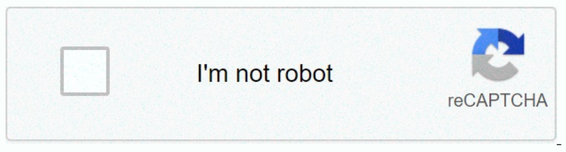
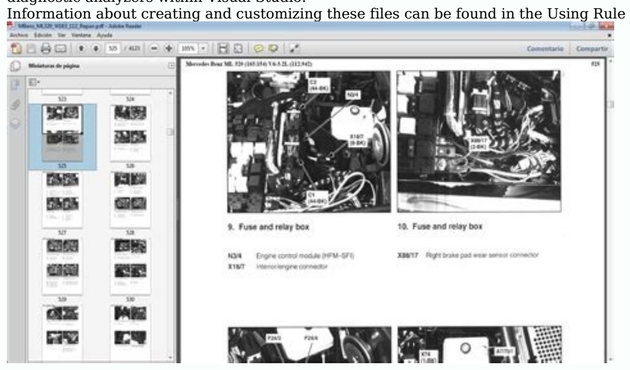


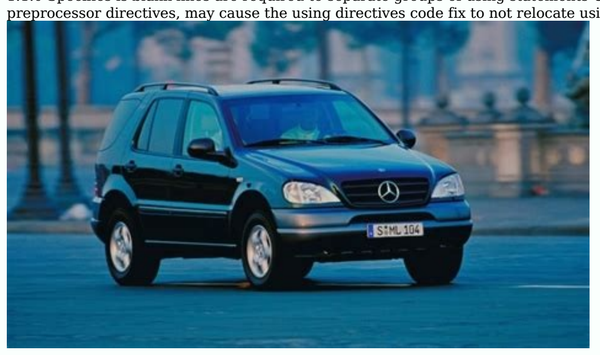
Continue

W163 repair manual pdf

StyleCop Analyzers is configured using two separate mechanisms: code analysis rule set files, and stylecop.json.
Code analysis rule set files Enable and disable individual rules Configure the severity of violations reported by individual rules stylecop.json Specify project-specific text, such as the name of the company and the structure to use for copyright headers Fine-tune the behavior of certain rules Code analysis rule sets are the standard way to configure most diagnostic analyzers within Visual Studio.
Information about creating and customizing these files can be found in the Using Rule Sets to Group Code Analysis Rules documentation on docs.microsoft.com.

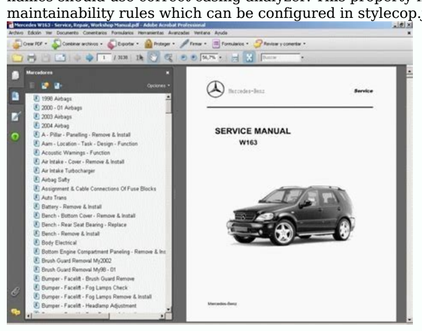


An example rule set file containing the default StyleCop Analyzers configuration is available at . Getting Started with stylecop.json The easiest way to add a stylecop.json configuration file to a new project is using a code fix provided by the project. To invoke the code fix, open any file where SA1633 is reported¹ and press Ctrl+. to bring up the Quick Fix menu. From the menu, select Add StyleCop settings file to the project. The dot file naming convention is also supported, which makes it possible to name the configuration file .stylecop.json. JSON Schema for IntelliSense A JSON schema is available for stylecop.json. By including a reference in stylecop.json to this schema, Visual Studio will offer IntelliSense functionality (code completion, quick info, etc.) while editing this file. The schema may be configured by adding the following top-level property in stylecop.json: { "$schema": " } The code fix described previously automatically configures stylecop.json to reference the schema. If the schema appears to be out-of-date in Visual Studio, right click anywhere in the stylecop.json document and then select Reload Schemas. Source Control For best results, stylecop.json should be included in source control. This will automatically propagate the expected settings to all team members working on the project. ⚠ If you are working in Git, make sure your .gitignore file does not contain the following line. This line should be removed if present. Indentation This section describes the indentation rules which can be configured in stylecop.json. Each of the described properties are configured in the indentation object, which is shown in the following sample file. { "settings": { "indentation": { } } } Basic Indentation The following properties are used to configure basic indentation in StyleCop Analyzers. Property Default Value Minimum Version Summary indentationSize 4 1.1.0 The number of columns to use for each indentation of code. Depending on the useTabs and tabSize settings, this will be filled with tabs and/or spaces. tabSize 4 1.1.0 The width of a hard tab character in source code. This value is used when converting between tabs and spaces. useTabs false 1.1.0 true to indent using hard tabs; otherwise, false to indent using spaces When working in Visual Studio, the IDE will not automatically adjust editor settings according to the values in stylecop.json. To provide this functionality as well, we recommend duplicating the basic indentation settings in a .editorconfig file. Users of the EditorConfig extension for Visual Studio will not need to update their C# indentation settings in order to match your project style. Spacing Rules This section describes the features of spacing rules which can be configured in stylecop.json. Each of the described properties are configured in the spacingRules object, which is shown in the following sample file. { "settings": { "spacingRules": { } } } Currently there are no configurable settings for spacing rules. Readability Rules This section describes the features of readability rules which can be configured in stylecop.json. Each of the described properties are configured in the readabilityRules object, which is shown in the following sample file. { "settings": { "readabilityRules": { } } } Aliases for Built-In Types Property Default Value Minimum Version Summary allowBuiltInTypeAliases false 1.1.0-beta007 Specifies whether aliases are allowed for built-in types. By default, SA1121 reports a diagnostic for the use of named aliases for built-in types: using HRESULT = System.Int32; HRESULT hr = SomeNativeOperation(); // SA1121 The allowBuiltInTypeAliases configuration property can be set to true to allow cases like this while continuing to report diagnostics for direct references to the metadata type name, Int32. Ordering Rules This section describes the features of ordering rules which can be configured in stylecop.json. Each of the described properties are configured in the orderingRules object, which is shown in the following sample file. { "settings": { "orderingRules": { } } } Element Order The following properties are used to configure element ordering in StyleCop Analyzers. Property Default Value Minimum Version Summary elementOrder ["kind", "accessibility", "constant", "static", "readonly"] 1.0.0 Specifies the traits used for ordering elements within a document, along with their precedence The elementOrder property is an array of element traits. The ordering rules (SA1201, SA1202, SA1203, SA1204, SA1214, and SA1215) evaluate these traits in the order they are defined to identify ordering problems, and the code fix uses this property when reordering code elements. Any traits which are omitted from the array are ignored. The following traits are supported: kind: Elements are ordered according to their kind (see SA1201 for this predefined order) accessibility: Elements are ordered according to their declared accessibility (see SA1202 for this predefined order) constant: Constant elements are ordered before non-constant elements static: Static elements are ordered before non-static elements readonly: Readonly elements are ordered before non-readonly elements This configuration property allows for a wide variety of ordering configurations, as shown in the following examples. Example: All Constants First The following example shows a customized element order where all constant fields are placed before non-constant fields, regardless of accessibility. { "settings": { "orderingRules": { "elementOrder": ["kind", "constant", "accessibility", "static", "readonly"] } } } Example: Ignore Accessibility The following example shows a customized element order where element accessibility is simply ignored, but other ordering rules remain enforced. { "settings": { "orderingRules": { "elementOrder": ["kind", "constant", "static", "readonly"] } } } Using Directives The following properties are used to configure using directives in StyleCop Analyzers. Property Default Value Minimum Version Summary systemUsingDirectivesFirst true 1.0.0 Specifies whether System using directives are placed before other using directives usingDirectivesPlacement "insideNamespace" 1.0.0 Specifies the desired placement of using directives blankLinesBetweenUsingGroups "allow" 1.1.0 Specifies is blank lines are required to separate groups of using statements Using Directives Placement The usingDirectivesPlacement property affects the behavior of the following rules which report incorrectly placed using directives. SA1200 Using directives should be placed correctly ⚠ Use of certain features, including but not limited to preprocessor directives, may cause the using directives code fix to not relocate using directives automatically. If SA1200 is still reported after applying the Fix All operation for using directives, the remaining cases will need to be resolved manually. This property has three allowed values, which are described as follows.

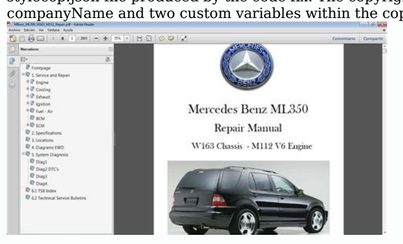


"insideNamespace" In this mode, using directives should be placed inside of namespace declarations. This is the default mode, and adheres to the original SA1200 behavior from StyleCop Classic. SA1200 reports using directives which are located outside of a namespace declaration (a few exceptions exist for cases where this is required) Using directives code fix moves using directives inside of namespace declarations where possible "outsideNamespace" In this mode, using directives should be placed outside of namespace declarations. SA1200 reports using directives which are located inside of a namespace declaration Using directives code fix moves using directives outside of namespace declarations where possible "preserve" In this mode, using directives may be placed inside or outside of namespaces. SA1200 does not report any violations Using directives code fix may reorder using directives, but does not relocate them Blank Lines Between Groups The blankLinesBetweenUsingGroups property affects the behavior of the following rules which report the presence / absence of blanks lines between groups of using directives. SA1516 Elements should be separated by blank line Using directives can grouped based on the purpose of the using directive. StyleCop Analyzers recognizes the following using directive group types: System using directives (only when systemUsingDirectivesFirst is true) Normal using directives Static using directives Alias using directives This property has three allowed values, which are described as follows. "allow" In this mode, a blank line between groups for using directives is optional.
No diagnostic will be produced. Using directives code fix will not insert blank lines.

"require" In this mode, a blank line between groups for using directives is mandatory. SA1516 reports missing blank lines between using directive groups. Using directives code fix will insert blank lines. SA1516 code fix will add a missing blank line. "omit" In this mode, a blank line between groups for using directives is not allowed. SA1516 reports blank lines between using directive groups. Using directives code fix will not insert blank lines. SA1516 code fix will remove blank lines between using directive groups.
Naming Rules This section describes the features of naming rules which can be configured in stylecop.json. Each of the described properties are configured in the namingRules object, which is shown in the following sample file. { "settings": { "namingRules": { } } } Hungarian Notation The following properties are used to configure allowable Hungarian notation prefixes in StyleCop Analyzers. Property Default Value Minimum Version Summary allowCommonHungarianPrefixes true 1.0.0 Specifies whether common non-Hungarian notation prefixes should be allowed. When true, the two-letter words 'as', 'at', 'by', 'do', 'go', 'if', 'in', 'is', 'it', 'no', 'of', 'on', 'or', and 'to' are allowed to appear as prefixes for variable names. allowedHungarianPrefixes [ ] 1.0.0 Specifies additional prefixes which are allowed to be used in variable names. See the example below for more information. The following example shows a settings file which allows the common prefixes as well as the custom prefixes 'md' and 'cd'. { "settings": { "namingRules": { "allowedHungarianPrefixes": ["cd", "md"] } } } Namespace Components The following property is used to configure allowable namespace components (e.g. ones that start with a lowercase letter). Property Default Value Minimum Version Summary allowedNamespaceComponents [ ] 1.2.0 Specifies namespace components that are allowed to be used. See the example below for more information. The following example shows a settings file which allows namespace components such as eBay or Apple.iPod. { "settings": { "namingRules": { "allowedNamespaceComponents": ["eBay", "iPod"] } } } Tuple element names The following properties are used to configure the behavior of the tuple element name analyzers. Property Default Value Minimum Version Summary includeInferredTupleElementNames false 1.2.0 Specifies whether inferred tuple element names will be analyzed as well. tupleElementNameCasing "PascalCase" 1.2.0 Specifies the casing convention used for tuple element names. The following example shows a settings file which requires tuple element names to use camel case for all tuple elements (including inferred element names). { "settings": { "namingRules": { "includeInferredTupleElementNames": true, "tupleElementNameCasing" : "camelCase" } } } Tuple Element Name Casing The tupleElementNameCasing property affects the behavior of the SA1316 Tuple element names should use correct casing analyzer. This property has two allowed values, which are described as follows. "camelCase" In this mode, tuple element names must start with a lowercase letter. "PascalCase" In this mode, tuple element names must start with an uppercase letter. Maintainability Rules This section describes the features of maintainability rules which can be configured in stylecop.json. Each of the described properties are configured in the maintainabilityRules object, which is shown in the following sample file.



{ "settings": { "maintainabilityRules": { } } } The following properties are used to configure maintainability rules in StyleCop Analyzers. Property Default Value Minimum Version Summary topLevelTypes ["class"] 1.1.0 Specifies which kind of types that should be placed in separate files The topLevelTypes property is an array which specifies which kind of types that should be placed in separate files according to rule SA1402. The following types are supported: class interface struct enum delegate Layout Rules This section describes the features of layout rules which can be configured in stylecop.json. Each of the described properties are configured in the layoutRules object, which is shown in the following sample file. { "settings": { "layoutRules": { } } } The following properties are used to configure layout rules in StyleCop Analyzers. Property Default Value Minimum Version Summary newlineAtEndOfFile "allow" 1.0.0 Specifies the handling for newline characters which appear at the end of a file allowConsecutiveUsings true 1.1.0 Specifies if SA1519 will allow consecutive using statements without braces allowDoWhileOnClosingBrace false >1.2.0 Specifies if SA1500 will allow the while expression of a do/while loop to be on the same line as the closing brace, as is generated by the default code snippet of Visual Studio Lines at End of File The behavior of SA1518 can be customized regarding the manner in which newline characters at the end of a file are handled. The newlineAtEndOfFile property supports the following values: "allow": Files are allowed to end with a single newline character, but it is not required "require": Files are required to end with a single newline character "omit": Files may not end with a newline character Consecutive using statements without braces The behavior of SA1519 can be customized regarding the manner in which consecutive using statements without braces are treated. The allowConsecutiveUsings property specifies the behavior: true: consecutive using statements without braces will not produce diagnostics false: consecutive using statements without braces will produce a SA1519 diagnostic This only allows omitting the braces for a using followed by another using statement. A using statement followed by any other type of statement will still require braces to used. Do-While Loop Placement The behavior of SA1500 can be customized regarding the manner in which the while expression of a do/while loop is allowed to be placed. The allowDoWhileOnClosingBrace property specified the behavior: true: the while expression of a do/while loop may be placed on the same line as the closing brace or on a separate line false: the while expression of a do/while loop must be on a separate line from the closing brace Documentation Rules This section describes the features of documentation rules which can be configured in stylecop.json. Each of the described properties are configured in the documentationRules object, which is shown in the following sample file. { "settings": { "documentationRules": { } } } Copyright Headers The following properties are used to configure copyright headers in StyleCop Analyzers. Property Default Value Minimum Version Summary companyName "PlaceholderCompany" 1.0.0 Specifies the company name which should appear in copyright notices copyrightText "Copyright (c) {companyName}. All rights reserved." 1.0.0 Specifies the default copyright text which should appear in copyright headers xmlHeader true 1.0.0 Specifies whether file headers should use standard StyleCop XML format, where the copyright notice is wrapped in a element variables n/a 1.0.0 Specifies replacement variables which can be referenced in the copyrightText value headerDecoration n/a 1.1.0 This value can be set to add a decoration for the header comment so headers look similar to the ones generated by the StyleCop Classic ReSharper fix Configuring Copyright Text In order to successfully use StyleCop-checked file headers, most projects will need to configure the companyName property. The companyName property is so frequently customized that it is included in the default stylecop.json file produced by the code fix. The copyrightText property is a string which may contain placeholders. Each placeholder has the form {variable}, where variable is either a built-in variable (see below), or the name of a property in the variables property. The following sample file shows a custom stylecop.json file which references both companyName and two custom variables within the copyrightText. { "settings": { "documentationRules": { "companyName": "FooCorp", "copyrightText": "Copyright (c) {companyName}.



All rights reserved.Licensed under the {licenseName} license. See {licenseFile} file in the project root for full license information.", "variables": { "licenseName": "MIT", "licenseFile": "LICENSE" } } } } With the above configuration, a file TypeName.cs would be expected to have the following header. // // Copyright (c) FooCorp. All rights reserved. // Licensed under the MIT license. See LICENSE file in the project root for full license information. // Built-In Variables Variable Meaning companyName The value of the companyName configuration property in stylecop.json fileName The file name of the current source file If a fileName variable is explicitly included within the variables property of stylecop.json, that value will be used instead of the name of the current source file. Configuring XML Headers When the xmlHeader property is true (the default), StyleCop Analyzers expects file headers to conform to the following standard StyleCop format. // // {copyrightText} // When the xmlHeader property is explicitly set to false, StyleCop Analyzers expects file headers to conform to the following customizable format. Configuring Copyright Text Header Decoration The headerDecoration property is a string which can contain text that's used for decorating the generated header so headers look similar to the ones generated by the StyleCop Classic ReSharper fix. The default value for the headerDecoration property is empty, so no decoration will be added. The header decoration is not checked, it's only used for fixing the header. { "settings": { "documentationRules": { "companyName": "FooCorp", "copyrightText": "Copyright (c) {companyName}. All rights reserved.", "headerDecoration": "--" } } } With the above configuration, the fix for a file TypeName.cs would look like the following. // -- // // Copyright (c) FooCorp. All rights reserved. // // -- Documentation Requirements StyleCop Analyzers includes rules which require developers to document the majority of a code base by default. This requirement can easily overwhelm a team which did not use StyleCop for the entire development process. To help guide developers towards a properly documented code base, several properties are available in stylecop.json to progressively increase the documentation requirements. Property Default Value Minimum Version Summary documentInterfaces true 1.0.0 Specifies whether interface members need to be documented.
When true, all interface members require documentation, regardless of accessibility. documentExposedElements true 1.0.0 Specifies whether exposed elements need to be documented. When true, all publicly-exposed types and members require documentation. documentInternalElements true 1.0.0 Specifies whether internal elements need to be documented. When true, all internally-exposed types and members require documentation.
documentPrivateElements false 1.0.0 Specifies whether private elements need to be documented. When true, all types and members except for declared private fields require documentation. documentPrivateFields false 1.0.0 Specifies whether private fields need to be documented.
When true, all fields require documentation, regardless of accessibility.
These properties affect the behavior of the following rules which report missing documentation. Rules which report incorrect or incomplete documentation continue to apply to all documentation comments in the code. The following example shows a configuration file which requires developers to document all publicly-accessible members and all interfaces (regardless of accessibility), but does not require other internal or private members to be documented. Documenting interfaces is a low-effort task compared to documenting an entire code base, but provides high value in the fact that it covers the sections of code most likely to impact cross-team usage scenarios. { "settings": { "documentationRules": { "documentInterfaces": true, "documentInternalElements": false } } } Documentation Culture Some documentation rules require summary texts to start with specific strings. To allow teams to document their code in their native language, stylecop.json contains the documentationCulture property. Property Default Value Minimum Version Summary documentationCulture "en-US" 1.1.0 Specifies the culture or language to be used for certain documentation texts. This property affects the behavior of the following rules which report incorrect documentation. The default value for documentationCulture is fixed instead of reflecting the user's system language. This is to ensure that different developers working on the same project always use the same value. The following values are currently supported. Unsupported values will automatically fall back to the default value. "de-DE" "en-GB" "en-US" "es-MX" "fr-FR" "pl-PL" "pt-BR" "ru-RU" { "settings": { "documentationRules": { "documentationCulture": "de-DE" } } } File naming conventions The fileNamingConvention property will determine how the SA1649 File name should match type name analyzer will check file names. Given the following code: public class Class1 { } The analyzer will expect file names according the table below. When the fileNamingConvention property is not set, the stylecop convention is used as default. File naming convention Expected file name stylecop Class1{T1,T2,T3}.cs metadata Class1`3.cs Text ending with a period The SA1629 Documentation Text Must End With A Period analyzer checks if sections within XML documentation end with a period. The following properties can be used to control the behavior of the analyzer: Property Default Value Minimum Version Summary excludeFromPunctuationCheck ["seealso"] 1.1.0 Specifies the top-level tags within XML documentation that will be excluded from analysis.
Sharing configuration among solutions It is possible to define your preferred configuration once and reuse it across multiple independent projects. This involves rolling out your own NuGet package, which will contain the stylecop.json configuration and potentially a custom ruleset file. A custom .props file glues that configuration to any project that will use the NuGet package. Example .nuspec file: acme.stylecop 1.0.0 Example .props file: Page 2 You can’t perform that action at this time. You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session. Page 3 You can’t perform that action at this time. You signed in with another tab or window.
Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session.