

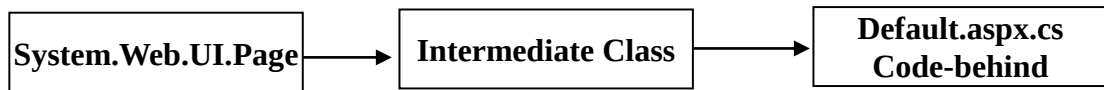
مقدمة تعريفية بـ ASP.Net :- الجزء السادس

Page Class

ها قد وصلنا إلى الجزء السادس , وفي الجزء السابق , عرفنا كيف أن الصفحة default.aspx تصبح default_aspx مع بداية التنفيذ , ولكن نحن على إتفاق انه في النهاية لابد وأن ترث كل الصفحات من Page , تلك الـ class الموجودة داخل System.Web.UI.Page , وأن الآوان للتعلم داخل هذه الـ Class حيث تقوم عليها صفحات asp.net .

فما هي Page Class ؟

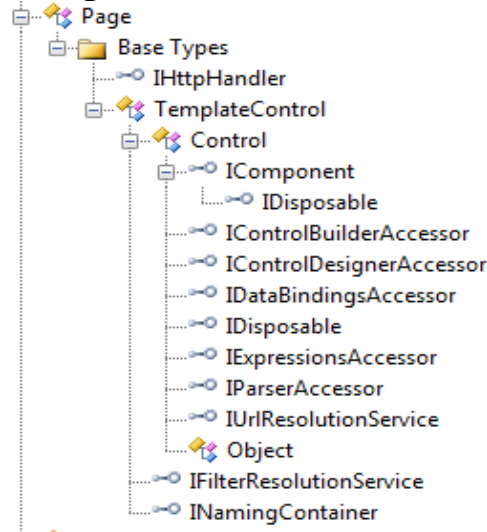
هي Class موجودة بالفعل ضمن معطيات النظام , ونجدها في هذا المسار System.Web.UI.Page , ويجب على كل صفحة aspx ان ترث من هذه الـ Class , أو قد ترث من Class وسيط بدورها ترث من Page . اى على هذا الشكل :-



لماذا Page Class ؟

يوجد هناك وظائف كثيرة مشتركة بين الصفحات وبعضها , فتم وضع هذه الوظائف المشتركة في Class , على أن يقوم البقية بالوراثة منها , واتفقنا أيضاً أنه لابد من وجود Handler ليتولى معالجة الـ Request القادم من المستخدم . فكانت هذه الـ Class داعمة لما سبق التعريف به في اجزاء سابقة وهو IHttpHandler .

ومن الجيد أن نلقى نظرة هنا على التسلسل الشجرى لـ Page Class :-



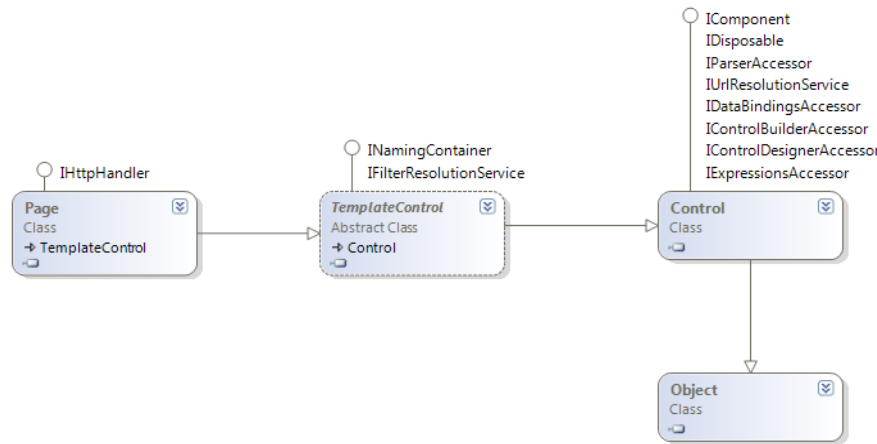
نجد من الشكل الذى أمامنا أن اول الشجرة هو Page Class وآخرها هي Object Class , يتوسطهما كلا من TemplateControl وControl والبقية التى تراها أمامك , ما هي إلا بعضاً من الـ Interfaces تقوم مجموعة الـ Classes التى سبق ذكرها بعمل Implement لها لتوفير المهام اللازمة لإجراء عملياتها . ونأتى بعد قليل للتفصيل.

ملحوظة :-

Object Class هي الـ Class الرئيسية التى لابد وان يرث كل الكائنات داخل الـ .Net , وتسمى

إنجليزياً باسم *Base Class* . وكما سبق وقلنا أنه لا بد من هذه الوراثة , فإن كل الـ *Classes* ترث من *Object* افتراضياً أى لا دخل لك فى ذلك . ولا عجب حين ترى الدوال الأربعة التى تتشكل منها *Object Class* موجودة مع أى *Instance* مأخوذ من أى *class* , هذه الدوال هى , *ToString* , *Equals* , *GetType* , *GetHashCode* , مع العلم أن ثلاثة منهم تم تعريفهم كـ *virtual* , أى تم وضع إمكانية أن تقوم بعمل *Override* لها وتغيير محتواها فى مكان آخر . وواحدة فقط وهى *GetType* هى التى لا يمكن تغيير محتواها وهذا منطقى , لأن نوع الـ *Class* لا يجوز بحال من الأحوال تغييره , ولعلك تجد نفعاً كبيراً من دالة *GetType* حين يأتى بك الكود إلى ما تم تعريفه بـ *Reflection* .

إذا أخذنا الشكل التالى , كمرجع لنا إن أردنا تفصيلاً , فأعتقد سيكون أمر يسيراً إن شاء الله , ويجب عليك ملاحظة كيف يسير إتجاه سير العملية من *Page* إلى *Object* :-

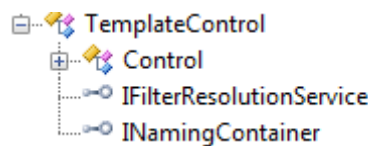


تأمل هذا الشكل ثم أكمل القراءة , ولناخذ أول جزء على حده , وهو *Page Class* , نجد هنا أن *Page* ترث من *TemplateControl* , وتقوم بعمل *Implement* لـ *IHttpHandler* وإذا جئنا للتفصيل نجد أن *IHttpHandler* هو المسئول عن معالجة الطلب القادم ولا يحتوى إلا على دالة واحدة وهى *ProcessRequest* وقد سبق الحديث عليه من قبل فى [الجزء الرابع](#) والذى تحدث عن *Modules & Handlers*.

ملحوظة :-

تستخدم كلمة *Inherit* مع الـ *classes* التى تقبل الوراثة , وكلمة *Implement* مع *Interfaces* وكذلك *Abstract Class* . ومعنى كلمة *Implement* , أى إضافة محتوى للدوال المأخوذة من الـ *Interface* .

ونجد أن *TemplateControl* هى التى تحتوى على الوظائف الأساسية للصفحة , وللعلم , *TemplateControl* تعمل كـ *Parent* لكلاً من *Page* و *UserControl* فقط . وإذا تابعنا المسير دخولاً إلى *TemplateControl* , سنجد أنها ترث من *Control Class* , حيث *Control Class* هى المصدر الرئيسى لإمداد جميع الأدوات بالوظائف الأساسية , ويخبرنا هذا أن الصفحة فى النهاية ما هى إلا أداة . وكما نرى من الشكل التالى :-



نجد أيضاً أن *TemplateControl* تقوم بعمل *Implement* لإثنين من الـ *Interfaces* وهما

IFilterResolutionService وهي المسئولة عن التعامل مع مختلف المتصفحات ,وتسمى Devices , حيث تساعد في توفير مخرجات (عناصر Html) تتناسب مع متصفحات دون غيرها . فلربما رأيت يوماً أن أحداً قام بضبط إثنين أو أكثر من الصفحات التي تعرف بإسم Master Page لصفحة ما , كل واحدة منهم مفترضاً لها أن تعمل مع متصفح مختلف , على هذا الشكل :-

```
<? Page Title="" Language="C#" MasterPageFile="~/Site1.Master"
IE:MasterPageFile="~/Site2.Master"
mozilla:MasterPageFile="~/Site3.Master"
AutoEventWireup="true" CodeBehind="WebForm2.aspx.cs" Inherits="WebApplication1.WebForm2" >
```

ما تجده هنا هو إعداد Master Page سيتم عرضها مع متصفح Internt Explorer ويشير إليها الاختصار IE والإخرى مع متصفح Fire Fox ويشير إليها الاختصار Mozilla و الثالثة عامة , اى مع غير ذلك . وننوه أن هذه الـ Interface لا يقوم المطور بإستخدامها , ولكن من يستخدمها هم مصصمى الأدوات .

ملحوظة :

يمكنك الحصول على إختصار المتصفحات واكوادها فى مجلد تجده داخل ملفات النظام الخاصة بـ .Net framework وأيضاً تستطيع أن تعرف قدرات وإمكانيات كل متصفح على حده من خلال إستكشافك لملفات هذا المجلد وهي ملفات ذات إمتداد browser. وهذا هو مسارها :

C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\Browsers

لدينا أيضاً Interface أخرى وهي INamingContainer, وهي المسئولة عن إنشاء Unique Id لكل عنصر بداخل الأداة التي تم إعلان أنها تقوم بعمل Implement لهذه الـ Interface. وكما تم الذكر فى درس سابق أنها Marker Interface اى لا تحتوى على دوال أو خصائص , هي فقط تخبر ASP.Net Runtime من ضرورة ضمان وجود ID مميز لكل أداة داخل الصفحة أو الأداة التي تحتوى على أدوات بداخلها وتقوم بإنتاج Namespace وهمى لكامل أدوات الصفحة .وبالتالى لا عجب من وجودها مع أى أداة تحمل بداخلها أدوات أخرى . بالطبع هناك بعض الأدوات التي تقوم أنت بتحديد Id لها , لكن إذا لم تقم بذلك؟ ماذا سيحدث حينها ؟! الإجابة , تتضح عند رؤيتك للشكل التالى :-

Control Tree		
Control UniqueID	Type	Render Size Bytes (including children)
__Page	ASP.dataaccesslayer_webform1_aspx	767
ctl02	System.Web.UI.LiteralControl	174
ctl00	System.Web.UI.HtmlControls.HtmlHead	32
ctl01	System.Web.UI.HtmlControls.HtmlTitle	19
ctl03	System.Web.UI.LiteralControl	14
form1	System.Web.UI.HtmlControls.HtmlForm	527
ctl04	System.Web.UI.LiteralControl	21
Button1	System.Web.UI.WebControls.Button	66
ctl05	System.Web.UI.LiteralControl	18
ctl06	System.Web.UI.LiteralControl	20

لاحظ أن كل عنصر بداخل الصفحة قد أخذ ID مميز, لن يأخذه غيره. كما ترى بالشكل, ولعلك قد رأيت هذا الأمر من خلال الكود فى الجزء الخامس من هذه السلسلة حين تكلمنا عن Compilation وكيفية تكوين الأدوات أثناء وقت التشغيل . وإن شاء الله سيأتى حديث حول

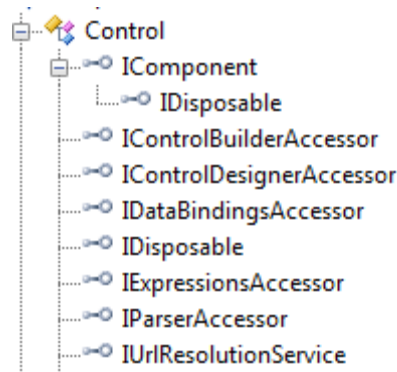
Rendering وما يفعله بـ ID إذا كانت الإداة Child لإداة أخرى .

ملحوظة:-

الصورة السابقة , قد تم التوصل إليها من خلال الـ Trace , والـ Trace هو تعقب العمل داخل صفحة واحدة أو الموقع ككل , فمن خلالها , يمكنك معرفة معلومات عن خط سير تنفيذ العمل , وذلك بمساعدة الـ Class التي تسمى بـ Trace وذلك من خلال الكود , أى أنك كمثال لديك دالة ما تريد أن ترى إن تم الدخول إليها أم لا , فتكتب هذا السطر ("We Are Inside Loop" Trace.Write) فيمكن تتبع الـ Trace لذلك , أيضاً , يمكنك معرفة معلومات عن الأدوات الموجودة في الصفحة وكم الحجم التي تأخذه أداة بعينها في كود HTML أو الحجم الذي تحجزه أداة في ViewState وكل هذا بالطبع يقاس بـ Byte . هناك الكثير مما قد يتطلب درساً كاملاً عنها وعن فوائدها , ولكن يبقى أن تعرف أنه يمكن ضبطها للصفحة الواحدة من خلال ضبط هذه الخاصية Trace=True في Page Directive , كما أنه يمكن ضبطها لكامل الصفحات من خلال Web.config من خلال العنصر trace ثم ضبط enabled=true بداخل system.Web .

```
<system.web>
  <trace enabled="true"/>
</system.web>
```

ننتقل إلى الجزء الثالث من الشكل الذي يوضح تسلسل Page Class , وهو Class بإسم Control , وهى الـ Class الرئيسية لكل الأدوات لدينا فى ASP .Net أى أن كل الأدوات لابد وأن ترث فى النهاية منها . حيث تحمل كل الوظائف المشتركة بين كل الأدوات . وبالتالي فإن جميع الخصائص والأحداث التي تشترك فيها كل الأدوات بما فيهم الصفحة فهى فى النهاية أداة , تجدهم جميعاً قد تم تعريفهم داخل هذه الـ Class . فيمكنك رؤية هذه الخصائص بعدة طرق منها على سبيل المثال , إستخدام نافذة Object Browser أو تتصفحهم من خلال رؤية الأكواد الداخلية للخصائص والحوال والأحداث , بإستخدام واحدة من أدوات De-Compilation أو كما تسمى Reflector وهى موجودة بكثرة عبر الإنترنت . أما يخص الـ Interfaces المتعلقة بـ Control Class وهى :-



فكما تراهم قد تتضح الواحدة من إسمها , فنجد أن IComponent فتوضح تعامل الأداة مع باقى الأدوات وتستعين بـ IDisposable والتي تأتى بالدالة Dispose والتي قد نستخدمها فى تحرير الذاكرة من بعض الـ Objects التي تم تعريفها داخل الكود .

وتجد أيضاً IControlBuilderAccessor تشارك فى بناء الأداة داخل الصفحة والأدوات التي قد تكون بداخلها . كذلك IControlDesignerAccessor , وهى المسئولة عن كيفية تعامل الأداة مع واجهة التصميم .

نجد أيضاً IControlBindingAccessor , وهى المعنية بأمور ربط الأداة بمصدر البيانات ,

أى تجعل الأداة تدعم Data Binding فى وقت التصميم . وأيضاً نجد IExpressionAccessor , تجعل الأداة تدعم التعامل مع Expression , الـ Expression هنا يقصد بها التعامل مع الخاصية Expression .

ملحوظة :-

الخاصية Expression تسمح بالحصول على بيانات من كلاً من عنصر ConnectionStrings و AppSettings وكلاهما موجودان فى ملف Web.Config , وأيضاً يمكنك الحصول على بيانات من ملفات Resources , تلك الملفات التى تدعم العمل بأكثر من لغة , ويطلق على هذه العملية بـ globalization . ومؤخراً , يمكن للأداة التعامل من خلال Expression مع القيم القادمة من Routing , والـ Routing بتعريف مختصر , هو التحكم فى شكل عنوان الصفحة ليظهر بشكل معين . ولعلك قد رأيت العلامات الدالة على Expression وهى علامات Sever tags بجانبها علامة \$ مثال:- <%\$ %> : %> وأخير يجب عليك أن تعرف أنه يمكنك أن تقوم بعمل Expression خاص بك , من خلال القيام بعمل Implement ExpressionBuilder ومنها إلى ExpressionBuilder Compilation

وأخيراً , IParserAccessor و IUrlResolutionService , فالأولى تساعد الأداة على معرفة متى يتم ترجمة الأدوات التى بداخلها وتحويلهم إلى HTML . والثانية , تقوم بحل المسار , ولتقرب إليك الصورة , لعلك قد رأيت عنواناً بهذا الشكل :- ~img1.jpg و نقصد خاصة العلامة ~ , ماذا يحدث أثناء وقت التشغيل أو حتى وقت التصميم , يتم حل هذا المسار الوهمى بالمسار الفعلى للصورة , أى مسار التطبيق والمجلد الذى توجد فيه هذه الصورة . وهذا ما يطلق عليه Resolve Url .

أبرز الخصائص والدوال داخل Page Class

نجد مبدئياً خصائص التعامل مع الواجهة والتصميم , مثال Theme والتى تختص بتنسيقات الصفحة , ونجد بداخلها أو متعلقاً بها ما نعرفه بـ Skin وكذلك StyleSheet وهو المعروف لدى الجميع بإسم CSS , وبالطبع الصور أيضاً , فالكلى يتم وضعهم داخل مجلد يسمى App_Themes . والجميع يعمل لهدف واحد , وهو تنسيق الصفحة . و لا نغفل عن ذكر الخاصية MasterPageFile والتى تتيح لنا ضبط الصفحة الأم من حيث المظهر . وذكرنا أيضاً منذ قليل Skin , وهو الخاص بضبط تنسيق معين لنوعية محددة من الأدوات , كضبط تنسيق لـ Button ما , فنجد أن التنسيق سيتم تطبيقه على كل button فى الصفحة إلا إذا قمنا باستخدام الخاصية SkinId والتى تعتنى بضبط التنسيق المحدد لأداة أو أكثر على حسب ما نحدد من خلال هذه الخاصية .

كذلك الخاصية Master والتى يمكنك من خلالها الوصول إلى صفحة Master Page أثناء وقت التشغيل (Run-time) . وأيضاً الخاصية Page والتى يمكنك من الوصول برمجياً إلى الصفحة الحالية , وأخيراً الخاصية PreviousPage والتى من خلالها يمكننا الوصول إلى الصفحة السابقة وتصفح محتوياتها برمجياً , أى من خلال الكود .

ولعلنا ذكرنا من الخصائص أيضاً فى الجزء الثانى حين تكلمنا عن Request و Response و Server و Trace ونزيد هنا أموراً تخص حفظ حالة الصفحة كـ ViewState و Profile و Session و Application Cache و Cookies , حيث تسمح لك ASP.Net أن تقوم بحفظ حالة الصفحة من بيانات فى هذه المستودعات (أو ربما نطلق عليها أماكن تخزين) , كذلك الخاصية User والتى تسمح لك بمعرفة إسم المستخدم الحالى , ونوعية تسجيله , وكذلك للكشف عن إذا ما قام الزائر الحالى بالتسجيل من خلال الخاصية User.Identity.IsAuthenticated

والتي يمكن أن ننفذ بعض المهام بناء على ذلك أم لا كتنفيذ كود فقط للأعضاء الذين قاموا بتسجيل الدخول.

كذلك نجد خاصية تسمى Validators والتي من خلالها يمكنك الوصول إلى كل أدوات الـ Validation الموجودة في الصفحة , تلك الأدوات التي تتأكد من صحة المدخلات , حيث توفر لك طريقة آمنة للتأكد أن المدخلات تتوافق مع ما قمت بتحديدك , وأيضاً قد تحتاج إلى بعض الأدوات وليس الكل , ففي هذه الحالة قد تفيدك دالة GetValidators والتي تتيح لك أن تطلب مجموعة من أدوات الـ Validation دون الكل , وذلك من خلال تحديد إسم المجموعة GroupName وسيأتي إن شاء الله شرحاً مفصلاً لكيفية عمل هذه الأدوات وما السر وراءها .

وحين نأتي للحديث عن دعم ASP. Net لإظهار محتوى بأكثر من لغة , فإننا لاشك سنذكر خاصيتين هما Culture و UICulture. حيث نقوم بتحديد اللغة التي نريد إظهار المحتوى بها من خلال هاتين الخاصيتين . ويمكنك الإستزاده من خلال البحث في موضوع بإسم Localization and Globalization . ونذكر أيضاً أن هاتين الخاصيتين يمكن ضبطهما من خلال إستخدام Page Directive أو من خلال code-Behind للصفحة الواحدة , أو ضبطهما لكامل صفحات الموقع من خلال Web.config من خلال العنصر Pages داخل System.Web.

كذلك عند الحديث عن معالجة الأخطاء , فإننا سنذكر خمسة طرق لمعالجة الخطأ على مستوى الصفحة والتطبيق ككل , من هذه الخمسة , خاصية تسمى Error Page , وما عليك إلا أن تكتب إسم صفحة الخطأ وبالطبع مصحوبة بالمسار والإمتداد من خلال هذه الخاصية. أما الأربعة الآخرين فهم , حدث Page_Error وحدث Application_Error والخاصية التابعة لملف Web.config وهي Custom Error . وأخيراً نذكر عمل Override للدالة OnError الخاصة بالصفحة , ولهم ترتيب يجب عليك معرفته , ونرجو الله أن يبسر لنا ويأتي الوقت المناسب لنبينه إن شاء الله .

أيضاً مما لاشك فيه أن معرفتك لخصائص الفحص مثال IsValid أو IsPostBack أو IsCallback أو IsAsync أو IsCrossPagePostBack أو IsViewStateEnabled وغيرهم القليل , سيحدد ماذا يمكن أن تفعله , أي سيحدد ردة فعلك لأمر ما قد حدث , فنجد أن , عند حاجتك لتنفيذ بعض المهام في أول مرة يقوم المستخدم بتشغيل صفحة ما , سيكون بلا شك بعد فحص الخاصية IsPostBack والتأكد من أنها تعود بالقيمة false , وذلك تجنباً لتكرار التنفيذ مع كل مرة يحدث فيها Postback , أو لربما أتى المستخدم إلى الصفحة الحالية عن طريق Postback من صفحة أخرى , وتريد أن تحصل على معلومات من الصفحة السابقة , فمن الأفضل أن نقوم بفحص الخاصية IsCrossPostBack والتأكد منها أنها تعود بالقيمة True , ومن ثم نصل إلى بيانات الصفحة السابقة من خلال الخاصية PreviousPage , كذلك الخاصية IsValid والتي نفحصها عن الحاجة إلى التأكد من أن أدوات الـ Validation قامت بعملها كما أعد لها . أيضاً نذكر أن IsAsync تستخدم عند ضبط الصفحة لتعمل في وضع Asynchronous وهذا الوضع يستخدم عند أداء الصفحة الحالية لمهام تستغرق الكثير من الوقت , لذلك نقوم بضبط الخاصية Async إلى True وذلك في Page Directive ومن ثم العمل مع دالة AddOnPreRenderCompleteAsync وضبطها في حدث Page Load ومن ثم ضبط المهمة التي نريد . وأخيراً نستخدم IsCallback عند دعم الصفحة لإستدعاء دوال موجودة على الـ Server ليتم تنفيذها من خلال الـ Client أي من خلال JavaScript وفي هذه

الحالة تقوم الصفحة بعمل Implementg لـ Interface تسمى ICallbackEventHandler والتي تحتوى على دالتين أحدهما لإعطاء البدء بالتنفيذ والآخرى للحصول على النتائج , وبالمطبع لا شك أن هناك اكواد تكتب من خلال JavaScript والهدف من هذه العملية , هو , الإستغناء عن حدوث PostBack .

الخاصية ClientScript وما يتعلق بها من خصائص ودوال أخرى تتلخص في العمل مع كود يعمل على جهاز المستخدم وفي الواقع هو كود JavaScript , وسيأتى تفصيلاً إن شاء الله الحديث عن أكواد ClientScript .

وأخيراً الخاصية Controls , والتي من خلالها نستطيع الوصول بل وإضافة وحذف أدوات من وإلى الصفحة ويساعدها في العمل دالتين , لا نستطيع التعامل عنهما لإهميتهما , وهما دالة LoadControl والدالة الأخرى هي FindControl . فالأولى نستطيع بها تحميل (إن شئت قل إدراج) أدوات داخل الصفحة , وفي العادة نستخدمها لإدراج User Control أثناء وقت التشغيل . والدالة الأخرى , نستخدمها للبحث داخل الصفحة أو ربما داخل الأداة عن أداة أخرى , مثال البحث داخل الصفحة أو داخل GridView عن Button , ولعلك تتعجب لماذا أحتاج إلى أن أبحث وأنا أستطيع أن أصل إلى هذه الأداة من خلال ID الخاص بالأداة , كأن نقول Button1.Text = "Click Me" . نقول وبالله التوفيق , أنه في بعض الأوقات , لا نستطيع رؤية أداة من الأدوات , أو ربما مجموعة من الأدوات بشكل مباشر من خلال الكود , فتكون غير متاحة إلا أثناء وقت التشغيل , مثال , وجود TextBox سيظهر عند الدخول إلى وضع التحرير الخاص بـ GridView , ونريد أن نحصل على القيمة التي بداخله , عندها يمكن أن نبحث باستخدام هذه الدالة (FindControl) عن هذا الـ TextBox , ولإكمال المعرفة لديك , يجب أن ننوه أنه لا بد من إختيار الوقت والمكان الذي نبحث فيه عن الاداة المراد إيجادها , فلو أخذنا في الاعتبار المثال السابق , نجد أنه متاح لديك أن تبحث داخل بعض أحداث GridView , ولمعرفة ذلك الوقت والمكان الذى نتحدث عنه لابد لك من معرفة دورة حياة الصفحة ثم معرفة دورة حياة الأدوات حتى تتمكن من إختيار الوقت والمكان الأفضل أو ربما الأصح .

دالة أخرى قد تكون متعلقة بالفقرة السابقة , وهى RenderControl . وتستخدم لإخراج محتوى صفحة بكاملها أو أداة

داخل تلك الصفحة على شكل HTML بهدف تصديره , مثال , تصدير الصفحة إلى PDF أو ربما DOC أو XLS أو غيرهم . أما الدالة الأخرى المشابهة لها , وهى دالة Render فستأتى معنا إن شاء الله في دورة حياة الصفحة بعد قليل .

دالة Eavl و XPath , ويتم إستخدامهما في عملية Binding , تلك العملية التى تشير إلى ربط مصدر البيانات بالأداه المستهدفه . فالأولى تستخدم مع قواعد البيانات التى تحتوى على أعمدة وصفوف , والثانية تستخدم مع ملفات XML والتى كما نعرف تحتوى على عناصر تتمثل في Tags .

الدالة الثالثة المتعلقة بهما أيضاً , هى دالة DataBind وهى التى تعلن إشارة البدء في عملية الربط التى تتم بين الاداة ومصدر البيانات .

ملحوظة :-

قد أزيد من كم المعلومات في مكان وأنقصه في مكان آخر , وذلك حسب ما تقتضيه الحاجة , ولعلنى أستفيض في تلك الامور التى لا يعرفها الكثيرون منا . وأميل إلى التلخيص وإظهار الفكرة العامة في أمور قد فاضت بها المواقع العربية ذرعاً .

فى نهاية الحديث عن الخصائص , قد أكون أغفلت الحديث عن بعضها ولكن هذا ما يسر الله لنا أن نتحدث عنه فيما يخص خصائص ودوال الصفحة وإليك حديثاً عن الأحداث فيما يخص الصفحة , والله المستعان .

رحلات الذهاب والإياب الخاصة بـ **HttpRequest**

حين نتكلم عن رحلات الذهاب والإياب , فلنضع فى الاعتبار أن هناك رحلة تخص التطبيق ككل , والأخرى تخص الصفحة الواحدة . فالأولى تسمى Application Life Cycle والأخرى تسمى Page Life Cycle .

أما ما يخص Application Life Cycle فقد رأيتها فى جزء سابق , وأما ما يخص Page Life Cycle فأعلم أنها تبدأ من بداية إنشاء الصفحة وقت التشغيل حتى إسترجاع Html المكون للصفحة التى سيتم عرضها للمستخدم فى المتصفح .

ملحوظة :

كما أن للتطبيق دورة حياة وكذلك للصفحة دورة حياة فإن الأدوات التى هى نواة الصفحة أيضاً لها دورة حياة وتتغير هذه الدورة بتغير نوعية الأداة التى تعمل معها .

ماذا نقصد بدورة حياة التطبيق أو الصفحة أو حتى الأدوات ؟

دورة الحياة , هنا , يقصد بها مجموعة من الأحداث والدوال التى يتم تنفيذها فى ترتيب ثابت تم تحديده من قبل , ومن قام بتحديده هم مهندسوا تطوير ASP. Net . أى أنك لا تملك تغير ترتيب هذه الأحداث والدوال . ولكن ما يمكنك فعله , هو , أن تستخدم هذه الأحداث لفعل شيئاً مفيداً .

وليكن فى حسابنا أن هذا الترتيب يصاحبه بعض القيود (حين تفهم دورة الحياة ستجد هذه القيود منطقية جداً) . بمعنى أنك لن تستطيع فعل شئ إلا فى وقته المسموح لك به . مثال ذلك , تغير اللغة المستخدمة فى عرض المحتويات للصفحة , نجد أن لها وقتاً معيناً فى هذه الدورة . لذلك لا تحاول أن تقوم بتغير اللغة فى غير المكان المحدد . مثال آخر , هل تعلم أن الصفحة يتم تدميرها وإعادة إنشاؤها فى كل طلب للصفحة , ولعلك تعرف أن مكونات هذه الصفحة تحتوى على بعض الأدوات , فهذه الأدوات يتم إنشاؤها فى وقت معين فى الدورة , فلا حاجة لإتعب نفسك بالمحاولة فى الوصول إلى هذه الأدوات , وقد تكون لم يتم إنشائها بعد , أى أن هذه الأدوات لا توجد فى الوقت الحالى , وبالتالي ستكون غير متاحة . فتحتاج ان تعرف الحدث الذى يتم فيه إعادة إنشاؤها من جديد , حتى يمكنك إستخدامها .

ملحوظة:

تختلف دورة الحياة عند حدوث Postback , فنجد أن بعض الأحداث لا يتم تنفيذها إلا فى حدوث PostBack , فهذه الأحداث الجديدة تخص الأداة المسؤولة عن التسبب فى حدوث PostBack . وهناك أيضاً بعض الدوال التى يتم تنفيذها فى حالة PostBack , والتى بدورها تحافظ على تحديث حالة التطبيق , والتأكد من صحة البيانات المدخلة أو ما ستعرفه بعد بإسم Validation .

ملحوظة :

عند عمل تحديث جزئى للصفحة بإستخدام Ajax كما ذكرنا من قبل أيضاً يتم تنفيذ نفس الأحداث بنفس الترتيب , وبدون غياب واحد منهم , كما لو قمت بتحديث الصفحة ككل .

دورة حياة الصفحة :

يتم تقسيم هذه الأحداث فى عدة مراحل وعددها سبعة مراحل :-

– مرحلة طلب الصفحة :-

- وفيها يتم البحث عن الصفحة , ورؤية إذا ما كانت الصفحة قد تم ترجمتها ووضعها داخل Cache بداخل ملف dll , أم أن هذه هي المرة الأولى التي يتم طلب الصفحة بها أو ربما تم التعديل في أكوادها , فعند ذلك يحتاج إلى ترجمتها ومن ثم تخزينها في Cache للطلبات القادمة , وقد تم الإشارة إلى هذا الأمر في الجزء السابق من هذه المقدمة بعنوان Compilation .

- مرحلة البدء :-

- وفيها يتم ضبط خصائص الصفحة كـ Request و Response , وإعدادها من أجل العمل لمعالجة الطلب , كذلك لابد من تحديد نوعية الطلب للصفحة , هل هي لأول مرة من هذا المستخدم , أى باستخدام الأمر Get أم أن الطلب يأتي لمرات متتالية من نفس المستخدم , أى أنه Postback , وذلك للأخذ في الاعتبار تنفيذ الأحداث الخاصة بـ Postback منها على سبيل المثال تحديث البيانات الموجودة بداخل ViewState , وكذلك تنفيذ الحديث المسئول عن إحداث هذا الـ Postback .

- مرحلة إعطاء القيم الابتدائية :-

- كما ذكرنا أن الصفحة ما هي إلا Object من Class يتم إنشائه وتدميره في كل طلب لهذه الصفحة , فإنه من المنطقي أن يكون لهذا الـ Object قيمة ابتدائية يبدأ بها عمله . وعلى هذا يتم إنشاء الأدوات بداخل هذه الصفحة والبدء بإعطائها القيم الابتدائية . ومن هذه القيم الابتدائية , تغيير اللغة المستخدمة في عرض محتويات الصفحة , كذلك ضبط صفحة MasterPage وأيضاً ضبط Theme وكذلك إنشاء أى أدوات أثناء وقت التشغيل ويفضل هذا المكان لتيسير إمكانية معالجة أحداث هذه الأدوات .

- مرحلة تحميل الصفحة وأداتها :-

- في هذه المرحلة في حالة أن الطلب قادم باستخدام Postback يتم تحديث خصائص الأدوات باستخدام البيانات القادمة من المتصفح على هيئة ViewState , وكذلك معالجة الطلب وتجهيز الصفحة لعملية إرسال محتوى Html إلى المتصفح .

- مرحلة معالجة طلب المستخدم :-

- في حالة أن الطلب قادم من خلال Postback , فإن ذلك يتطلب تنفيذ الحدث الذي تسبب في حدوث Postback , وغالباً ما يتعلق بأحد أدوات الصفحة ليتم به تحديث حالة الصفحة أو ربما إجراء بعض العمليات داخل الموقع من خلال هذا الحدث . ولكن قبل هذا يتم الفحص أولاً , إذا ما كانت الصفحة تحتوي على أى من أدوات التحقق من الصحة المعروفة باسم Validation , فإذا وجد أحد هذه الأدوات فيتم تنفيذ دالة تسمى Validate لتقوم بإجراء عملية التحقق من خلال الخادم .

- مرحلة إخراج محتوى أكواد Html للمتصفح :-

- تطلق على هذه المرحلة اسم Rendering , وهي عملية تحويل كافة أدوات الصفحة التي هي في الأصل Objects قد تم إنشائها من عدة Classes إلى أكواد Html ليتم إرسالها إلى المتصفح الذي لا يفهم إلا Html . ولكن قبل هذا يتم حفظ الحالة الحالية للصفحة وأدواتها المتمثلة في ViewState , وتتم عملية الإخراج هذه من خلال استدعاء دالة Render لكل أداة داخل الصفحة ومن ثم للصفحة نفسها .

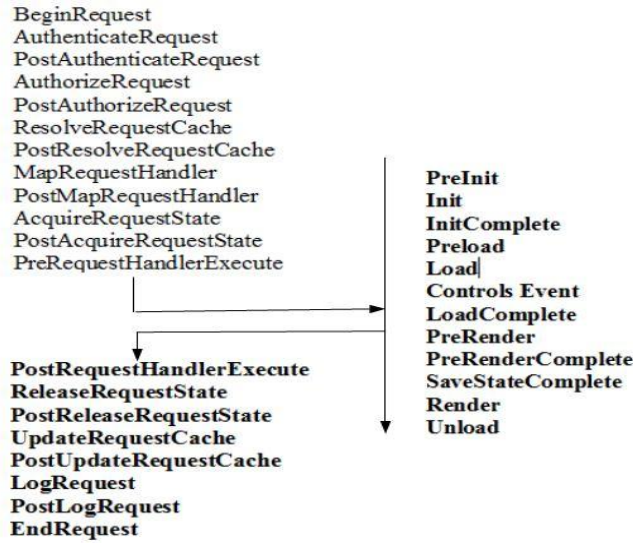
- مرحلة تحرير الذاكرة :-

- هنا يتم تحرير الذاكرة من موارد الصفحة , وتفرغ الخصائص مثل Response و Request لذلك لا يمكن إستخدامهم في هذه المرحلة , لأن الصفحة قد تم إرسالها إلى المتصفح , وتعتبر هذه المرحلة بمثابة مرحلة تدمير للصفحة ليعاد إنشائها عند الطلب القادم لنفس الصفحة .

داخل هذه المراحل تكمن دورة حياة الصفحة , وتتمثل في مجموعة من الأحداث , والأحداث بطبيعتها هي دوال يتم تنفيذها في وقت وترتيب معين , فالحدث ما هو إلا دالة لها شكلاً معيناً قد تم ضبطه باستخدام ما يعرف بـ Delegate . فيما يلي سنقوم إن شاء الله بعرض لهذه الأحداث

أحداث الصفحة

تعد دورة حياة الصفحة جزء من دورة حياة الطلب (Request) التي قمنا بدراستها من قبل في الجزء الثالث , وبالتحديد تبدأ عند الحدث PreRequestHandlerExecute , والترجمة لهذا الحدث " ما قبل تنفيذ معالج الطلب " , وتنتهي عند الحدث PostRequestHandlerExecute , والترجمة لهذا الحدث " ما بعد تنفيذ معالجة الطلب " . نعم , مما ذكرنا في الأجزاء السابقة , نجد أن Page Class , هي معالج الطلب , أى هي ما ستقوم بالرد على الطلب القادم من المستخدم , وهنا يتمثل المعالج في ملف Code-behind وبه نجد مجموعة من الأحداث , تمثل هذه الأحداث بدورها دورة حياة الصفحة وتبدأ كما بالشكل .



في الحقيقة لا تقتصر دورة حياة الصفحة على هذه الأحداث فقط , بل هناك العديد من الدوال التي يتم تنفيذها في ترتيب قد تم إعداده مسبقاً ليؤدي دوره في خدمة إنشاء الصفحة وأداء مهامها على أكمل وجه .

نذكر من هذه الدوال, دالة تسمى InitializeCulture , تلك الدالة التي تقوم بإعلان تحديد اللغة المستخدمة في عرض الصفحة , ولعلك يوماً قد تعاملت مع هذه الدالة في تغيير اللغة لصفحات موقعك .

وجدير بالذكر هنا, أن نشير إلى الخاصية AutoEventWireUp والتي تساعد في تعرف الصفحة على أحداثها , فحيث يقوم بالبحث عن كل حدث يبدأ بكلمة Page ثم _ ثم إسم الحدث , أى على هذا الشكل Page_EventName , فإذا كانت هذه الخاصية تحمل القيمة false , فهناك مشكلة , لأن الصفحة لن تتعرف على الأحداث المعروفة , وتحتاج حينها أن تخبر الصفحة بإسماء الأحداث ومثال على ذلك أن تقوم بعمل التالي :-

```

public partial class _Default : System.Web.UI.Page
{
    public _Default()
    {
        this.Load += new EventHandler(PageLoading);
    }
    protected void PageLoading (object sender, EventArgs e)
    {
        Response.Write("Hello World! ");
    }
}
  
```

كما تلاحظ فإننا قد قمنا باستخدام Page Constructor لنبلغه بالأحداث الى يجب على الصفحة تنفيذها .

وإليك تفصيلاً لكل حدث على حدث :-

Page_PreInit

يعتبر هذا الحدث هو أول أحداث الصفحة , وبه يتم الإعداد لإعطاء القيم الابتدائية , فيمكننا بداخل هذا الحدث أن نقوم بضبط الخاصية Theme و MasterPageFile .

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        this.Theme = Request.QueryString["theme"] ?? "Navy";
    }
}
```

Page_Init

وهنا من المفترض أن كل أدوات الصفحة قد أخذت القيم الابتدائية ومما لاشك فيه أن كل حدث Init لكل الأدوات وكذلك الحدث Init الخاص بـ Master Page جميعهم قد تم تنفيذهم قبل الحدث Init الخاص بالصفحة , أى تنفيذ الحدث Init الخاص بكلاً من الأدوات ثم Master ثم الصفحة على نفس الترتيب.

Page_InitComplete

وهنا نهاية مرحلة إعطاء القيم الابتدائية لينتقل منها إلى مرحلة التحميل وبدء عملية Loading , ولكن عليك أن تعرف أن ما بين هذا الحدث والحدث السابق يتم تعقب التغيرات التى ستحدث فى ViewState , ولهذا يمكنك التفاعل مع ViewState بدايةً من هذا الحدث , ولكن لاحظ أنه لم يتم تحميل البيانات المخزنة فى ViewState مسبقاً , لذلك لا تحاول إستدعائها .

Page_PreLoad

هنا قد تم تحميل البيانات القادمة من ViewState وذلك بعد تنفيذ الدالة LoadViewState وبها يتم تحميل بيانات ViewState للصفحة ولأدواتها , كذلك جلب البيانات القادمة من form الخاصة بتحديثات PostBack .

Page_Load

يتم تنفيذ الحدث Load للصفحة ومن ثم الحدث Load للصفحة Master وبعدها الحدث Load لأدوات الصفحة , ويمكن فيه أن تتفاعل مع الصفحة بجميع أدواتها لضبط لخصائص والعمل مع قواعد البيانات وغيرها من العمليات اليومية .

أحداث الأدوات (Control Event)

إن كان الطلب الحالى من نوع Postback , فلا شك أن هناك أداة قد تسببت فى حدوث هذا الـ Postback وهناك حدثاً ينتظر التنفيذ وهو بالفعل الحدث الذى تسبب فى حدوث Postback مثال Click أو ربما CheckChanged أو SelectedIndexChanged أو غيرهم , فهنا فى هذا الوقت يأتى دوره للتنفيذ .

Page_LoadComplete

بهذا الحدث تكتمل عملية تحميل الصفحة وجميع أدواتها ويمكنك التواصل بين الأدوات وبعضها .

Page_PreRender

وهنا يتم الإعداد لإخراج محتوى HTML لكافة عناصر الصفحة والصفحة نفسها , حيث يتم إستدعاء الحدث PreRender للصفحة ومن ثم لصفحة Master وبعدهما للأدوات على هذا الترتيب المذكور .

وهنا عليك بالتفاعل الأخير مع الصفحة وأدواتها قبل أن يتم تحويلهم إلى HTML إستعداداً لإرسالهم إلى المتصفح .

PreRenderComplete

هذا الحدث يعلن أنه بالفعل قد تمت عملية الإعداد لإخراج محتوى HTML, وقد تمت عملية الربط بين الأدوات ومصدر البيانات .

وبعد هذا الحدث مباشرة يتم تنفيذ الدالة SaveViewState وهى المسئولة عن حفظ البيانات بداخل ViewState ولاحظ أن هذه الدالة يتم تنفيذها فقط فى حالة حدوث PostBack .

Render

وهذه ليست حدثاً, إنما هى دالة , وهى المسئولة عن إخراج محتوى HTML, حيث تملك كل أدوات ASP.Net المسماة Server Controls دالة تسمى Render, وبها يتم إخراج محتوى HTML لكل الأدوات, وهى الدالة نفسها التى نستخدمها فى التفاعل معها حين نقوم بإنشاء أداة جديدة من إنشائنا, فقد تحتاجها حين تقوم بإنشاء أداة خاصة بك .

Page_Unload

هذا الحدث يعلن نهاية التفاعل مع الصفحة, فهنا يتم تفريغ الذاكرة من الصفحة وأدواتها , حيث لا يمكنك إستخدام Request أو Response فى هذا الحدث , لإنهما لم يعد لهما وجود, ويمكنك إستخدام هذا الحدث لتفريغ الذاكرة من أى Object قمت بالتفاعل معه , كإنهاء عمليات التفاعل مع قوعد البيانات وغيرها .

نهاية هذا الدرس بالتوفيق للجميع إن شاء الله .