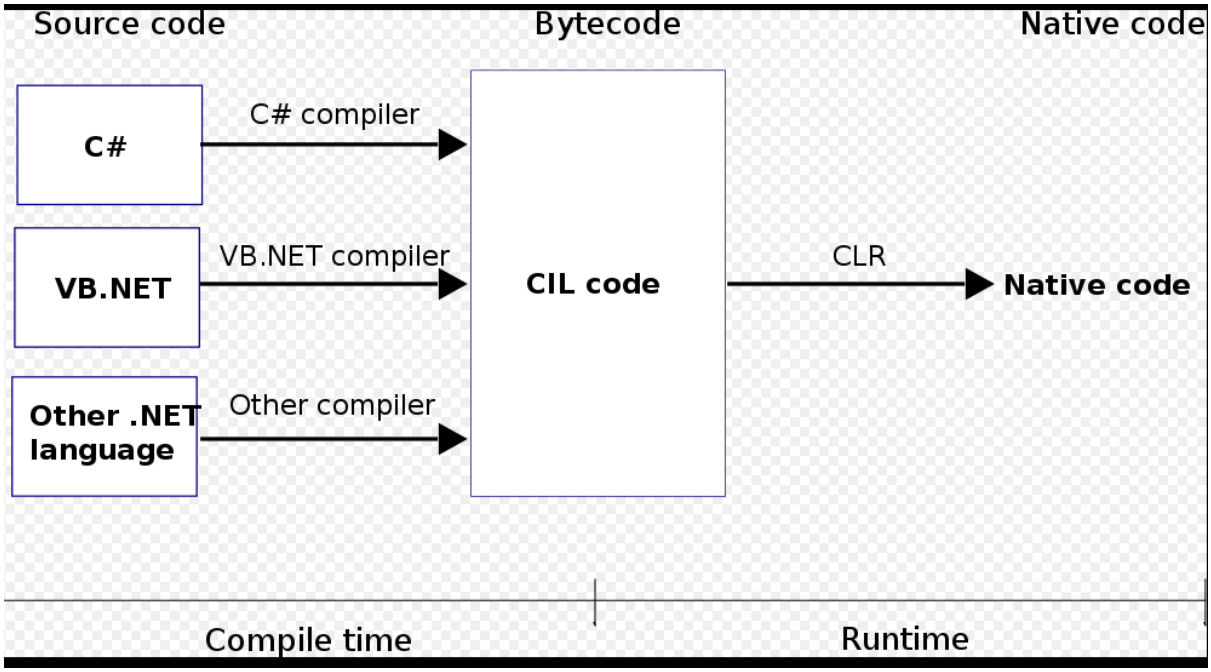
Broardge 2

Common Language Runtime

The **Common Language Runtime** (**CLR**), the virtual machine component of Microsoft's .NET framework, manages the execution of .NET programs.

A process known as just-in-time compilation converts compiled code into machine instructions which the computer's CPU then executes.[1]

The CLR provides additional services including memory management, type safety, exception handling, garbage collection, security and thread management. All programs written for the .NET framework, regardless of programming language, are executed by the CLR.



Benefits[edit]

The runtime provides the following benefits:-

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded, scalable applications.
- Support for structured exception handling.
- Support for custom attributes.
- Garbage collection.

- Use of delegates instead of function pointers for increased type safety and security. For more information about delegates, see Common Type System.

Using Cookie-less Session in ASP.NET

> We use **Session** in ASP.NET application to **maintain the state of the user**.
>
> These Sessions **too use Cookies** in the background to associate Sessions with the correct user.
>
> But if a user has turned off his browser's cookies then our application will not work on these browsers.
>
> For this situation we use **Cookie-less Sessions**. **In Cookie-less Sessions, the values that are required to associate users with their sessions are appended to the browser's URL.**

By default a session uses a cookie in the background. To enable a cookie-less session, we need to change some configuration in the Web.Config file. Follow these steps:

1. Open Web.Config file
2. Add a <sessionState> tag under <system.web> tag
3. Add an attribute "cookieless" in the <sessionState> tag and set its value to "AutoDetect" like below:

```
<sessionState cookieless="AutoDetect" regenerateExpiredSessionId="true"/>
```

The possible values for "cookieless" attribute are:

- **AutoDetect :** Session uses background cookie if cookies are enabled. If cookies are disabled, then the URL is used to store session information.
- **UseCookie:** Session always use background cookie. This is default.
- **UseDeviceProfile:** Session uses background cookie if browser supports cookies else URL is used.
- **UseUri:** Session always use URL.

ASP.NET Page Life Cycle

Following are the different stages of an ASP.NET page:

- **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
- **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
- **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
- **Page load** - At this stage, control properties are set using **the view state** and control state values.
- **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
- **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
- **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
- **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

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.

Adding Dynamic Rows in GridView with TextBoxes

```
private void AddNewRowToGrid(){

        int rowIndex =0;
        if (ViewState["CurrentTable"] != null)
        {
            DataTable dtCurrentTable = (DataTable)ViewState["CurrentTable"]
;
            DataRow drCurrentRow = null;
            if (dtCurrentTable.Rows.Count > 0)
            {
                for (int i = 1; i <= dtCurrentTable.Rows.Count; i++)
                {
                    //extract the TextBox values
                    TextBox box1 = (TextBox)Gridview1.Rows[rowIndex].Cells[
1].FindControl("TextBox1");
                    TextBox box2 = (TextBox)Gridview1.Rows[rowIndex].Cells[
2].FindControl("TextBox2");
                    TextBox box3 = (TextBox)Gridview1.Rows[rowIndex].Cells[
3].FindControl("TextBox3");

                    drCurrentRow = dtCurrentTable.NewRow();
                    drCurrentRow["RowNumber"] = i + 1;
                    drCurrentRow["Column1"] = box1.Text;
                    drCurrentRow["Column2"] = box2.Text;
                    drCurrentRow["Column3"] = box3.Text;

                    rowIndex++;
                }

                //add new row to DataTable
                dtCurrentTable.Rows.Add(drCurrentRow);
                //Store the current data to ViewState
                ViewState["CurrentTable"] = dtCurrentTable;

                //Rebind the Grid with the current data
                Gridview1.DataSource = dtCurrentTable;
                Gridview1.DataBind();
```

```
            }
        }
        else
        {
            Response.Write("ViewState is null");
        }

        //Set Previous Data on Postbacks
        SetPreviousData();
    }
```

```
protected void ButtonAdd_Click(object sender, EventArgs e){
        AddNewRowToGrid();
}
```