Cheatography

C# Naming Conventions Cheat Sheet

by Greg Finzer (GregFinzer) via cheatography.com/15280/cs/17465/

Casing Rules

Identifier	Case	Example
Namespace	Pascal	System.Drawing
Class	Pascal	Customer
Class Field	Camel	_transferConfig
Interface	Pascal	IPerson
Property	Pascal	TransferConfig
Method	Pascal	TransferAccount
Parameter	Camel	accountNumber
Constant	Pascal	InfiniteThrottle
Enumeration Type	Pascal	BrowserType
Enumeration Value	Pascal	InternetExplorer
Event	Pascal	TransferComplete
Exception Class	Pascal	TransferException
Pre-Processor	Upper	NETSTANDARD

Except for parameters, class fields, and pre-processor directives, all other identifiers follow a Pascal naming convention.

Namespaces

Choose names that indicate functionality

The general format for a namespace name is as follows:

<Company>.(<Project>)[.<Feature>][.<Subnamespace>]

Example: NewtonSoft.Json.Linq

Assemblies and DLL Names

Choose names that suggest large chunks of functionality.

It is advisable if assembly and DLL names follow the namespace names.

The following pattern may be followed for naming DLLs:

<Company>.<Component>.dll

Where *<component>* contains one or more dot separated clauses.

Example: NewtonSoft.Json.dll

Parameters

Choose parameter names that indicate what data is being affected.

Good: *firstName* - Uses camel casing and is descriptive

Bad: *decimalSalary* - Name should not be based on type

Resources

Nested identifiers with clear hierarchy

Example: *Menus.File.Close.Text*

Classes, Structs, and Interfaces

Use pascal cased nouns, noun phrases or adjective phrases like Customer or Invoice. This distinguishes type names from methods, which are named with verb phrases like SaveCustomer or LoadInvoice.

Use of suffixes and prefixes

Derived class should have suffix representing the base class. e,g OvalShape

TransferCompleteEventHandler – EventHandler suffix for handlers

TransferCompleteCallback – Callback suffix to delegates

TransferException – Exception suffix for deriving from Exception

AccountDictionary – Dictionary suffix for dictionary implementations

SocketStream - Stream suffix for inheriting from System.IO.Stream

Do use the prefix I for Interfaces. Example: ITransfer

General Naming Conventions

Do	Don't
Use easily readable identifier names. Favor readability over brevity.	Use underscore, hyphen or Hungarian notation.
Use semantically interesting generic names. eg. GetAmountDue vs GetDecimalValue	Use identifiers that conflict with C#
Use acronyms only if required, and only use widely accepted ones	Use abbreviations as part of the identifiers.

Types

Fields

Typically nouns or noun phrases are used as names for the fields. e.g. _salary

Properties

Nouns, noun phrases or adjectives are used for naming properties

Properties and Get methods should not be named alike.

Boolean properties should be named with phrases like Is or Has.

Methods

Typically verbs or verb phrases are used as names for the methods. e.g. GetEncodingString()

Events

Typically verbs or verb phrases are used as names for the events.

In event handlers, use two parameter named sender and e.

Concept of before and after should be given, e.g Closing, Closed, etc

By **Greg Finzer** (GregFinzer)
cheatography.com/gregfinzer/
www.kellermansoftware.com

Published 15th October, 2018.
Last updated 15th October, 2018.

Cheatography

C# Naming Conventions Cheat Sheet

by Greg Finzer (GregFinzer) via cheatography.com/15280/cs/17465/

Enums

Do not use prefixes or suffixes

Usually names are plural nouns. E.g Teams, Colors

Do not use flag as suffix for the names of flag enumerations

By **Greg Finzer** (GregFinzer)
cheatography.com/gregfinzer/
www.kellermansoftware.com

Published 15th October, 2018.
Last updated 15th October, 2018.

Cheatography

dotnet cli Cheat Sheet

by Olcay Bayram (oba) via cheatography.com/64093/cs/16212/

dotnet new

`dotnet new sln`	Solution file
`dotnet new console`	Console application
`dotnet new classlib`	Class library
`dotnet new mvc`	ASP.NET Core Web App (Model-View-Controller)
`dotnet new xunit`	xUnit test project
`dotnet new -l`	Obtain a list of the available templates

dotnet sln

`dotnet sln list`	List all projects in a solution file
`dotnet sln todo.sln add todo-app/todo-app.csproj`	Add a C# project to a solution
`dotnet sln todo.sln remove todo-app/todo-app.csproj`	Remove a C# project from a solution
`dotnet sln todo.sln add **/*.csproj`	Add multiple C# projects to a solution using a globbing pattern

dotnet add

`dotnet add package Newtonsoft.Json`	Add *Newtonsoft.Json* NuGet package to a project
`dotnet add reference lib1/lib1.csproj lib2/lib2.csproj`	Add multiple project references to the project in the current directory
`dotnet add app/app.csproj reference **/*.csproj`	Add multiple project references using a globbing pattern on Linux/Unix

dotnet build

`dotnet build`	Build a project and all of its dependencies
`dotnet build --configuration Release`	Build a project and its dependencies using Release configuration
`dotnet build --runtime ubuntu.16.04-x64`	Build a project and its dependencies for a specific runtime (in this example, Ubuntu 16.04)

Starting with .NET Core 2.0, you don't have to run `dotnet restore` because it's run implicitly.

dotnet run

`dotnet run`	Run the project in the current directory
`dotnet run --project ./projects/proj1/proj1.csproj`	Run the specified project
`dotnet myapp.dll`	Run a framework-dependent app named `myapp.dll`

dotnet clean

`dotnet clean`	Clean the output of a project
`dotnet clean --configuration Release`	Clean a project built using the Release configuration

Only the outputs created during the build are cleaned. Both intermediate (*obj*) and final output (*bin*) folders are cleaned.

dotnet publish

`dotnet publish`	Publish the project in the current directory
`dotnet publish ~/projects/app1/app1.csproj`	Publish the application using the specified project file

The `dotnet publish` command's output is ready for deployment to a hosting system (for example, a server, PC, Mac, laptop) for execution.

dotnet ef

`dotnet ef migrations add`	Add a new migration
`dotnet ef migrations list`	List available migrations
`dotnet ef migrations remove`	Remove the last migration
`dotnet ef migrations script`	Generate a SQL script from migrations
`dotnet ef database update`	Update the database to a specified migration
`dotnet ef database drop`	Drop the database
`dotnet ef dbcontext list`	List available DbContext types
`dotnet ef dbcontext info`	Get information about a DbContext type
`dotnet ef dbcontext scaffold`	Scaffolds a DbContext and entity types for a database

By **Olcay Bayram** (oba)
cheatography.com/oba/
en.otomatikmuhendis.com

Published 30th June, 2018.
Last updated 30th June, 2018.

Cheatography

dotnet cli Cheat Sheet

by Olcay Bayram (oba) via cheatography.com/64093/cs/16212/

dotnet pack

`dotnet pack`	Build the project and create NuGet packages
`dotnet pack --no-build --output nupkgs`	Pack the project in the current directory into the *nupkgs* folder and skip the build step
`dotnet pack /p:PackageVersion=2.1.0`	Set the package version to *2.1.0* with the *PackageVersion* MSBuild property

dotnet nuget

`dotnet nuget locals -l all`	Display the paths of all the local cache directories
`dotnet nuget push foo.nupkg -k 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a`	Push *foo.nupkg* to the default push source, specifying an API key
`dotnet nuget delete Microsoft.AspNetCore.Mvc 1.0 --non-interactive`	Delete version 1.0 of package *Microsoft.AspNetCore.Mvc*, not prompting user for credentials or other input

dotnet remove

`dotnet remove package Newtonsoft.Json`	Remove *Newtonsoft.Json* NuGet package from a project in the current directory
`dotnet remove reference lib/lib.csproj`	Remove a project reference from the current project
`dotnet remove app/app.csproj reference **/*.csproj`	Remove multiple project references using a glob pattern on Unix/Linux

etc.

`dotnet help`	Show more detailed documentation online for the command
`dotnet migrate`	Migrate a Preview 2 .NET Core project to a .NET Core SDK 1.0 project
`dotnet msbuild`	Provides access to a fully functional MSBuild
`dotnet test`	Run the tests in the project in the current directory
`dotnet list reference`	List the project references for the project in the current directory

Environment variables

DOTNET_PACKAGES

> The primary package cache.

DOTNET_SERVICING

> Specifies the location of the servicing index to use by the shared host when loading the runtime.

DOTNET_CLI_TELEMETRY_OPTOUT

> Specifies whether data about the .NET Core tools usage is collected and sent to Microsoft.

DOTNET_MULTILEVEL_LOOKUP

> Specifies whether .NET Core runtime, shared framework, or SDK are resolved from the global location.

DOTNET_ROLL_FORWARD_ON_NO_CANDIDATE_FX

> Disables minor version roll forward. For more information, see Roll forward.

By **Olcay Bayram** (oba)
cheatography.com/oba/
en.otomatikmuhendis.com

Published 30th June, 2018.
Last updated 30th June, 2018.