ASP.NET Development

Architectural Analysis

This document explores ASP.NET development using Visual Studio 2013.

2015

Heary, James

3/6/2015

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/21/2014 | 1.0 | Created | Jim Heary |
| 03/06/2015 | 2.0 | Updated | Jim Heary |
|  |  |  |  |

Table of Contents

Introduction 3

Overview 3

Creating ASP.NET Web Projects 3

Web application projects versus web site projects 3

Compilation 4

Deployment 5

IIS/ASP.NET Architecture 6

Overview 6

Protocol Listeners 6

Design View 6

Structure 6

Behavior 9

Process View 10

Deployment View 12

Security 12

Performance 12

Scalability 12

Testing 12

WebForms 13

Overview 13

MVC 14

Overview 14

WebAPI 15

Overview 15

SPA 16

Overview 16

Appendix 17

ASP.NET Routing 17

Overview 17

Defining URL Routes 17

How URLs Are Matched To Routes 19

Creating URLs From Routes 19

CSS 20

Overview 20

Web Design Approaches 20

Rendering Engines 21

ViewPort 21

Media Queries 21

CSS Reference 21

# Introduction

## Overview

ASP.NET is an open source server-side Web application framework designed for Web development to produce dynamic Web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language. The ASP.NET SOAP extension framework allows ASP.NET components to process SOAP messages. ASP.NET is in the process of being re-implemented as a modern and modular web framework, together with other frameworks like Entity Framework. The new framework will make use of the new open-source .NET Compiler Platform (Code-name "Roslyn") and be cross platform. ASP.NET Web Pages and ASP.NET MVC will merge into a unified MVC 6. The project is called ASP.NET vNext.

#### Web Sites

ASP.NET enables building standards-based websites with HTML5, CSS3, and JavaScript. ASP.NET supports three approaches for making web sites. ASP.NET Web Forms uses controls and an event-model for component-based development. ASP.NET MVC values separation of concerns and enables easier test-driven development. ASP.NET Web Pages prefers a single page model that mixes code and HTML markup. You can mix and match these techniques within one application depending on your needs.

#### API’s

ASP.NET Web API creates rich REST-ful Web Services that return JSON, XML, or any kind of content the web supports. ASP.NET Web APIs can provide data services to mobile apps like Windows Phones, iPhones, Android and more. ASP.NET Web APIs can be used in any ASP.NET Web Application, including ASP.NET MVC, Web Forms, or Web Pages.

#### Real Time

ASP.NET SignalR is a new library for ASP.NET that enables real-time bi-directional communication between client and server. ASP.NET SignalR enables new standards like Web Sockets while gracefully falling back to support older browsers. ASP.NET SignalR can support JavaScript clients, as well as clients on Android, iPhone, and all C# clients like Windows Phone and Windows 8.

#### Mobile

ASP.NET can power mobile applications with responsive design frameworks like Twitter Bootstrap included out of the box with VS2013. You can use any CSS framework or open grid system you prefer, or perhaps select a powerful mobile system like jQuery Mobile or Sencha and great mobile applications with PhoneGap.

## Creating ASP.NET Web Projects

### Web application projects versus web site projects

ASP.NET gives you a choice between two kinds of web projects: web application projects and web site projects.

#### Web Application Project Scenarios

* You want to be able to use the [Edit and Continue](http://msdn.microsoft.com/en-us/library/bcew296c(v=vs.110).aspx) feature of the Visual Studio debugger.
* You want to run unit tests on code that is in the class files that are associated with ASP.NET pages.
* You want to refer to the classes that are associated with pages and user controls from standalone classes.
* You want to establish project dependencies between multiple web projects.
* You want the compiler to create a single assembly for the entire site.
* You want control over the assembly name and version number that is generated for the site.
* You want to use MSBuild or Team Build to compile the project.
* You want to avoid putting source code on a production server.

#### Web Site Project Scenarios

* You want to open the production site in Visual Studio and update it in real time by using FTP.
* You do not want to have to explicitly compile the project in order to deploy it.
* If you do precompile the site, you want the compiler to create multiple assemblies for the site, which can include one assembly per page or user control, or one or more assemblies per folder.
* You want to be able to update individual files in production by just copying new versions to the production server, or by editing the files directly on the production server.
* If you precompile the site, you want to be able to update individual ASP.NET web pages without having to recompile the entire site.
* You like to keep your source code on the production server because it can serve as an additional backup copy.

### Compilation

For web application projects, you typically build the project in Visual Studio or by using the ASP.NET batch compiler on a computer that is not the production IIS server. All code-behind class files and standalone class files in the project are compiled into a single assembly, which is then put in the web application project's Bin folder. (The .aspx and .ascx files are compiled dynamically in a manner similar to what is done for web site projects.)

For web site projects, you do not have to manually compile the project. Web site projects are typically compiled dynamically by ASP.NET (on both the development computer and the production IIS server). You can choose between batch compilation mode, which typically produces one assembly per folder, and fixed compilation mode, which typically produces one assembly for each page or user control.

#### Advantages of the compilation model for web application projects

* You can use MSBuild to create a custom batch-compilation process.
* It is easy to specify assembly attributes such as name and version.
* Compiling in advance ensures users do not have to wait while the site compiles on the production server.
* You have complete control over where you put code files in the project folder structure, and how classes in the project refer to each other. (Dynamic compilation requires that the source code for any classes that are used throughout the site must be in the App\_Code folder. You cannot refer to a page or user control class from a class in App\_Code.)

#### Advantages of the compilation model for Web site projects

* You can test specific pages regardless of the state of other pages. This is because running an individual page does not require that the whole site compile successfully, only the page and any components it depends on, such as code in the App\_Code folder or the Global.asax file.
* It is easy to update a Web site in production. You can update individual source code files on the production server without having to explicitly recompile the site. You can update individual files that are ready for deployment even if other files are not ready due to compile errors. You can also open the Web site on the production IIS server directly in Visual Studio and update the Web site in real time.
* Precompiling to multiple assemblies can have a performance advantage in some scenarios. A typical example is a site that has many pages with lots of code written for them. Most of the pages are rarely requested and only some are used frequently. If you compile a site like this into multiple assemblies, the production server can load only the assemblies that are required for the current requests. If a page is not requested, its corresponding assembly is not loaded.

### Deployment

You typically copy all of your source code to the web server when you deploy a web site project. In a web application project, source code is compiled into an assembly (a .dll file), and that is what has to be on the web server.

An advantage of web application projects is that you can avoid deploying source code to the IIS server. In some scenarios, such as shared hosting environments, you might be concerned about unauthorized access to source code on the IIS server. For a web site project, you can avoid this risk by precompiling on a development computer and deploying the generated assemblies instead of the source code. However, in that case you lose some of the benefits of easy site updates.

An advantage of web site projects is that when you make a small change to a web site, you do not have to redeploy the whole site. Instead, you can copy just the changed file or files to the production IIS server. You can also edit files directly on the production server. Because a web application project's code files are compiled into a single assembly file, you must deploy the whole site even for small changes, unless the only change is to an .aspx or .ascx file.

Deployment often involves other tasks in addition to copying assemblies or code to a server. For example, database scripts might have to run in production, and connection strings in the Web.config file might have to be changed for a production server. Visual Studio provides tools such as one-click publish that work with web application projects to automate many of these tasks. These tools are available for both web application projects and web site projects.

# IIS/ASP.NET Architecture

## Overview

Internet Information Services (IIS) 7 and above provides a request-processing architecture that includes:

* The Windows Process Activation Service (WAS), which enables sites to use protocols other than HTTP and HTTPS.
* A Web server engine that can be customized by adding or removing modules.
* Integrated request-processing pipelines from IIS and ASP.NET.

### Protocol Listeners

Protocol listeners receive protocol-specific requests, send them to IIS for processing, and then return responses to requestors. By default, IIS provides HTTP.sys as the protocol listener that listens for HTTP and HTTPS requests. To support services and applications that use protocols other than HTTP and HTTPS, you can use technologies such as Windows Communication Foundation (WCF). WCF has listener adapters that provide the functionality of both a protocol listener and a listener adapter.

## Design View

### Structure

The following drawing shows the structure of IIS in terms of modules, components, and interfaces.



#### Http.sys

The HTTP listener is part of the networking subsystem of Windows operating systems, and it is implemented as a kernel-mode device driver called the HTTP protocol stack. HTTP.sys listens for HTTP requests from the network, passes the requests onto IIS for processing, and then returns processed responses to client browsers. HTTP.sys provides Kernel-mode response caching, Kernel-mode request queuing, and request pre-processing and security filtering.

#### W3SVC

In IIS 7 and above, functionality that was previously handled by the World Wide Web Publishing Service (WWW Service) alone is now split between two services: WWW Service and a new service, Windows Process Activation Service (WAS). These two services run as LocalSystem in the same Svchost.exe process, and share the same binaries. In IIS, the WWW service no longer manages worker processes. Instead, the WWW Service is the listener adapter for the HTTP listener, HTTP.sys. As the listener adapter, the WWW Service is primarily responsible for configuring HTTP.sys, updating HTTP.sys when configuration changes, and notifying WAS when a request enters the request queue. Additionally, the WWW Service continues to collect the counters for Web sites. Because performance counters remain part of the WWW Service, they are HTTP specific and do not apply to WAS.

#### WAS

In IIS 7 and above, Windows Process Activation Service (WAS) manages application pool configuration and worker processes instead of the WWW Service. This enables you to use the same configuration and process model for HTTP and non-HTTP sites.

*Configuration Management*

On startup, WAS reads certain information from the ApplicationHost.config file, and passes that information to listener adapters on the server. Listener adapters are components that establish communication between WAS and protocol listeners, such as HTTP.sys. Once listener adapters receive configuration information, they configure their related protocol listeners and prepare the listeners to listen for requests. In the case of WCF, a listener adapter includes the functionality of a protocol listener. So, a WCF listener adapter, such as NetTcpActivator, is configured based on information from WAS. Once NetTcpActivator is configured, it listens for requests that use the net.tcp protocol. WAS reads global configuration information, protocol configuration information for both HTTP and non-HTTP protocols, application pool configuration (i.e. process account information), site configuration (i.e. bindings and applications), and application configuration (i.e. enabled protocols and application pools). If ApplicationHost.config changes, WAS receives a notification and updates the listener adapters with the new information.

*Process Management*

WAS manages application pools and worker processes for both HTTP and non-HTTP requests. When a protocol listener picks up a client request, WAS determines if a worker process is running or not. If an application pool already has a worker process that is servicing requests, the listener adapter passes the request onto the worker process for processing. If there is no worker process in the application pool, WAS will start a worker process so that the listener adapter can pass the request to it for processing. Because WAS manages processes for both HTTP and non-HTTP protocols, you can run applications with different protocols in the same application pool.

#### W3WP

In IIS 7.0, ASP.NET run time is integrated with the core web server. The IIS and ASP.NET request pipelines are combined, providing a unified (i.e. integrated) request processing pipeline that is exposed to both native and managed modules. The IIS 7.0 request processing pipeline is implemented by the core web server engine. It enables multiple independent modules to provide services for the same request. All of the web server features are implemented as stand-alone modules. There are over 40 separate native and managed modules. Each module implements a particular Web server feature or functionality, such as logging or output caching.

Native modules are implemented as dynamic-link libraries (DLLs) based on public IIS 7.0 C++ extensibility APIs. Managed modules are implemented as managed .NET Framework classes based on the ASP.NET integration model in IIS 7.0. (IIS 7.0 has integrated the existing IHttpModule API for ASP.NET.) Both of these APIs enable modules to participate in the IIS 7.0 request processing pipeline and access all events for all requests. A pipeline is an ordered list consisting of native and managed modules that perform specific tasks in response to requests. When a worker process in an application pool receives a request from HTTP.sys, the request passes through an ordered list of stages. As a result of processing, the response is generated and sent back to HTTP.sys.

Each stage in the pipeline raises an event. Native and managed modules subscribe to events in the stages of the pipeline that are relevant to them. When the event is raised, the native and managed modules that subscribe to that event are notified and do their work to process the request. The pipeline event model enables multiple modules to execute during request processing.

Most of the pipeline events are intended for a specific type of task, such as authentication, authorization, caching, and logging. The following list describes stages and corresponding events in the request processing pipeline:

* Begin Request stage. This stage starts request processing. The *BeginRequest* event is raised.
* Authenticate Request stage. This stage authenticates the requesting user. The *AuthenticateRequest* event is raised.
* Authorize Request stage. At this stage, the *AuthorizeRequest* event is raised. This stage checks access to the requested resource for the authenticated user. If access is denied, the request is rejected.
* Resolve Cache stage. At this stage, *ResolveRequestCache* event is raised. This stage checks to see if the response to the request can be retrieved from a cache.
* Map Handler stage. At this stage, the *MapRequestHandler* event is raised. This stage determines the handler for the request.
* Acquire State stage. At this stage, the *AcquireRequestState* event is raised. This stage retrieves the required state for the request.
* Pre-execute Handler stage. At this stage, the *PreExecuteRequestHandler* event is raised. This stage signals that the handler is about to be executed and performs the preprocessing tasks if needed.
* Execute Handler stage. At this stage, the *ExecuteRequestHandler* event is raised. The handler executes and generates the response.
* Release State stage. At this stage, the *ReleaseRequestState* event is raised. This stage releases the request state.
* Update Cache stage. This stage updates the cache. The *UpdateRequestCache* event is raised.
* Log Request stage. At this stage, the request is logged. The *LogRequest* event is raised.
* End Request stage. At this stage, the *EndRequest* event is raised, which signals that the request processing is about to complete.

Though native and managed modules implement the same logical module concept, they use two different APIs. To enable an integrated pipeline model for both native and managed modules, IIS 7.0 provides a special native module called Managed Engine. The Managed Engine module in effect provides an integration wrapper for ASP.NET modules that enables these managed modules to act as if they were native IIS modules and handlers. It acts as a proxy for event notifications and propagates a required request state to the managed modules. Together with the ASP.NET engine, it sets up the integrated pipeline and is also responsible for reading the managed modules and handlers configuration. When a request requires a managed module, the Managed Engine module creates an AppDomain where that managed module can perform the necessary processing, such as authenticating a user with Forms authentication.

*Module Scope*

Modules can be installed and enabled on different levels. Modules that are enabled on the server level provide a default feature set for all applications on the server. The IIS global configuration store, applicationHost.config, provides the unified list of both native and managed modules. Each time WAS activates a worker process, it gets the configuration from the configuration store, and the worker process loads all globally listed modules.

Native modules can be installed only at the server level. They cannot be installed at the application level. At the application level, the global native modules that are enabled at the server level can be removed, or those that are installed but not enabled globally can be enabled for that application. Managed modules can be added at the server, site, and application levels. Application-specific modules are loaded upon the first request to the application. You can manage both native and managed modules using the Modules feature in IIS Manager.

*Module Ordering*

The pipeline model ensures that the typical Web server processing tasks are performed in the correct order. The server uses the sequence of modules list in the <modules> configuration section to order module execution within each request processing stage. By executing during the relevant stage, the majority of modules automatically avoid ordering problems. However, multiple modules that execute within the same stage may have ordering dependencies. For example, the built-in authentication modules that run at the Authenticate Request stage should be executed in the strongest to weakest order so that the request is authenticated with the strongest credentials available. To manage ordering dependencies, the administrator can control the ordering of modules by changing the order in which they are listed in the <modules> section.

### Behavior



## Process View

The process view shown below describes the running processes for IIS request processing and the identities (i.e. IIdentity) they run in.



ApplicationPoolIdentity- the Windows operating system provides a feature called "virtual accounts" that allows IIS to create a unique identity for each of its application pools.

* LocalSystem- the LocalSystem account is a predefined local account used by the service control manager. It has extensive privileges on the local computer, and acts as the computer on the network. Its token includes the NT AUTHORITY\SYSTEM and BUILTIN\Administrators SIDs; these accounts have access to most system objects. The name of the account in all locales is .\LocalSystem. The name LocalSystem or ComputerName\LocalSystem can also be used. This account does not have a password.
* LocalService- the LocalService account is a predefined local account used by the service control manager. It has minimum privileges on the local computer and presents anonymous credentials on the network. Its token includes the NT AUTHORITY\LOCALSERVICE SID. This account does not have a password.
* NetworkService- the NetworkService account is a predefined local account used by the service control manager. It has minimum privileges on the local computer and acts as the local computer on the network. Its token includes the NT AUTHORITY\NETWORKSERVICE SID. This account does not have a password. A service that runs in the context of the NetworkService account presents the computer's credentials to remote servers.

## Deployment View



## Security

## Performance

## Scalability

## Testing

# WebForms

## Overview

With ASP.NET Web Forms, you can build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

* WYSIWYG designer in Visual Studio.
* Server controls that render HTML and that you can customize by setting properties and styles.
* A rich assortment of controls for data access and data display.
* An event model that exposes events which you can program like you would program a client application such as WPF.
* Automatic preservation of state (data) between HTTP requests.

In general, creating a Web Forms application requires less programming effort than creating the same application by using the ASP.NET MVC framework. However, Web Forms is not just for rapid application development. There are many complex commercial applications and frameworks built on top of Web Forms.

Because a Web Forms page and the controls on the page automatically generate much of the markup that's sent to the browser, you don't have the kind of fine-grained control over the HTML that ASP.NET MVC offers. The declarative model for configuring pages and controls minimizes the amount of code you have to write but hides some of the behavior of HTML and HTTP. For example, it's not always possible to specify exactly what markup might be generated by a control.

The Web Forms framework doesn't lend itself as readily as ASP.NET MVC to patterns-based development practices such as test-driven development, separation of concerns, inversion of control, and dependency injection. If you want to write code factored that way, you can; it’s just not as automatic as it is in the ASP.NET MVC framework. The ASP.NET Web Forms MVP project shows an approach that facilitates separation of concerns and testability while maintaining the rapid development that Web Forms was built to deliver. Microsoft SharePoint is built on Web Forms MVP.

The Web Forms template creates a sample Web Forms application that uses Bootstrap to provide responsive design and theming features. The following illustration shows the home page.

# MVC

## Overview

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. ASP.NET MVC includes many features that enable fast, TDD-friendly development for creating sophisticated applications that use the latest web standards.

ASP.NET MVC was designed to facilitate patterns-based development practices such as test-driven development, separation of concerns, inversion of control, and dependency injection. The framework encourages separating the business logic layer of a web application from its presentation layer. By dividing the application into models (M), views (V), and controllers (C), ASP.NET MVC can make it easier to manage complexity in larger applications.

With ASP.NET MVC, you work more directly with HTML and HTTP than in Web Forms. For example, Web Forms can automatically preserve state between HTTP requests, but you have to code that explicitly in MVC. The advantage of the MVC model is that it enables you to take complete control over exactly what your application is doing and how it behaves in the web environment. The disadvantage is that you have to write more code.

MVC was designed to be extensible, providing power developers the ability to customize the framework for their application needs. In addition, the ASP.NET MVC source code is available under an OSI license.

The MVC template creates a sample MVC 5 application that uses Bootstrap to provide responsive design and theming features.

# WebAPI

## Overview

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful services on the .NET Framework.

The Web API template creates a sample web service based on Web API, including API help pages based on MVC.

# SPA

## Overview

A single-page application (SPA), is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application. In a SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does control transfer to another page, although modern web technologies (such as those included in HTML5) can provide the perception and navigability of separate logical pages in the application. Interaction with the single page application often involves dynamic communication with the web server behind the scenes.

There are a few distinct characteristics of the professional Single Page Application:

* Chunking – the web page is constructed by loading chunks of HTML fragments and JSON data instead of receiving full HTML from a web server on every request.
* Controllers – JavaScript code that handles complex DOM and data manipulations, application logic and AJAX calls is replaced by controllers that separate views and models using MVC or MVVM patterns.
* Templating – coding of UI and DOM manipulations are replaced by declarative binding of data to HTML templates.
* Routing – selection of views and navigation (without page reloads) that preserves page state, elements and data
* Real-time communication – two-way communication of a client application and web server replaces one-way requests from a browser (i.e. HTML5 Web Sockets, SignalR)

Local storage – capabilities of storing data on a browser for performance and offline access replace cookies and intensive data loads from web server (i.e. HTML5 Local storage).

The Single Page Application (SPA) template creates a sample application that uses JavaScript, HTML 5, and KnockoutJS on the client, and ASP.NET Web API on the server.

The only authentication option for the SPA template is Individual User Accounts.

# Appendix

## ASP.NET Routing

### Overview

ASP.NET routing enables you to use URLs that do not have to map to specific files in a Web site. Because the URL does not have to map to a file, you can use URLs in a Web application that are descriptive of the user's action and therefore more easily understood by users. In ASP.NET routing, you define URL patterns that contain placeholders for values that are used when you handle URL requests. At run time, the pieces of the URL that follow the application name are parsed into discrete values, based on a URL pattern that you have defined. For example, in the request for http://server/application/Products/show/beverages, the routing parser can pass the values Products, show, and beverages to a handler for the request. You can also use the URL patterns to programmatically create URLs that correspond to the routes. This enables you to centralize the logic for creating hyperlinks in your ASP.NET application.

### Defining URL Routes

The URL patterns that you define are known as routes. In a route, you specify placeholders that are mapped to values that are parsed from the URL request. You can also specify constant values that are used for matching URL requests. In a route, you define placeholders (referred to as URL parameters) by enclosing them in braces ( { and } ). The / character is interpreted as a delimiter when the URL is parsed. Information in the route definition that is not a delimiter and that is not in braces is treated as a constant value. Values that are extracted from between the delimiters are assigned to placeholders. You can define more than one placeholder between delimiters, but they must be separated by a constant value. For example, {language}-{country}/{action} is a valid route pattern. However, {language}{country}/{action} is not a valid pattern, because there is no constant or delimiter between the placeholders.

Typically, you add routes in a method that is called from the handler for the Application\_Start event in the Global.asax file. This approach makes sure that the routes are available when the application starts. It also enables you to call the method directly when you unit-test the application. If you want to call a method directly when you are unit-testing the application, the method that registers the routes must be static and must have a RouteCollection parameter. You add routes by adding them to the static Routes property of the RouteTable class. The Routes property is a RouteCollection object that stores all the routes for the ASP.NET application.

protected void Application\_Start(object sender, EventArgs e) {

RegisterRoutes(RouteTable.Routes);

}

public static void RegisterRoutes(RouteCollection routes) {

routes.Add(new Route( "Category/{action}/{categoryName}", new CategoryRouteHandler()));

}

#### Setting Default Values for Route Parameters

When you define a route, you can assign a default value for a parameter. The default value is used if a value for that parameter is not included in the URL. You set default values for a route by assigning a dictionary to the Defaults property of the Route class. The following example shows a route with default values.

public static void RegisterRoutes(RouteCollection routes) {

routes.Add(new Route("Category/{action}/{categoryName}",new CategoryRouteHandler())

{Defaults = new RouteValueDictionary({"categoryName", "food"}, {"action", "show"})})

}

#### Handling a Variable Number of Segments

Sometimes you have to handle URL requests that contain a variable number of URL segments. When you define a route, you can specify that the last parameter should match the rest of the URL by marking the parameter with an asterisk (\*). This is then referred to as a catch-all parameter. A route with a catch-all parameter will also match URLs that do not contain any values for the last parameter. The following example shows a route pattern that matches an unknown number of segments.

query/{queryname}/{\*queryvalues}

When ASP.NET routing handles a URL request, the route definition shown results as follows:

/query/select/bikes/onsale queryname=”select”, queryvalues=”bikes/onsale”

/query/select queryname=”select”, queryvalues=Empty String

#### Adding Constraints to Routes

In addition to matching a URL request to a route definition by the number of parameters in the URL, you can specify that values in the parameters meet certain constraints. If a URL contains values that are outside the constraints for a route, that route is not used to handle the request. You add constraints to make sure that the URL parameters contain values that will work in your application.

Constraints are defined by using regular expressions or by using objects that implement the IRouteConstraint interface. When you add the route definition to the Routes collection, you add constraints by creating a RouteValueDictionary object that contains the verification test. You then assign this object to the Constraints property. The key in the dictionary identifies the parameter that the constraint applies to. The value in the dictionary can be either a string that represents a regular expression or an object that implements the IRouteConstraint interface. If you provide a string, routing treats the string as a regular expression and checks whether the parameter value is valid by calling the IsMatch method of the Regex class. The regular expression is always treated as case-insensitive. If you provide an IRouteConstraint object, ASP.NET routing checks whether the parameter value is valid by calling the Match method of the IRouteConstraint object. The Match method returns a Boolean value that indicates whether the parameter value is valid.

public static void RegisterRoutes(RouteCollection routes) {

routes.Add(new Route("{locale}/{year}", new ReportRouteHandler())

{Constraints = new RouteValueDictionary({"locale", "[a-z]{2}-[a-z]{2}"},{year, @"\d{4}")}});

}

#### Scenarios When Routing Is Not Applied

By default, routing does not handle requests that map to an existing physical file on the Web server. For example, a request for http://server/application/Products/Beverages/Coffee.aspx is not handled by routing if a physical file exists at Products/Beverages/Coffee.aspx. Routing does not handle the request even if it matches a defined pattern, such as {controller}/{action}/{id}.

If you want routing to handle all requests, even requests that point to files, you can overwrite the default behavior by setting the RouteExistingFiles property of the RouteCollection object to true. When you set this value to true, all requests that match a defined pattern are handled by routing.

You can also specify that routing should not handle certain URL requests. You prevent routing from handling certain requests by defining a route and specifying that the StopRoutingHandler class should be used to handle that pattern. When a request is handled by a StopRoutingHandler object, the StopRoutingHandler object blocks any additional processing of the request as a route. Instead, the request is processed as an ASP.NET page, Web service, or other ASP.NET endpoint. For example, you can add the following route definition to prevent routing from handling requests for the WebResource.axd file.

public static void RegisterRoutes(RouteCollection routes) {

routes.Add(new Route("{resource}.axd/{\*pathInfo}", new StopRoutingHandler()));

}

### How URLs Are Matched To Routes

When routing handles URL requests, it tries to match the URL of the request to a route. Matching a URL request to a route depends on all the following conditions:

* The defined route patterns or the default route patterns, if any, that are included in the project type.
* The order in which routes are added to the Routes collection.
* Any default values that have been provided for a route.
* Any constraints that have been provided for a route.
* Whether routing definitions exist to handle requests that match a physical file.

To avoid having the wrong handler handle a request, you must consider all these conditions when you define routes. The order in which Route objects appear in the Routes collection is significant. Route matching is tried from the first route to the last route in the collection. When a match occurs, no more routes are evaluated. In general, add routes to the Routes property in order from the most specific route definitions to least specific ones. If a URL does not match any Route object defined in the RouteTable collection, ASP.NET routing does not process the request. Instead, processing is passed to an ASP.NET page, Web service, or other ASP.NET endpoint.

### Creating URLs From Routes

You can use routes to generate URLs when you want to centralize the logic for constructing URLs. You create a URL by passing parameter values as a dictionary to the GetVirtualPath method of the RouteCollection object. The GetVirtualPath method looks for the first route in the RouteCollection object that matches the parameters in the dictionary. The matching route is used to generate the URL. The following example shows a route definition. In the example, CategoryRouteHandler is a custom route handler.

public static void RegisterRoutes(RouteCollection routes) {

routes.Add(new Route("Category/{action}/{categoryName}", new CategoryRouteHandler())

{Defaults = new RouteValueDictionary {{"categoryName", "food"}, {"action", "show"}}});

}

HyperLink1.NavigateUrl = RouteTable.Routes.GetVirtualPath(context,

new RouteValueDictionary({"categoryName","beverages"}, {"action","summarize"})).VirtualPath;

## CSS

### Overview

Cascading Style Sheets (CSS) is a language used for describing the look and formatting of a document written in a markup language. While most often used to change the style of web pages written in HTML and XHTML, the language can be applied to any kind of XML document including plain XML, SVG and XUL. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages and user interfaces for web and mobile applications.

CSS is designed to enable the separation of document content from document presentation, including elements such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content. CSS makes it possible to separate presentation instructions from the HTML content in a separate file or style section of the HTML file. For each matching HTML element, it provides a list of formatting instructions.

This separation of formatting and content makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice, and on Braille-based, tactile devices. It can also be used to display the web page differently depending on screen size or device type. Another advantage of CSS is that aesthetic changes to the graphic design of a document(s) can be changed quickly and easily by editing a few lines in one file, rather than by changing markup in every affected document.

The CSS specification describes a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called *cascade*, priorities or *weights* are calculated and assigned to rules, so that the results are predictable.

### Web Design Approaches

#### Fixed

A web design approach that uses table layouts with fixed (i.e. pixels/points) units. The site is designed to accommodate the smallest anticipated browser viewport; it then displays the same in larger viewports.

#### Flexible/Fluid

A web design approach that uses table layouts with relative (i.e. percentages) units. The site is designed to accommodate a range of viewport dimensions.

#### Adaptive/Progressive Enhancement

#### Responsive

A web design approach aimed at creating sites that provide an optimal viewing experience (i.e. easy reading and navigation with minimal resizing/panning/scrolling) across a wide range of devices from desktop to mobile. A responsive site adapts the layout to the viewing environment by using fluid, proportion-based grids, flexible images, and CSS3 media queries (i.e. @media) in the following ways:

* A fluid grid uses page element sizing in relative units (i.e. percentages) rather than absolute units (i.e. pixels or points)
* Flexible images are also sized in relative units to prevent displaying outside their containing element
* Media queries allow the page to use different CSS style rules based on characteristics of the display device, most commonly the width of the browser

### Rendering Engines

Every browser is backed by a rendering engine to draw the HTML/CSS web pages:

Browser Engine CSS-prefix

* **Internet Explorer** Trident -msie
* **Firefox** Gecko -moz
* **Opera** Presto -o
* **Safari & Chrome** WebKit -webkit

### ViewPort

The viewport is the area that determines how content is laid out and where text wraps on the webpage. The viewport can be larger or smaller than the visible area. If the viewport is larger than the visible area, then the user pans to see more of the web page. If the viewport is smaller than the visible area, then white space fills the viewport on either side of the content. For example, on iOS, the default viewport is 980px; therefor, a 980px wide web page will display its’ full width (i.e. no horizontal scrollbar).

Use the viewport meta tag to improve the presentation of your web content on a mobile device. Typically, you use the viewport meta tag to set the width and initial scale of the viewport. For example, if your web page is narrower than 980pixels, then you should set the width of the viewport to fit your web content. If you are designing a mobile touch-specific web application, then set the width to the width of the device. Because a mobile OS can run on devices with different screen resolutions, use the constants instead of numeric values when referring to the dimensions of a device. Use device-width for the width of the device and device-height for the height in portrait orientation. You do not need to set every viewport property. If only a subset of the properties are set, then the browser engine infers the other values. For example, if you set the scale to 1.0, the browser engine assumes the width is device-width in portrait and device-height in landscape orientation. Therefore, if you want the width to be 980pixels and the initial scale to be 1.0, then set both of these properties.

If you are designing a web application specifically for mobile, then the recommended size for your web pages is the size of the visible area on the device. It is recommended that you set the width to device-width so that the scale is 1.0 in portrait orientation and the viewport is not resized when the user changes to landscape orientation. If you do not change the viewport properties, the browser engine displays your web page in the upper-left corner (i.e. viewport width=980px). Setting the viewport width should be the first task when designing web applications for mobile to avoid the user zooming in before using your application.

<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.4; user-scalable=yes" />

where:

width=device-width or width=*nnn* where *nnn* is the number of pixels between 200 and 10000 (default: 980px)

height=device-height or height=*nnn* where *nnn* is the number of pixels between 223 and 10000

minimum-scale=*f.ff* where *f.ff* is a floating point number between 0.0 and 10.0 such as in (default : 0.25)

maximum-scale=*f.ff* where *f.ff* is a floating point number between 0.0 and 10.0 such as in (default: 1.6)

initial-scale=*f.ff* where *f.ff* is a floating point number between minimum-scale and maximum-scale

user-scalable= yes/no to allow the user to zoom in/out on the web page (default: yes)

### Media Queries

Media Queries is a CSS3 module allow content rendering to adapt to conditions such as screen resolution (e.g. smartphone screen vs. computer screen). It is a cornerstone technology of Responsive web design.

#### Different style sheets based upon media:

<link rel="stylesheet" media="mediatype and|not|only (media feature)" href="stylesheet.css" />

For example:

<link rel="stylesheet" media="screen" href="stylesheet.css" />

<link rel="stylesheet" media="print" href="printstylesheet.css " />

#### Different CSS attributes based upon media:

@media not|only *mediatype* and (*media feature*) {  
    CSS-Code;  
}

For example:

@media screen and (max-width: 300px) {  
    body {  
        background-color: lightblue;  
    }  
}

Media types include: all, print, screen, speech

Media features include:

|  |  |
| --- | --- |
| orientation | Specifies whether the display is in landscape mode or portrait mode |
| grid | Specifies whether the device is a grid device or not |
| device-height | Specifies the height of the device, such as a computer screen |
| device-width | Specifies the width of the device, such as a computer screen |
| max-device-height | Specifies the maximum height of the device, such as a computer screen |
| max-device-width | Specifies the maximum width of the device, such as a computer screen |
| min-device-height | Specifies the minimum height of the device, such as a computer screen |
| min-device-width | Specifies the minimum width of the device, such as a computer screen |
| height | Specifies the height of the display area, such as a browser window |
| width | Specifies the width of the display area, such as a browser window |
| max-height | Specifies the maximum height of the display area, such as a browser window |
| max-width | Specifies the maximum width of the display area, such as a browser window |
| min-height | Specifies the minimum height of the display area, such as a browser window |
| min-width | Specifies the minimum width of the display area, such as a browser window |

### CSS Reference

#### Combinators

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator. There are four different combinators in CSS3.

#### Descendant Selector

The descendant selector matches all element that are descendants of a specified element. The following example selects all <p> elements inside <div> elements:

div p { background-color: yellow; }

#### Child Selector

The child selector selects all elements that are the immediate children of a specified element. The following example selects all <p> elements that are immediate children of a <div> element:

div > p { background-color: yellow; }

#### Adjacent Sibling Selector

The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element. Sibling elements must have the same parent element, and "adjacent" means "immediately following". The following example selects all <p> elements that are placed immediately after <div> elements:

div + p { background-color: yellow; }

#### General Sibling Selector

The general sibling selector selects all elements that are siblings of a specified element. The following example selects all <p> elements that are siblings of <div> elements:

div ~ p { background-color: yellow; }