File Content Replacer

File Content Replacer is the build feature which processes text files by performing regular expression replacements before a build. After the build, it restores the file content to the original state.

> File Content Replacer should be used with the automatic checkout only: after this build feature is configured, it will run before the first build step. TeamCity will first perform replacement in the specified files found in the build checkout directory and then run your build.

The common case of using File Content Replacer is replacing one attribute at a time in particular files, e.g. it can be used to patch files with the build number.

You can add more than one File Content Replacer build feature if you wish to:

- replace more than one attribute,
- replace one and the same attribute in different files/projects with different values.

This feature extends the capabilities provided by AssemblyInfo Patcher.

Check the Adding Build Features section for notes on how to add a build feature.

On this page:

File Content Replacer Settings

You can specify the values manually or use value presets for replacement, which can be edited if needed.

Option	Description
Template (optional):	File Content Replacer provides a template for every attribute to be replaced. Clicking the Load Template... button displays the combobox with templates containing value presets for replacement. The templates can be filtered by language (e.g. `C#`), file (e.g. `AssemblyInfo`) or attribute (e.g. `AssemblyVersion`) by typing in the combobox. When a template is selected, the settings are automatically filled with predefined values. See the section below for template details.
Process files:	Click Edit file list and specify paths to files where the values to be replaced will be searched. Provide a newline- or comma-separated set of rules in the form of +|-:[path relative to the checkout directory] . Ant-like wildcards are supported, e.g. dir/**/*.cs. If a pre-defined template is selected, the files associated with that template will be used.
File encoding:	By default, TeamCity will auto-detect the file encoding. To specify the encoding explicitly, select it from the drop-down. When specifying a custom encoding, make sure it is supported by the agent. If a pre-defined template is selected, the file encoding associated with that template will be used.
Find what:	Specify a pattern to search for, in the regular expression format. MULTILINE mode is on by default. If a pre-defined template is selected, the pattern associated with that template will be used.
Match case:	By default, the comparison is case-sensitive. Uncheck for case-insensitive languages. If a pre-defined template is selected, the comparison associated with that template will be used.

Regex:	Since TeamCity 2017.1 The box is checked by default and applies to both the search and replacement strings. Uncheck to use fixed strings.
	
	⚠ If you use versioned settings (XML or Kotlin DSL), in addition to the default `REGEX` and non-default `FIXED_STRINGS mode, the REGEX_MIXED mode is available`. In this mode, the search pattern is interpreted as a regular expression, but the replacement text will be quoted (the \and $ characters have no special meaning).
	
	See a sample File Content Replacer configuration for settings in Kotlin:
	
	`features {`
	`  replaceContent {`
	`  fileRules = "**/*"`
	`  pattern = "(?iu)the\h+pattern\h+to\h+search\h+for"`
	`  regexMode = FileContentReplacer.RegexMode.REGEX_MIXED`
	`  replacement = """%teamcity.agent.work.dir%\nd_r\bin\isf"""`
	`  }`
	`}`

Replace with:	Type the text to be used for replacing the characters in the Find what box. To delete the characters in the Find what box from your file, leave this box blank. $N sequence references N-th capturing group. All backslashes (\) and dollar signs ($) without a special meaning should be quoted (as \\ and \$, respectively).

Templates

This section lists the available replacement templates.

.NET templates

The templates for replacing the following Assembly attributes are provided (see this section for comparison with AssemblyInfo Patcher):

- `AssemblyTitle`
- `AssemblyDescription`
- `AssemblyConfiguration`
- `AssemblyCompany`
- `AssemblyProduct`
- `AssemblyCopyright`
- `AssemblyTrademark`
- `AssemblyCulture`
- `AssemblyVersion`
- `AssemblyFileVersion`
- `AssemblyInformationalVersion`
- `AssemblyKeyFile`
- `AssemblyKeyName`
- `AssemblyDelaySign`
- `CLSCompliant`
- `ComVisible`
- `Guid`
- `InternalsVisibleTo`
- `AllowPartiallyTrustedCallers`
- `NeutralResourcesLanguageAttribute`

.Net Core csproj templates

The csproj format is supported since TeamCity 2017.1:

- `AssemblyName`
- `AssemblyTitle`
- `AssemblyVersion`
- `Authors`
- `Company`
- `Copyright`
- `Description`
- `FileVersion`
- `PackageId`
- `PackageVersion`
- `Product`
- `Title`
- `Version`
- `VersionPrefix`
- `VersionSuffix`

MFC templates

The templates for replacing the following MFC C++ resource keys are are provided:

- `FileDescription`
- `CompanyName`
- `ProductName`
- `LegalCopyright`
- `FileVersion*`
- `ProductVersion*`

In MFC *.rc files, both `FileVersion` and `ProductVersion` occur twice, once in a dot-separated (e.g. **`1.2.3.4`**) and once in a comma-separated (e.g. **`1,2,3,4`**) form. If your `%build.number%` parameter has a format of **`1.2.3.{0}`**, using two build parameters with different delimiters instead of a single `%build.number%` is recommended.

Xcode templates

The templates for replacing the following Core Foundation Keys in `Info.plist` files are provided:

- `CFBundleVersion`
- `CFBundleShortVersionString`
- or both `CFBundleVersion` and `CFBundleShortVersionString` at the same time

Examples

Extending an attribute value with a custom suffix

Suppose you do not want to replace your `AssemblyConfiguration` with a fixed literal, but want to preserve your `AssemblyConfiguration` from `AssemblyInfo.cs` and just extend it with a custom suffix, e.g.:
`[assembly: AssemblyConfiguration("${AssemblyConfiguration} built by TeamCity")])`

Do the following:

change the default replacement	$1MyAssemblyConfiguration$7	to	$1$5 built by TeamCity$7

For changing complex regex patterns, this external tool might be useful.

Patching only specified files

The default `AssemblyInfo` templates follow the usual Visual Studio project/solution layout; but a lot of information may be shared across multiple projects and can reside in a shared file (e.g. `CommonAssemblyInfo.cs`).

Suppose you want to patch this shared file only; or you want to patch `AssemblyInfo.cs` files on a per-project bassis.

Do the following:

1. Load the `AssemblyInfo` template corresponding to the attribute you are trying to process (e.g. `AssemblyVersion`)
2. Change the list of file paths in the Look in field from the default `*/Properties/AssemblyInfo.cs` to `*/**CommonAssemblyInfo.cs**` or list several files comma- or new-line separated files here, e.g. `myproject1/Properties/AssemblyInfo.cs`, `myproject2/Properties/AssemblyInfo.cs` .

Specifying path patterns which contain spaces

Spaces are usually considered a part of the pattern, unless they follow a comma, as the comma is recognised as a delimiter.

Note that the TeamCity server UI trims leading and trailing spaces in input fields, so a single-line pattern like `<spaces>foo.bar` will become `foo.bar` upon save. The following workarounds are available:

For a single pattern containing leading spaces	For [a single pattern containing] trailing spaces	Alternatively, to add a single pattern containing leading spaces, use an explicit inclusion rule:
, <spaces>foo.bar	foo.bar<spaces>,	+:<spaces>foo.bar|

Changing only the last version part / build number of the **`AssemblyVersion`** attribute:

Suppose, your `AssemblyVersion` in `AssemblyInfo.cs` is `Major.Minor.Revision.Build` (set to `1.2.3.*`) , and you want to replace the `Build` portion (following the last dot (the `*`) only.

1. Load the `AssemblyVersion` in AssemblyInfo (C#)_ template and change the default pattern:

   ```
   (^\s*\[\s*assembly\s*:\s*((System\s*\.)?\s*Reflection\s*\.)?\s*AssemblyVersion(Attribute)?\s*\(\s*@?\")(([0-9\*])+\.?)+(\"\s*\)\s*\])
   ```

   to

   ```
   (^\s*\[\s*assembly\s*:\s*((System\s*\.)?\s*Reflection\s*\.)?\s*AssemblyVersion(Attribute)?\s*\(\s*@?\")(([0-9\*]+\.)+)[0-9\*]+(\"\s*\)\s*\])
   ```

   and change the default replacement:

   ```
   $1\%build.number%$7
   ```

   to

   ```
   $1$5\%build.number%$7
   ```

   .

%build.number% format
Make sure your `%build.number%` format contains just a decimal number without any dots, or you may end up with a

non-standard version like `1.2.3.4.5.6.2600` (i.e. **`%build.counter%`** is a valid value here while **`4.5.6.%build.counter%`** is not).