Creating HttpModules in ASP.NET

Created on *19 août 2007*
Edited on *20 août 2007*
By Pierre-Emmanuel Dautreppe
Reviewed by

1 Summary

1.1 Table of content

1.2 Table of sources

2 Introduction

A few days ago, Loïc Bar (http://www.loicbar.com) has posted a new feed to explain how to customize an ASP.NET application using master pages (and how to store this value in the user profile).
As added in the feed comments, I wanted to react to go a bit further.

Indeed – as explained in his feed – to change of master page, we need to handle the Page.PreInit event. But of course, to propagate this change to all website, this should be done on all pages, which is a bit unhandy. Several solutions exist to avoid doing a treatment on all pages:

- Doing the job on the master page used by the related pages
    - This solution is not applicable here because the master pages do not expose a PreInit event. The reason is simple : this event is the only place where we should change of master pages so of course.
- Creating a common base class (inheriting of System.Web.UI.Page) and making all the other pages of the web site inheriting of this new page. This new page would do the job by handling the PreInit event for all pages.
    - This is not a very elegant solution as we'll need to update all the pages we have already created in the web site – and of course when the client ask for customization, you may have already created dozens of pages
    - Moreover this kind of solution is risky as a real project would deal with many developers and many maintenance guys (from the same company or not). And we are never "indelicate developers-proof". As a consequence, it can happen (and so it WILL happen) to have some pages not inheriting from the base class
    - Anyway it forces us to use a new level of inheritance all over the website which can be avoided

Anyway, we are now dealing with a transversal functionality and so this must be dealt as a transversal development and implementation. This is typically AOP: Aspect Oriented Programming.
In .NET, there are several ways to achieve AOP: let's point out context bound objects and HttpModule.

In this case HttpModules are the most appropriate choice and we'll see how to use them to complete Loïc's job on web sites customization.

3 What is an HttpModule?

HttpModules are classes that are plugged into the ASP.NET request pipeline and so that will get called on each request.
Typically, .NET register some by default (you can see the .NET default web.config). See below the main modules .NET uses:

- Security and authentication
  - WindowsAuthenticationModule
  - FormsAuthenticationModule
  - PassportAuthenticationModule
  - AnonymousIdentificationModule
- Security and authorization
  - FileAuthorizationModule
  - UrlAuthorizationModule
  - RoleManagerModule
- Other transversal functionalities
  - Session management
  - Error handling
  - Caching
  - Profile management

HttpModules can be plugged wherever we need in the request pipeline, ie at the beginning of the request, during authentication, during authorization, …

4 Creating an HttpModule

4.1 Basic implementation of a HttpModule

An HttpModule is a simple class implementing the interface IHttpModule.
In our case, in our solution we will:

- Create a new ClassLibrary (let's name it HttpModule)
- Adding a reference to System.Web.dll
- Adding a new class (let's name it MasterPageManagerModule)

We want to handle the PreInit event of the Page to do some treatment. To do so, we will access the HttpContext.CurrentHandler property that will give us the current IHttpHandler of the current request. If this handler is a Page, we can do our work.
Note that we need to have the request handler available so we'll plug us into the "`PreRequestHandlerExecute`" event.
You could replace the anonymous method by a simple event handler if you want to run in .NET 1.x.

```
using System;
using System.Web;
using System.Web.UI;

namespace HttpModule
{
   public class MasterPageManagerModule : IHttpModule
   {
      #region IHttpModule Members

      public void Dispose()
      {
         // No speacial treatment to do in the Dispose method
      }

      public void Init(HttpApplication context)
      {
         context.PreRequestHandlerExecute += delegate(object sender, EventArgs e)
            {
               /* We want to plug our module in the Page.PreInit event
                * Note that the pages are just a special kind of HttpHandlers
                */
               Page page = HttpContext.Current.CurrentHandler as Page;
               if ( page != null )
                  page.PreInit += new EventHandler(Page PreInit);
            };
      }

      #endregion

      private void Page PreInit(object sender, EventArgs e)
      {
         // To do later
      }
   }
}
```

Step 1 : Basic Implementation of a HttpModule

4.2 Raising an event

4.2.1. Creating a custom EventArgs

In our case, we won't let the module itself changing the master page but we'll give the developer using the HttpModule the ability to manage that. Indeed the master page change can occur for different reasons:

- Default value
- Value overridden in the profile
- Value passed as a query string
- …

So the best for us is to raise an event, requesting for the new MasterPage to use.
We'll need for that to create our own event and our own EventArgs. Here we propose a very basic implementation that will provide the user just with the current master page file and let him update this property. We won't provide him with any extra information as he could retrieve them from the HttpContext.Current property by himself.

```
using System;
using System.Web;

namespace HttpModule
{
   public class MasterPageRequestingEventArgs : EventArgs
   {
      private string masterPageFile;

      /// <summary>
      /// Gets / Sets the master page file for the page handling the current request
      /// </summary>
      public string MasterPageFile
      {
         get { return masterPageFile; }
         set { masterPageFile = value; }
      }

      /// <summary>
      /// Initialize a new MasterPageRequestingEventArgs
      /// </summary>
      /// <param name="currentMasterPageFile">The master page file that is currently
      /// being used by the page handling the request
      /// </param>
      public MasterPageRequestingEventArgs(string currentMasterPageFile)
      {
         this.masterPageFile = currentMasterPageFile;
      }
   }
}
```

Step 2 : Creating an event args for our module

4.2.2. Raising the event in our module

We can now update our module to raise the event.

```
using System;
using System.Web;
using System.Web.UI;

namespace HttpModule
{
   public class MasterPageManagerModule : IHttpModule
   {
      /// <summary>
      /// Our module will raise an event when we can provide the master page
      /// </summary>
      public event EventHandler<MasterPageRequestingEventArgs> Requesting;

      #region IHttpModule Members
      // Implementation unchanged
      #endregion

      private void Page PreInit(object sender, EventArgs e)
      {
         Page page = sender as Page;

         if ( Requesting != null )
         {
            MasterPageRequestingEventArgs args
                 = new MasterPageRequestingEventArgs(page.MasterPageFile);
            Requesting(this, args);
            if ( string.Compare(page.MasterPageFile, args.MasterPageFile, true) != 0 )
               page.MasterPageFile = args.MasterPageFile;
         }
      }
   }
}
```

Step 3 : Updating our HttpModule to raise an event

5 Handling the event to change the MasterPage

5.1 Registering the HttpModule

We are now ready to use our HttpModule. The first thing to do is to register it in the web.config file. First we'll add a reference to our DLL in the web site and then we can update the web.config as seen below.

```
<system.web>
        <httpModules>
                <add name="MasterPageManager"
                     type="HttpModule.MasterPageManagerModule, HttpModule"/>
        </httpModules>
</system.web>
```

Step 4 : Updating the web.config file to register the HttpModule

Note that the name attribute is up to the developer that is using the HttpModule.

5.2 Using the HttpModule in the Global.asax

Now that the HttpModule is registered, we can update the global.asax file to handle it.
You must know that the events handled in the global.asax are wired up depending of the name. For HttpModule, we must follow the convention:

HttpModuleName_EventName

where "HttpModuleName" is the name as specified in the web.config and "EventName" is simply the name of the event as declared in the class.

```
<%@ Application Language="C#" %>
<%@ Import Namespace="HttpModule" %>

<script RunAt="server">
   void MasterPageManager Requesting(object sender, MasterPageRequestingEventArgs e)
   {
      /* Here you can now provide the name of the master page based on the profile,
       * the query string, some post parameters, …
       */
      e.MasterPageFile = "~/MasterPage2.master";
   }
</script>
```

Step 5 : Updating the global.asax to handle the module's event

6 Conclusion

Now – without modifying any of your pages – you can dynamically manage your web site customization and master pages in a central location.
Always identify transversal functionalities and always try to provide in such a case a central development that will let all your code unchanged.