Template Engine

*December 2015.*

Script Library

***Contents:***

[1 To do 1](#_Toc439717502)

[2 Formalnosti 1](#_Toc439717503)

[2.1 Pravila, zakoni 2](#_Toc439717504)

[2.2 Davki 2](#_Toc439717505)

[2.3 Pogodbe 2](#_Toc439717506)

[2.4 Povezave 2](#_Toc439717507)

[3 Začetek 3](#_Toc439717508)

# To do

Narediti je treba naslednje:



# Copy – Short user Manual (Draft)

(copied when automatic copy to MarkDown did not work – needs to change to .md)

# ExpandTemplate.ps1 – User Manual

## Overview

ExpandTemplate.ps1 is a **template expansion engine** written in PowerShell.  
It takes a text template file with **placeholders** ({{ ... }}) and expands them using **variables**, **environment variables**, and **filters**.

It is designed for:

* Generating configuration files
* Testing escape/unescape filters
* Automating documentation and scripts with parameterized text

## Syntax

.\ExpandTemplate.ps1 `

-Template <TemplateFile> `

-Output <OutputFile> `

[-Var <"Name=Value"[,...]>] `

[-Variables <Hashtable>] `

[-VarsFile <FileWithVars>] `

[-Encoding <Encoding>] `

[-Verbose] [-Debug]

## Parameters

### -Template <string>

Path to the template file (\*.tmpl) that contains placeholders.

### -Output <string>

Path to the expanded output file.

### -Var <array of strings>

Inline variable assignments in the form Name=Value.  
Can be repeated multiple times, or passed as an array:

-Var "Name1=Value1", "Name2=Value2"

-Var @("Name1=Value1", "Name2=Value2")

### -Variables <hashtable>

Hashtable of variables, e.g.:

-Variables @{ Project="MyProj"; Version="1.2.3" }

### -VarsFile <string>

Optional file with variable assignments (line format: Name=Value).  
Values from -Var and -Variables override -VarsFile.

### -Encoding <string>

Encoding of the output file. Default: UTF8.

### -Verbose / -Debug

Enable detailed or very detailed tracing of placeholder parsing and filter application.

## Placeholders

### Format

{{ head | filter1[:arg1[:arg2...]] | filter2 ... }}

* **Head**:
  + var.Name → variable defined via -Var, -Variables, or -VarsFile
  + env.NAME → environment variable
* **Filters**: Transformations applied left-to-right.

### Examples

{{ var.Project }} → expands to "MyProj"

{{ env.USERNAME | upper }} → expands to uppercase user name

{{ var.PathWin | pathappend:"bin" }}

→ expands to a path with "bin" appended

## Filters

### String Manipulation

* lower – convert to lowercase
* upper – convert to uppercase
* trim – trim whitespace
* append:Text – append text
* prepend:Text – prepend text
* replace:Old:New – replace substring

### Path Handling

* pathappend:Part – append path component
* pathwin / pathwinabs – normalize to Windows path
* pathlinux / pathlinuxabs – normalize to Linux path
* pathos / pathosabs – normalize to current OS
* pathquote – quote path safely

### Escaping/Encoding

* esccs / fromesccs – C# escaping
* escjava / fromescjava – Java escaping
* escc / fromescc – C/C++ escaping
* urlenc / urldec – URL encoding/decoding
* xmlenc / xmldec – XML escaping/unescaping
* regq – regex quoting
* regesc – regex escaping

### Examples

{{ var.MyVarSimple | append:"\_end" }}

→ "value\_end"

{{ var.MyVarLong | replace:"demo":"test" | upper }}

→ all "demo" replaced with "test", result in uppercase

{{ var.PathWin | pathappend:"dir1\dir2\file.txt" | replace:"\\":"/" }}

→ normalized Unix-style path

## Unquoted Filter Arguments

Filter arguments **do not need quotes** if they contain only:

* letters, digits, underscores
* no whitespace
* no :, |, or }

Examples:

{{ var.MyVarSimple | append:suffix }}

{{ var.PathWin | replace:\\:/ }}

Quoted form is still valid (and required if spaces or special chars are present):

{{ var.MyVarSimple | append:" with space" }}

## Escaping Placeholders

To emit literal {{ or }} in the output:

{{ "{{" }} → outputs {{

{{ "}}" }} → outputs }}

## Error Handling

* Unknown filters → error with filter name
* Unclosed placeholders → error with position
* Null input to filters → warning/error depending on context
* Variables not found → empty string

## Example

Template (example.tmpl):

Project: {{ var.Project }}

User: {{ env.USERNAME | upper }}

Path: {{ var.PathWin | pathappend:bin | replace:\\:/ }}

Escaped: {{ var.Special | escc }}

Call:

.\ExpandTemplate.ps1 `

-Template example.tmpl `

-Output example.txt `

-Var "Project=DemoApp" `

-Variables @{ PathWin="C:\apps\demo\"; Special="A\nB" }

Output (example.txt):

Project: DemoApp

User: ADMINUSER

Path: C:/apps/demo/bin

Escaped: A\nB

## Notes & Best Practices

* Use -Debug to trace placeholder parsing and filter pipelines.
* Use -Verbose for higher-level overview.
* Prefer quoted arguments unless you’re confident about unquoted safety.
* Keep template files in version control; generated files should not be checked in.

# Proper Reference Manual for the Template Engine (ExpandTemplate.ps1)

# ExpandTemplate.ps1 Reference Manual

## 1. Overview

ExpandTemplate.ps1 is a lightweight template engine implemented in PowerShell.  
It expands placeholders in a template file (.tmpl) into an output file using:

* **Variables** (-Var, -Variables, -VarsFile)
* **Environment variables**
* **Filters** (string transformers)

Its design is similar to template engines like Jinja or Mustache, but with PowerShell-style extensibility.

## 2. Syntax

### Placeholders

{{ var.Name | filter1:arg1:arg2 | filter2 }}

{{ env.ENVVAR | filter }}

* Placeholders are enclosed in {{ and }}.
* The **head** is either:
  + var.Name — a variable passed via script parameters
  + env.ENVNAME — an environment variable
* After the head, a **pipeline** of filters can be chained with |.

### Escaping Literal Braces

* Use {{{{ → outputs literal {{
* Use }}}} → outputs literal }}

## 3. Variables

### Sources

1. **-Var (CLI arguments)**  
   Example:
2. -Var @("MyVar=Hello", "Path=C:\temp")
3. **-Variables (hashtable)**  
   Example:
4. -Variables @{ MyVar="Hello"; Path="C:\temp" }
5. **-VarsFile (JSON or simple key=value file)**

### Precedence

VarsFile < Variables < Var

## 4. Filters

### General Notes

* Filters are **case-insensitive** (append = APPEND).
* Filter arguments:
  + **Quoted**: "text with spaces", "C:\path\to\file"
  + **Unquoted**: word (no whitespace, :, |, or } inside)
* Escaping inside quoted arguments:
  + \" → "
  + \\ → \

### String Manipulation

* upper → Uppercase string
* lower → Lowercase string
* trim → Trim whitespace
* append:TEXT → Append string
* prepend:TEXT → Prepend string
* replace:OLD:NEW → Replace substring

### Path Manipulation

* pathappend:SUBPATH → Join as path segment
* pathwin / pathwinabs → Normalize to Windows path (relative/absolute)
* pathlinux / pathlinuxabs → Normalize to Linux path
* pathos / pathosabs → Normalize to current OS

### Escaping/Unescaping

* esccs / fromesccs → C#-style escaping
* escjava / fromescjava → Java-style escaping
* escc / fromescc → C/C++-style escaping
* escurl / fromescurl → URL percent-encoding
* escxml / fromescxml → XML entities

### Regex-related

* regesc → Regex-escape a string
* regq → Wrap in \Q...\E for regex quoting

## 5. Examples

### Simple Replacement

Template:

Hello {{ var.Name }}!

Vars:

-Var "Name=World"

Output:

Hello World!

### Filters

Original: {{ var.MyVar }}

Upper: {{ var.MyVar | upper }}

### Paths

Win Path: {{ var.Path | pathwin }}

Linux Path: {{ var.Path | pathlinux }}

### Escaping

C string literal: {{ var.Input | escc }}

Round-trip: {{ var.Input | escc | fromescc }}

## 6. Advanced Behavior

* **Filter arguments without quotes** are accepted when they don’t contain whitespace, :, |, or }.
* **Round-trip correctness**: fromesc\* | esc\* should match fromesc\* alone.
* **Canonicalization**:
  + Null character → \0
  + Control characters → \uXXXX
  + Non-ASCII BMP → \uXXXX
  + Outside BMP → \UXXXXXXXX

## 7. Debugging

* -Verbose → High-level tracing (placeholders, values).
* -Debug → Low-level tracing (filter application, args).
* Internal dev flag $Trace (if enabled in script) → extremely detailed step-by-step.

## 8. Limitations

* **Nested variable expansion** (e.g. {{ var.{{ var.Name }} }}) not supported.
* No loops/conditionals — this is a straight text expander with filters.
* Performance optimized for clarity, not massive scale.

## 9. Extending with New Filters

To add a filter:

1. Open ExpandTemplate.ps1.
2. Locate Apply-Filters.
3. Add a new switch ($name) { ... } clause:
4. 'reverse' { $val = -join ($val.ToCharArray() | [Array]::Reverse()); continue }
5. Use in template:
6. {{ var.MyVar | reverse }}

⚡