Event-Driven Programming and Postback

Chapter 3

What Are You Going To Learn?

At the end of this lesson, you will be able to:

- Define event
- Differentiate HTML events and Server Control events
- Explain ASP.NET page events and PostBack
- Apply event-driven programming

Event

- An event is an action taken on your application by some force outside of your code.
- This external force is usually the user, but could be another program.
- An event can hold code that will run when the action occurs.

Event-driven Environment

- Event-driven environment can be broken down into 4 chronological sections:

Section	Examples
An event occurs	The user clicks on a button (force from outside)
The system detects the event	ASP.NET registers that an event has occurred
The system reacts to the event	Some code is executed, e.g. calculation and display result
The system then returns to its original state	Waits for the next event

Event-driven Programming

- The sequential way of code execution (line followed by line) is no longer necessary.
- In event-driven programming, execution depends on user, i.e.:
  - system will wait until an event occurs

Events

- 3 major groups of events are supported by ASP.NET:
  - HTML events
  - ASP.NET page-level events
  - ASP.NET server control events

HTML Events

- Occur on the Web page and are handled by the browser (client)
- No transfer of information to the server.
- Used to handle events on the client-side.
  - Example:

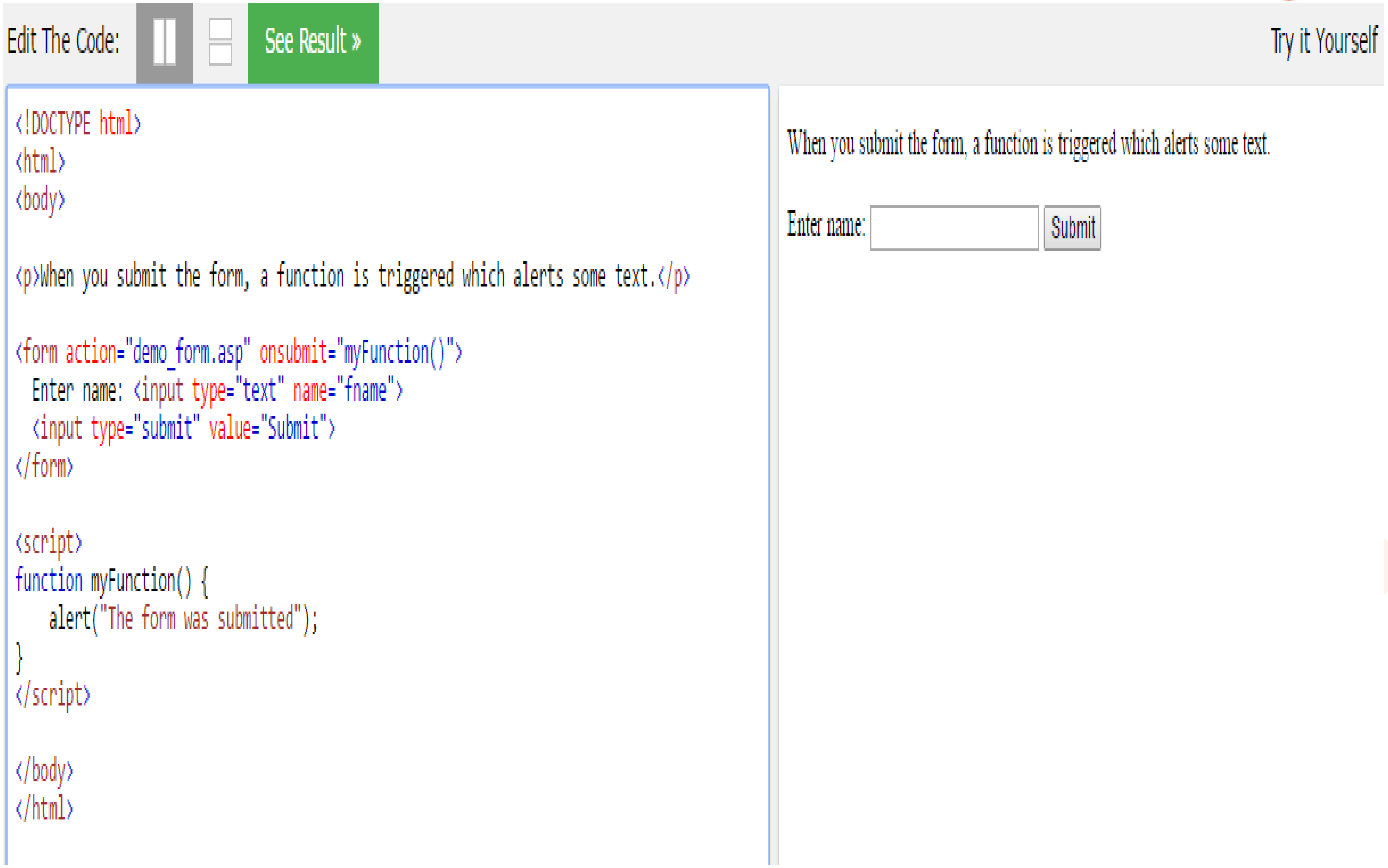
  1) ______________________________________

  2) ______________________________________

  3) ______________________________________

TARUMT TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

HTML Events



```
<!DOCTYPE html>
<html>
<body>

<p>When you submit the form, a function is triggered which alerts some text.</p>

<form action="demo_form.asp" onsubmit="myFunction()">
  Enter name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

<script>
function myFunction() {
    alert("The form was submitted");
}
</script>

</body>
</html>
```

When you submit the form, a function is triggered which alerts some text.

Enter name: [ ] Submit

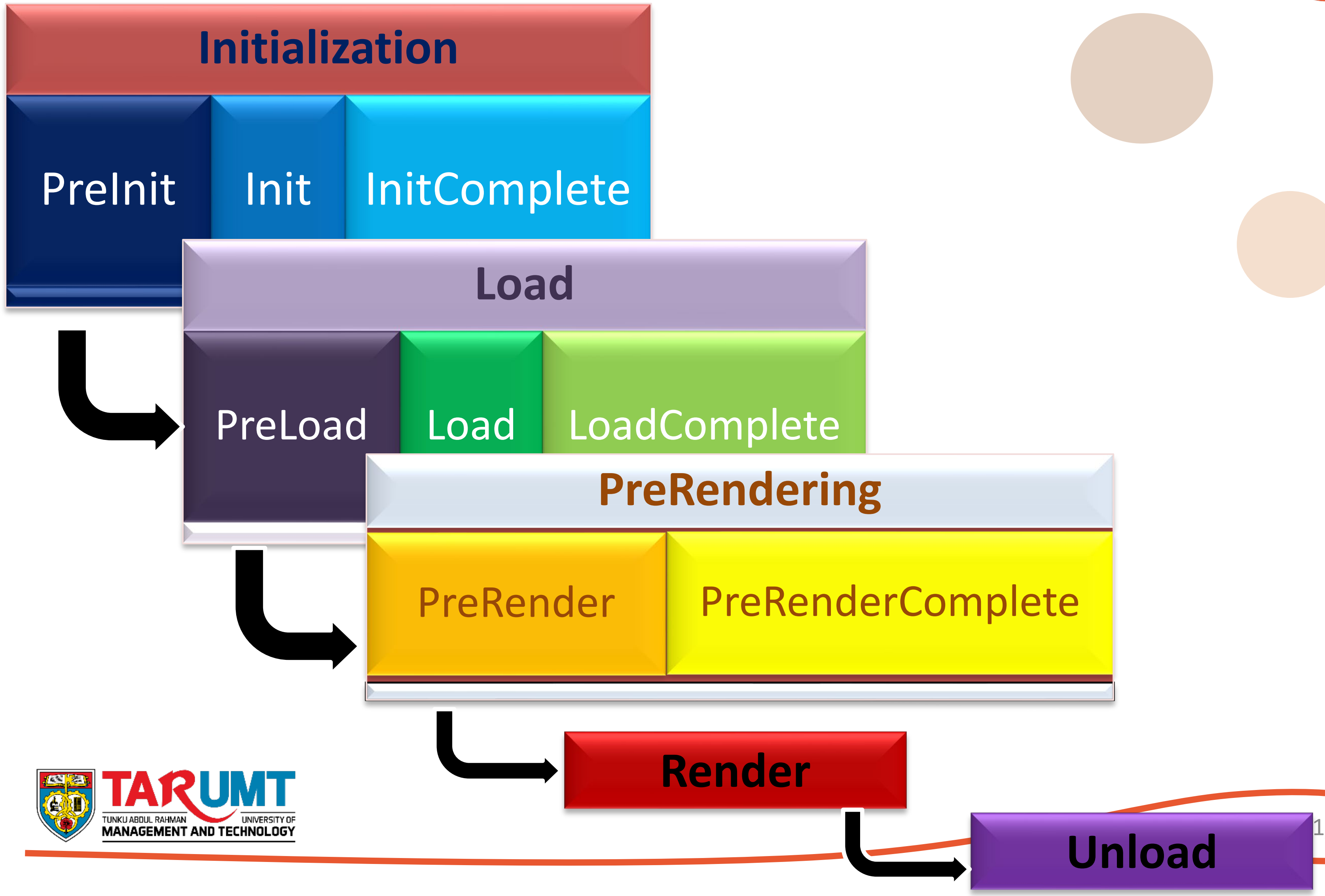
ASP.NET Events

- ASP.NET Page Events
- ASP.NET Server Control Events

ASP.NET Page Events

- Events that are automatically run by ASP.NET when a page loads
- They occur before the user can even sees the page.
- No user involvement.
- E.g. page load

Page Events and Stages



A series of events automatically occur on the Web server as follows:

Event	Occurs when ....
Init	A page is requested from the server. This event is raised before the view state of the page controls has been restored.
Load	A page is requested from the server, after all controls have been initialized and view state has been restored. This is the event you typically use to perform initialization operation such as retrieving data and initializing form controls.
PreRender	All the control events for the page have been processed but before the HTML that will be sent back to the browser is generated. (Not in your syllabus)
Unload	the page is unloaded from IIS memory and sent out to browser.

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

ASP.NET Server Control Events

- Largest events group
- Handled by server
- Occur due to user interaction with the page, i.e. it does not occur automatically like page event.
- E.g. click event that associates with <asp:Button>

ASP.NET Server Control Events

ADVANTAGES:

- All the event handler code executed on the server – more available resources like custom-built objects, etc.
- Not relying on the browser's capability to recognize and handle HTML events
- Can use any language to write code for event handler

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

Recall

1. What are the 3 major groups of events supported by ASP.NET?
2. What are the 2 groups of events handled by server?
3. Why is it necessary to handle events on the client side since server-side event handlers are so powerful.

Adding Events to Server Controls

2 steps:

1. Add attribute with the name of the event and set its value to an event handler

e.g.

```
<asp:Button runat="server" id="button1"
Text="button" onclick="Button1_Click" />
```

event

event handler name

TARUMT TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

Adding Events to Server Controls

2. Create the event handler – the function that runs when invoked by an event

```
protected void Button1_Click(object sender, EventArgs e)
 {
   // insert code to handle the click event
 }
```

provides a reference to the object (caller) that raised the event

an event class that captures information regarding the state of the event being handled, and passes an object that's specific to the event

Example of code - ImageClickEventArgs

```
<asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="~/fish.jpg“ OnClick="ButtonClick" />

<asp:ImageButton ID="ImageButton2" runat="server" ImageUrl="~/cat.jpg" OnClick="ButtonClick" />
```

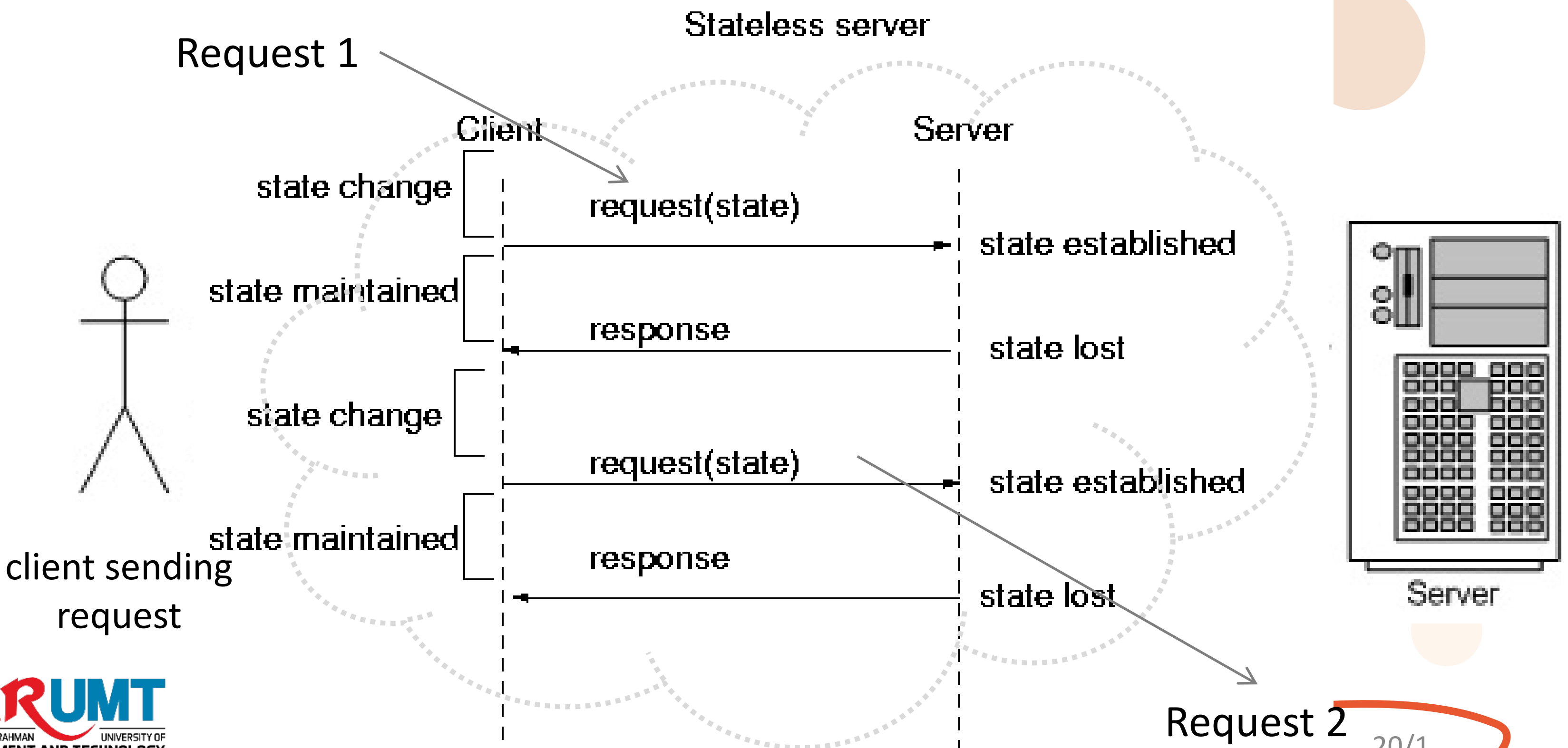
```
protected void ButtonClick(object sender, ImageClickEventArgs e)
   {
      if(sender.Equals(ImageButton1))
         Label1.Text = "You clicked Button 1 at position (";
      else
         Label1.Text = "You clicked Button 2 at position (";

    Label1.Text += e.X.ToString() + ", " + e.Y.ToString() + ")";
   }
```

Differences between HTML Event and Server Control Event

HTML Event Handler	Server Control Event Handler
Runs on client side	Runs on server side
Rely on browser support	Works across all browsers since event is handled by server
Event handling code can only be written in Javascript	Event handling code can be written in any languages supported by .NET
Resources restricted to client side	More available resources

Stateless Request

- HTTP is stateless - it retains no knowledge from one request to the next.

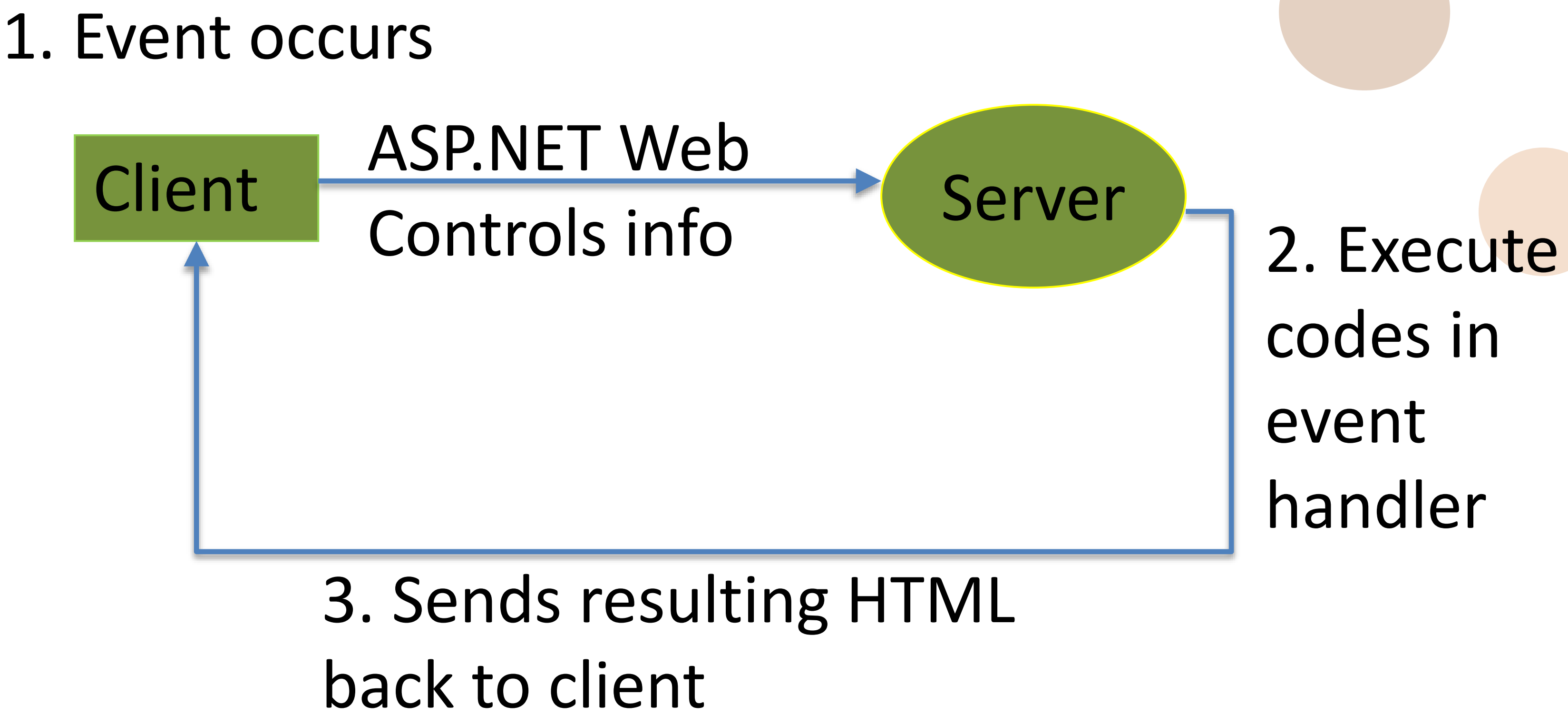


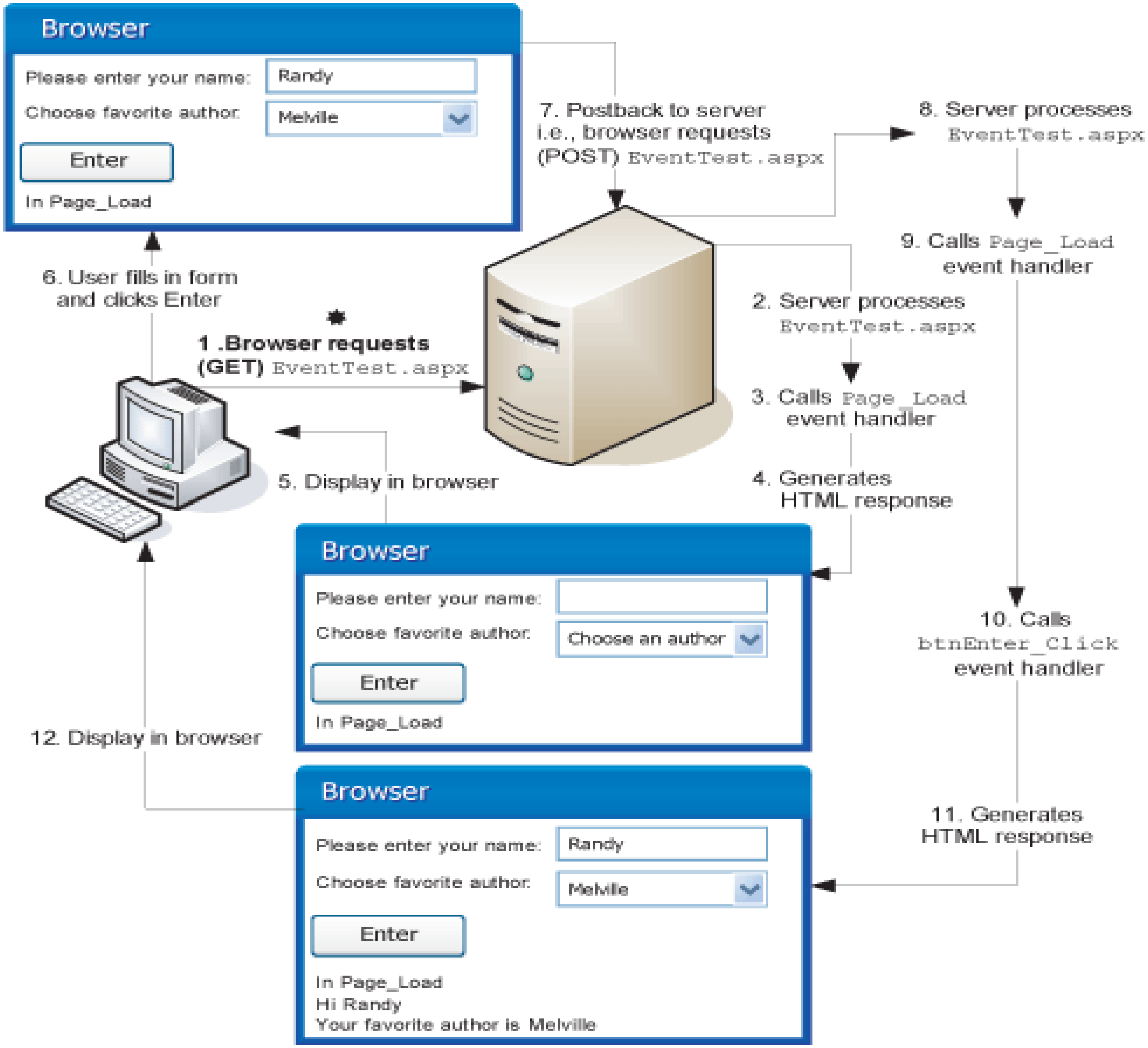
Stateful Postback

- server controls can retain their state across postbacks.
- Postback - process by which the browser sends information to the server, which then handles the event, sending html back to the client.

Web forms must have `runat="server"` and only ASP.NET web controls can retain state.

Postback Process





Browser
Please enter your name:
Randy
Choose favorite author:
Melvile
Enter
In Page_Load
7. Postback to server i.e., browser requests (POST) EventTest.aspx
8. Server processes EventTest.aspx
9. Calls Page_Load event handler
6. User fills in form and clicks Enter
1 .Browser requests (GET) EventTest.aspx
2. Server processes EventTest.aspx
3. Calls Page_Load event handler
4. Generates HTML response
5. Display in browser
Browser
Please enter your name:
Choose favorite author:
Choose an author
Enter
In Page_Load
10. Calls btnEnter_Click event handler
12. Display in browser
11. Generates HTML response
Browser
Please enter your name:
Randy
Choose favorite author:
Melvile
Enter
In Page_Load
Hi Randy
Your favorite author is Melville

_VIEWSTATE

- When Postback is used, information about the state of the ASP.NET form is sent back in an associated hidden control "_VIEWSTATE", i.e.

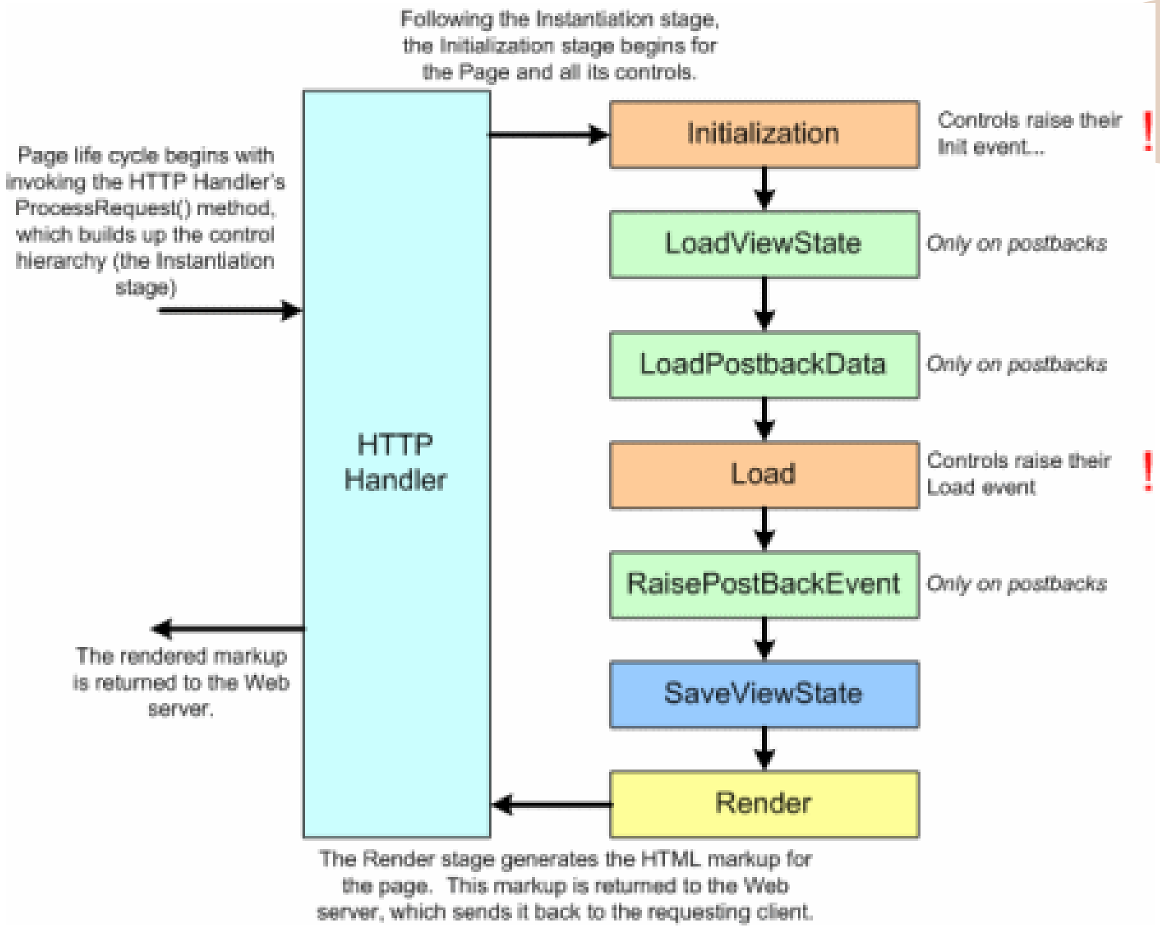
```
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTMwNzcxMzI0Ozs+1ey3CbAun6n
9axTH/QhVfpr1V/8=" />
```

encrypted value

_VIEWSTATE

- Information (value) in _VIEWSTATE is generated by ASP.NET by encrypting the values of the old state of the form (e.g. user selections for each Web control)
- With this information, ASP.NET can persist the state between page submissions.

_VIEWSTATE


Following the Instantiation stage, the Initialization stage begins for the Page and all its controls.
Page life cycle begins with invoking the HTTP Handler's ProcessRequest() method, which builds up the control hierarchy (the Instantiation stage)
HTTP Handler
Initialization
Controls raise their Init event...
!
LoadViewState
Only on postbacks
LoadPostbackData
Only on postbacks
Load
Controls raise their Load event
!
RaisePostBackEvent
Only on postbacks
SaveViewState
Render
The rendered markup is returned to the Web server.
The Render stage generates the HTML markup for the page. This markup is returned to the Web server, which sends it back to the requesting client.

TARUMT TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

_VIEWSTATE

- Stage 1 – Init page with default properties and values.
- Stage 2 - Load View State and restore to controls
- Stage 3 - Load postback data. Eg. Form data such as entered text. (NOTE: No viewstate involved here!)
- Stage 4 - Page_Load event triggered.
- Stage 5 - Raise postback event. Eg. Dropdownlist SelectedIndexChanged event, or button click event. Events can be data changed events or raised events. (NOTE: No viewstate involved here again!)
- Stage 6 - Save view state
- Stage 7 - Render

_VIEWSTATE

Q: So why need viewstate?

A: To store programmatic changes.

_VIEWSTATE

- Consider the following aspx:

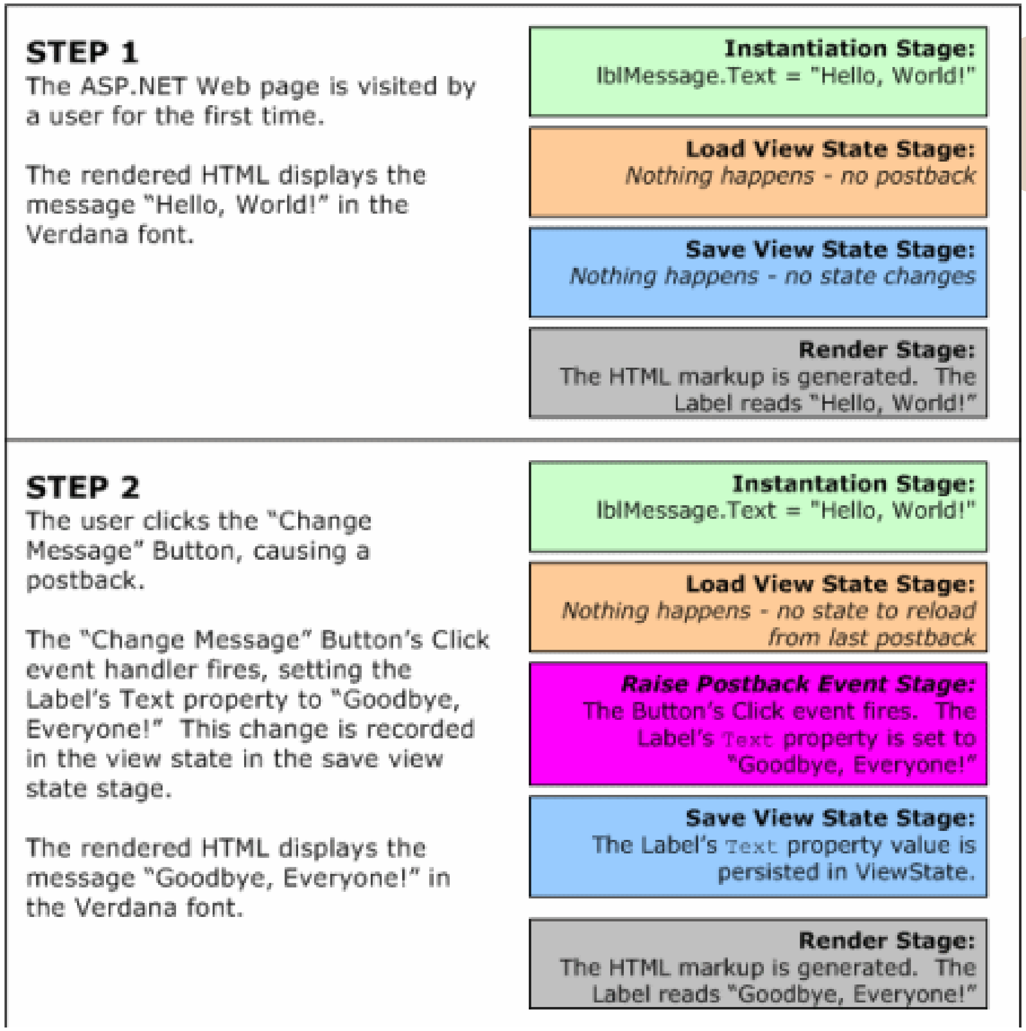
```
<asp:Label runat="server" ID="lblMessage"
  Font-Name="Verdana" Text="Hello, World!"></asp:Label>
<br />
<asp:Button runat="server"
  Text="Change Message" ID="btnSubmit"></asp:Button>
<br />
<asp:Button runat="server" Text="Empty Postback">
</asp:Button>
```

_VIEWSTATE

- The code-behind class has the following event handler:

```
private void btnSubmit_Click(object sender, EventArgs e)
{
  lblMessage.Text = "Goodbye, Everyone!";
}
```

_VIEWSTATE



STEP 1
The ASP.NET Web page is visited by a user for the first time.

The rendered HTML displays the message "Hello, World!" in the Verdana font.

Instantiation Stage:
lblMessage.Text = "Hello, World!"

Load View State Stage:
Nothing happens - no postback

Save View State Stage:
Nothing happens - no state changes

Render Stage:
The HTML markup is generated. The Label reads "Hello, World!"

STEP 2
The user clicks the "Change Message" Button, causing a postback.

The "Change Message" Button's Click event handler fires, setting the Label's Text property to "Goodbye, Everyone!" This change is recorded in the view state in the save view state stage.

The rendered HTML displays the message "Goodbye, Everyone!" in the Verdana font.

Instantiation Stage:
lblMessage.Text = "Hello, World!"

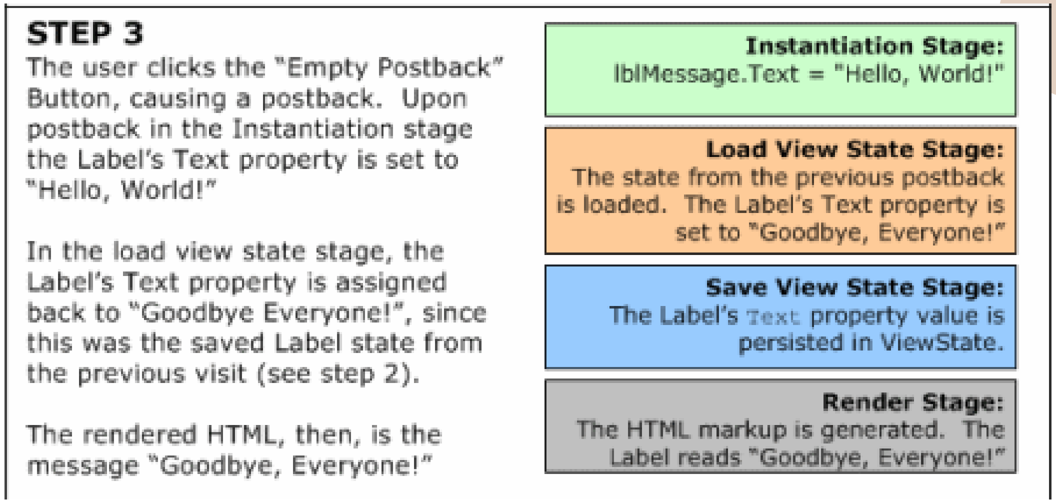
Load View State Stage:
Nothing happens - no state to reload from last postback

Raise Postback Event Stage:
The Button's Click event fires. The Label's `Text` property is set to "Goodbye, Everyone!"

Save View State Stage:
The Label's `Text` property value is persisted in ViewState.

Render Stage:
The HTML markup is generated. The Label reads "Goodbye, Everyone!"

_VIEWSTATE



STEP 3

The user clicks the "Empty Postback" Button, causing a postback. Upon postback in the Instantiation stage the Label's Text property is set to "Hello, World!"

In the load view state stage, the Label's Text property is assigned back to "Goodbye Everyone!", since this was the saved Label state from the previous visit (see step 2).

The rendered HTML, then, is the message "Goodbye, Everyone!"

Instantiation Stage:
lblMessage.Text = "Hello, World!"

Load View State Stage:
The state from the previous postback is loaded. The Label's Text property is set to "Goodbye, Everyone!"

Save View State Stage:
The Label's `Text` property value is persisted in ViewState.

Render Stage:
The HTML markup is generated. The Label reads "Goodbye, Everyone!"

_VIEWSTATE

- Can use ViewState property to persist page-specific and user-specific information across postbacks:

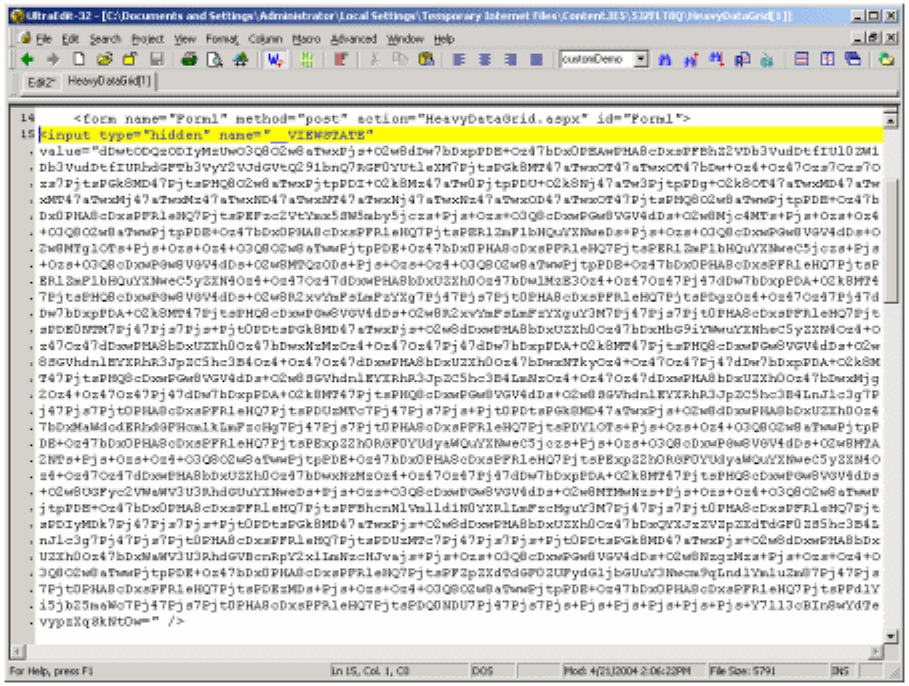
  ViewState[keyName] = value

- Usage scenario example: Creating a pageable, sortable DataGrid, since the sort expression must be persisted across postbacks.

Performance Cost of View State

1. Viewstate needs to be serialized into encoded string. Then, on postbacks, it needs to be deserialized, and update the controls.
2. Adds extra size to the Web page that the client must download.

Performance Cost of View State



```
<form name="Form1" method="post" action="HeavyDataGrid.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
```

value="dDwtODQzODIyMzUwO3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+Oz47bDx0PEAwPHA8cDxsPFBhZ2VDb3VudDtfIU10ZW1
Db3VudDtfIURhdGFTb3VyY2VJdGVtQ291bnQ7RGF0YUtleXM7PjtsPGk8MT47aTwxOT47aTwxOT47bDw+Oz4+Oz47Oz7O
zs7PjtsPGk8MD47PjtsPHQ8O2w8aTwxPjtpPDI+O2k8Mz47aTw0PjtpPDU+O2k8Nj47aTw3PjtpPDg+O2k8OT47aTwxMD47aTw
xMT47aTwxMj47aTwxMz47aTwxND47aTwxNT47aTwxNj47aTwxNz47aTwxOD47aTwxOT47PjtsPHQ8O2w8aTwwPjtpPDE+Oz47b
Dx0PHA8cDxsPFRleHQ7PjtsPEFzc2VtYmx5SW5mby5jczs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8Mjc4MTs+Pjs+Ozs+Oz4
+O3Q8O2w8aTwwPjtpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPER1ZmF1bHQuYXNweDs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O
2w8MTg1OTs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwPjtpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPER1ZmF1bHQuYXNweC5jczs+Pjs
+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8MTQzODs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwPjtpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsP
ER1ZmF1bHQuYXNweC5yZXN4Oz4+Oz47Oz47dDxwPHA8bDxUZXh0Oz47bDw1MzE3Oz4+Oz47Oz47Pj47dDw7bDxpPDA+O2k8MT4
7PjtsPHQ8cDxwPGw8VGV4dDs+O2w8R2xvYmFsLmFzYXg7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7PjtsPDgzOz4+Oz47Oz47Pj47d
Dw7bDxpPDA+O2k8MT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8R2xvYmFsLmFzYXguY3M7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7Pjt
sPDE0NTM7Pj47Pjs7Pjs+Pjt0PDtsPGk8MD47aTwxPjs+O2w8dDxwPHA8bDxUZXh0Oz47bDxHbG9iYWwuYXNheC5yZXN4Oz4+O
z47Oz47dDxwPHA8bDxUZXh0Oz47bDwxNzMzOz4+Oz47Oz47Pj47dDw7bDxpPDA+O2k8MT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w
8SGVhdnlEYXRhR3JpZC5hc3B4Oz4+Oz47Oz47dDxwPHA8bDxUZXh0Oz47bDwzNTkyOz4+Oz47Oz47Pj47dDw7bDxpPDA+O2k8M
T47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8SGVhdnlEYXRhR3JpZC5hc3B4LmNzOz4+Oz47Oz47dDxwPHA8bDxUZXh0Oz47bDxMjg
2Oz47Oz47Pj47dDw7bDxpPDA+O2k8MT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8SGVhdnlEYXRhR3JpZC5hc3B4LnJlc3g7PjtP
j47Pjs7Pjt0PHA8cDxsPFRleHQ7PjtsPDUzMTc7Pj47Pjs7Pjs+Pjt0PDtsPGk8MD47aTwxPjs+O2w8dDxwPHA8bDxUZXh0Oz4
7bDxMaWdodERhdGFHcmlkLmFzcHg7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7PjtsPDY1OTs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwPjtpP
DE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPExpZ2h0RGF0YUdyaWQuYXNweC5jczs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8MTA
2NTs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwPjtpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPExpZ2h0RGF0YUdyaWQuYXNweC5yZXN4O
z4+Oz47Oz47dDxwPHA8bDxUZXh0Oz47bDw1MzE3Oz4+Oz47Oz47Pj47dDw7bDxpPDA+O2k8MT47PjtsPHQ8cDxwPGw8VGV4dDs
+O2w8UGFyc2VWaWV3U3RhdGUuYXNweDs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8NzgzMzs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwP
jtpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPFBhcnNlVmlld1N0YXRlLmFzcHguY3M7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7Pjt
sPDIyMDk7Pj47Pjs7Pjs+Pjt0PDtsPGk8MD47aTwxPjs+O2w8dDxwPHA8bDxUZXh0Oz47bDxQYXJzZVZpZXdTdGF0ZS5hc3B4L
nJlc3g7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7PjtsPDE4MDs+Pjs+Ozs+Oz4+O3Q8O2w8aTwwPjtpPDE+Oz47bDx0PHA8cDxsPFR
leHQ7PjtsPERzMDs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8MTU7Pj47Pjs7Pjs+Pjs+Pjs+Pjs+Pjs+Pjs+Y7113cBInAwYd7e
vypzXq8kNtOw=" />

A DataGrid control's ViewState grows huge quickly

Cross Page PostBacks

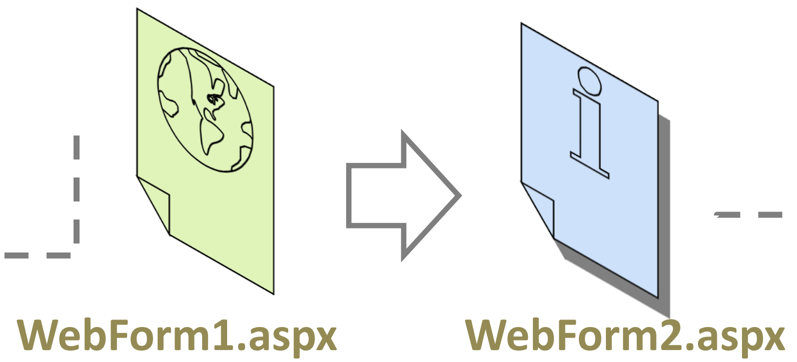
- ASP.NET 2.0 and above includes the notation of a `PostBackUrl` property.

```
<asp:Button PostBackUrl="url" ..>
```

used to get or set the URL of the page to post when the Button control is clicked.

the page is posted to the URL assigned, but not the page itself

Cross Page PostBacks



```
Button1.PostBackUrl = “WebForm2.aspx”;
```

```
TextBox t = PreviousPage.FindControl(“TextBox1”);
Label1.Text = t.Text;
```

`PreviousPage` property holds reference to the page that caused the postback (WebForm1).

`FindControl` method() - To get a control reference from the `PreviousPage` (you can also use the Controls property)

DEMO

Cross-Page PostBacks

NEXT WEEK

DATABASE PROGRAMMING