ASP.NET Master Pages

ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

This overview contains the following sections:

How Master Pages Work

Master pages actually consist of two pieces, the master page itself and one or more content pages.

Note
You can also nest master pages. For details, see Nested ASP.NET Master Pages.

Master Pages

A master page is an ASP.NET file with the extension .master (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special @ Master directive that replaces the @ Page directive that is used for ordinary .aspx pages. The directive looks like the following.

C#

```
<%@ Master Language="C#" %>
```

The **@ Master** directive can contain most of the same directives that a @ Control directive can contain. For example, the following master-page directive includes the name of a code-behind file, and assigns a class name to the master page.

C#

```
<%@ Master Language="C#" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

In addition to the **@ Master** directive, the master page also contains all of the top-level HTML elements for a page, such as **html**, **head**, and **form**. For example, on a master page you might use an HTML table for the layout, an **img** element for your company logo, static text for the copyright notice, and server controls to create standard navigation for your site. You can use any HTML and any ASP.NET elements as part of your master page.

Replaceable Content Placeholders

In addition to static text and controls that will appear on all pages, the master page also includes one or more ContentPlaceHolder controls. These placeholder controls define regions where replaceable content will appear. In turn, the replaceable content is defined in content pages. After you have defined the ContentPlaceHolder controls, a master page might look like the following.

C#

```
<%@ Master Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server" >
    <title>Master page title</title>
</head>
<body>
    <form id="form1" runat="server">
        <table>
           <tr>
               <td><asp:contentplaceholder id="Main" runat="server" /></td>
               <td><asp:contentplaceholder id="Footer" runat="server" /></td>
           </tr>
        </table>
    </form>
</body>
</html>
```

Content Pages

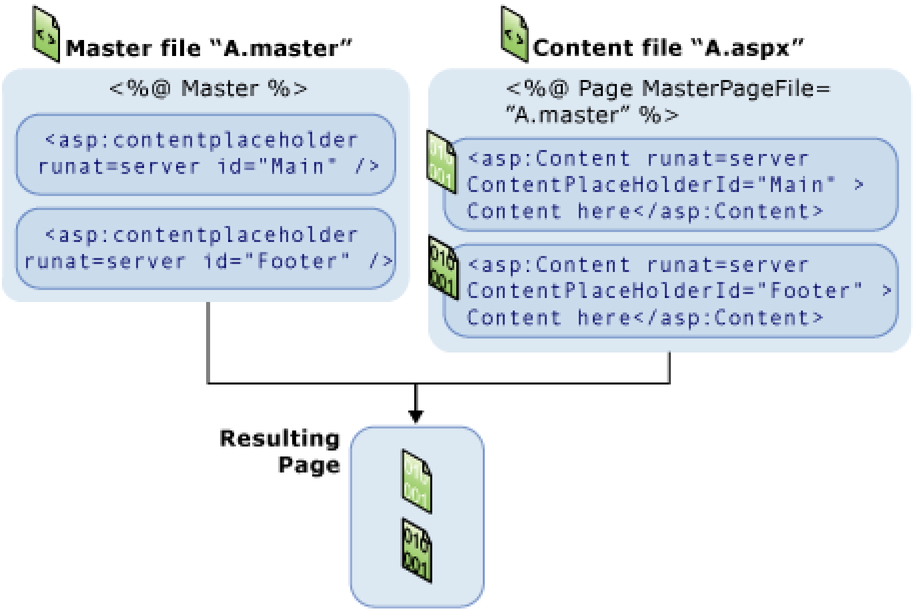
You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page. The binding is established in the content page's **@ Page** directive by including a MasterPageFile attribute that points to the master page to be used. For example, a content page might have the following **@ Page** directive, which binds it to the `Master1.master` page.

C#

```
<%@ Page Language="C#" MasterPageFile="~/MasterPages/Master1.master" Title="Content Page"%>
```

In the content page, you create the content by adding Content controls and mapping them to ContentPlaceHolder controls on the master page. For example, the master page might have content placeholders called `Main` and `Footer`. In the content page, you can create two Content controls, one that is mapped to the ContentPlaceHolder control `Main` and the other mapped to the ContentPlaceHolder control `Footer`, as shown in the following figure.

Replacing placeholder content


Master file "A.master"
<%@ Master %>
<asp:contentplaceholder runat=server id="Main" />
<asp:contentplaceholder runat=server id="Footer" />
Content file "A.aspx"
<%@ Page MasterPageFile= "A.master" %>
<asp:Content runat=server ContentPlaceHolderId="Main" > Content here</asp:Content>
<asp:Content runat=server ContentPlaceHolderId="Footer" > Content here</asp:Content>
Resulting Page

After creating Content controls, you add text and controls to them. In a content page, anything that is not inside the Content controls (except script blocks for server code) results in an error. You can perform any tasks in a content page that you do in an ASP.NET page. For example, you can generate content for a Content control using server controls and database queries or other dynamic mechanisms.

A content page might look like the following.

VB

```
<% @ Page Language="VB" MasterPageFile="~/Master.master" Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
    Main content.
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="Footer" Runat="Server" >
    Footer content.
</asp:content>
```

[C#]

```
<% @ Page Language="C#" MasterPageFile="~/Master.master" Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
    Main content.
</asp:Content>
```

```
<asp:Content ID="Content2" ContentPlaceHolderID="Footer" Runat="Server" >
    Footer content.
</asp:content>
```

The **@ Page** directive binds the content page to a specific master page, and it defines a title for the page that will be merged into the master page. Note that the content page contains no other markup outside of the Content controls. (The master page must contain a **head** element with the attribute `runat="server"` so that the title setting can be merged at run time.)

You can create multiple master pages to define different layouts for different parts of your site, and a different set of content pages for each master page.

Back to top

Advantages of Master Pages

Master pages provide functionality that developers have traditionally created by copying existing code, text, and control elements repeatedly; using framesets; using include files for common elements; using ASP.NET user controls; and so on. Advantages of master pages include the following:

- They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
- They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.
- They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
- They provide an object model that allows you to customize the master page from individual content pages.

Back to top

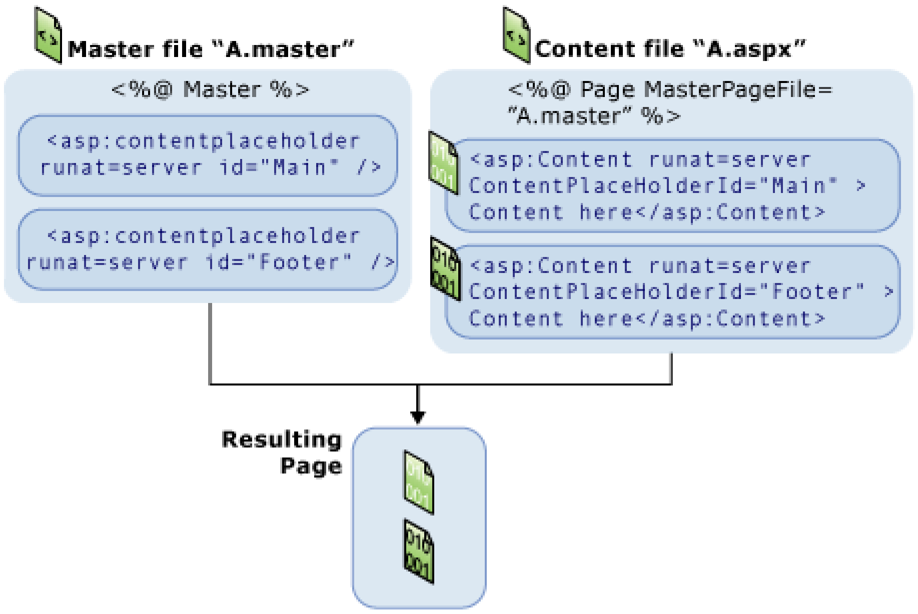
Run-time Behavior of Master Pages

At run time, master pages are handled in the following sequence:

1. Users request a page by typing the URL of the content page.
2. When the page is fetched, the **@ Page** directive is read. If the directive references a master page, the master page is read as well. If this is the first time the pages have been requested, both pages are compiled.
3. The master page with the updated content is merged into the control tree of the content page.
4. The content of individual Content controls is merged into the corresponding ContentPlaceHolder control in the master page.
5. The resulting merged page is rendered to the browser.

The process is illustrated in the following diagram.

Master pages at run time


Master file "A.master"
<%@ Master %>
<asp:contentplaceholder runat=server id="Main" />
<asp:contentplaceholder runat=server id="Footer" />
Content file "A.aspx"
<%@ Page MasterPageFile= "A.master" %>
<asp:Content runat=server ContentPlaceHolderId="Main" > Content here</asp:Content>
<asp:Content runat=server ContentPlaceHolderId="Footer" > Content here</asp:Content>
Resulting Page

From the user's perspective, the combined master and content pages are a single, discrete page. The URL of the page is that of the content page.

From a programming perspective, the two pages act as separate containers for their respective controls. The content page acts as a container for the master page. However, you can reference public master-page members from code in the content page, as described in the next section.

Note that the master page becomes a part of the content page. In effect, the master page acts in much the same way a user control acts — as a child of the content page and as a container within that page. In this case, however, the master page is the container for all of the server controls that are rendered to the browser. The control tree for a merged master and content page looks something like this:

```
Page
    Master Page
        (Master page markup and controls)
        ContentPlaceHolder
            Content page markup and server controls
        (Master page markup and controls)
        ContentPlaceHolder
            Content page markup and server controls
        (Master page markup and controls)
```

This diagram is simplified; if the content page does not have corresponding Content controls, the master page might also have markup and controls in the ContentPlaceHolder controls.

In general, this structure has no effect on how you construct your pages or program them. However, in some cases, if you set a page-wide property on the master page, it can affect the behavior of the content page, because the master page is the closest parent for the controls on the page. For example, if you set the EnableViewState property on the content page to **true** but set the same property to **false** in the master page, view state will effectively be disabled because the setting on the master page will take priority.

Back to top

Master Page and Content Page Paths

When a content page is requested, its content is merged with the master page, and the page runs in the context of the content page. For example, if you get the CurrentExecutionFilePath property of the HttpRequest object, whether in content page code or in master page code, the path represents the location of the content page.

The master page and content page do not have to be in the same folder. As long as the MasterPageFile attribute in the content page's **@ Page** directive resolves to a .master page, ASP.NET can merge the content and master pages into a single rendered page.

Referencing External Resources

Both the content page and master page can contain controls and elements that reference external resources. For example, both might contain image controls that reference image files, or they might contain anchors that reference other pages.

The context for the merged content and master pages is that of the content page. This can affect how you specify URLs for resources, such as image files and target pages, in anchors.

Server Controls

In server controls on master pages, ASP.NET dynamically modifies the URLs of properties that reference external resources. For example, you might put an Image control on a master page and set its ImageUrl property to be relative to the master page. At run time, ASP.NET will modify the URL so that it resolves correctly in the context of the content page.

ASP.NET can modify URLs in the following cases:

- The URL is a property of an ASP.NET server control.
- The property is marked internally in the control as being a URL. (The property is marked with the attribute UrlPropertyAttribute.) In practical terms, ASP.NET server control properties that are commonly used to reference external resources are marked in this way.

Other Elements

ASP.NET cannot modify URLs on elements that are not server controls. For example, if you use an **img** element on a master page and set its **src** attribute to a URL, ASP.NET will not modify the URL. In that case, the URL will be resolved in the context of the content page and create the URL accordingly.

In general, when working with elements on master pages, it is recommended that you use a server control, even for elements that do not require server code. For example, instead of using an **img** element, use an Image server control. That way, ASP.NET can resolve URLs correctly and you can avoid maintenance issues that might arise if you move the master or content page.

For more information about specifying paths for ASP.NET server controls, see ASP.NET Web Project Paths.

Back to top

Master Pages and Themes

You cannot directly apply an ASP.NET theme to a master page. If you add a theme attribute to the **@ Master** directive, the page will raise an error when it runs.

However, themes are applied to master pages under these circumstances:

- If a theme is defined in the content page. Master pages are resolved in the context of content pages, so the content page's theme is applied to the master page as well.
- If the site as a whole is configured to use a theme by including a theme definition in the pages Element (ASP.NET Settings Schema) element.

For more information, see ASP.NET Themes and Skins.

Back to top

Scoping Master Pages

You can attach content pages to a master page at three levels:

- **At the page level** You can use a page directive in each content page to bind it to a master page, as in the following code example.

  C#

  ```
  <%@ Page Language="C#" MasterPageFile="MySite.Master" %>
  ```

- **At the application level** By making a setting in the **pages** element of the application's configuration file (Web.config), you can specify that all ASP.NET pages (.aspx files) in the application automatically bind to a master page. The element might look like the following.

  ```
  <pages masterPageFile="MySite.Master" />
  ```

  If you use this strategy, all ASP.NET pages in the application that have Content controls are merged with the specified master page. (If an ASP.NET page does not contain Content controls, the master page is not applied.)

- **At the folder level** This strategy is like binding at the application level, except that you make the setting in a Web.config file in one folder only. The master-page bindings then apply to the ASP.NET pages in that folder.

Back to top

Related Topics

Title	Definition
Sharing Master Pages in Visual Studio	Describes how you can use the same master pages in multiple Web sites or projects.
Nested ASP.NET Master Pages	Describes how to include one master page inside another.
Working with ASP.NET Master Pages Programmatically	Provides information about how to access master-page elements in code.
Editing Master Pages in the Visual Web Developer Designer	Describes how to use the Visual Studio Web page designer to create and edit master pages and content pages.
Walkthrough: Creating and Using ASP.NET Master Pages in Visual Web Developer	Provides a step-by-step tutorial on how to create a master page and content pages.
Walkthrough: Using Nested Master Pages in ASP.NET	Shows you how to nest master pages so that the parent master page can provide a consistent layout throughout a Web site, and the child master page can be used as a template for consistent layout within the parent master page.
Events in ASP.NET Master and Content Pages	Describes how page and control events are raised in merged master and content pages.
ASP.NET Web Forms Pages	Provides links to topics on creating ASP.NET Web pages.

Back to top

Reference

System.Web.UI.MasterPage

Back to top