Preprocessor

The WiX preprocessor lets you use variables in your WiX authoring, share fragments of WiX authoring, and conditionally include or exclude XML elements based on conditional expressions. For example, you can conditionally include a fragment based on the value of an environment variable:

```
<?if $(env.MySku) = "Enterprise" ?>
  <?include EnterpriseStuff.wxi ?>
<?endif ?>
```

Include files

The `<?include?>` processing instruction inserts the contents of the specified file into the XML that is then passed to the WiX compiler.

The root element of the include file must be `Include`. The content within the `Include` element are inserted into the XML, replacing the `<?include?>` processing instruction.

Variables

WiX supports three types of variables:

Type	Example	Value
Environment variables	`$(env._NtPostBld)`	Environment variable `%_NtPostBld%`
Built-in WiX variables	`$(sys.CURRENTDIR)`	Built-in variable for the current directory
User-defined variables	`$(MyVariable)`	User-defined variable `MyVariable`, set in authoring or via the build

The preprocessor evaluates variables throughout the entire XML document, including in `<?if?>` expressions and attribute values.

Environment variables

Any environment variable can be referenced with the syntax `$(env.VariableName)`. For example, to get the value of the environment variable `%_BuildArch%`, use `$(env._BuildArch)`. Environment variable names are case-insensitive.

Built-in system variables

WiX has some built-in variables. They are referenced with the syntax `$(sys.VARIABLENAME)` and are always in upper case.

Name	Value
BUILDARCH	The platform (x86, x64, arm64) this package is compiled for
BUILDARCHSHORT	The platform (X86, X64, A64) this package is compiled for. This variable is especially useful to match the ids of WiX's architecture-specific custom actions and DLLs.

Name	Value
CURRENTDIR	The current directory where the build process is running
SOURCEFILEDIR	The directory containing the file being processed
SOURCEFILEPATH	The full path to the file being processed
WIXMAJORVERSION	The major version of WiX as one number
WIXVERSION	The version of WiX in major.minor.patch.build format

TIP

All built-in directory variables end in a backslash.

User-defined variables

You can define custom variables using the `<?define?>` processing instruction in WiX authoring, using the `-define` or `-d` switches on the `wix build` command line, or `DefineConstants` property in a .wixproj MSBuild project.

TIP

Variable names are case-sensitive.

To define a variable but not give it a value:

```
<?define MyVariable ?>
```

```
<?define MyVariable = "Hello World" ?>
<?define MyVariable = "$(otherVariableContainingSpaces)" ?>
<?define MyVariable = "$(BuildPath)\x86\bin\" ?>
```

TIP

The value of the variable can refer to another variable.

We recommend quoting values, though the quotes can be omitted if the value is a single word with no whitespace.

To undefine a variable:

```
<?undef MyVariable ?>
```

To refer to a variable's value:

```
$(MyVariable)
```

TIP

In WiX v3, user-defined variables were referenced with the syntax `$(var.MyVariable)`. WiX v4 uses `$(MyVariable)` instead and supports `$(var.MyVariable)` for backward compatibility.

Conditional blocks

There are several conditional blocks that let you include or exclude XML based on conditions:

Name	Description
<?if *expression* ?>	Include the XML fragment if the provided expression evaluates to true.
<?ifdef *variablename* ?>	Include the XML fragment if the specified variable is defined.
<?ifndef *variablename* ?>	Include the XML fragment if the specified variable is *not* defined.
<?elseif *expression* ?>	Include the XML fragment if a preceding `<?if?>`, `<?ifdef?>`, or `<?ifndef?>` block condition wasn't met *and* the provided expression evaluates to true.
<?else?>	Include the XML fragment if *all* preceding `<?if?>`, `<?ifdef?>`, or `<?ifndef?>` block conditions weren't met.
<?endif?>	Indicates the end of the conditional blocks.

Conditional blocks always begin with `<?if?>`, `<?ifdef?>`, or `<?ifndef ?>`, followed by optional `<?elseif?>`s and one optional `<?else?>`, and must end with `<?endif?>`:

```
{<?if?>|<?ifdef?>|<?ifndef?>}
```

```
  <xml />
  <xml />
[<?elseif>]
  <xml />
  <xml />
[<?elseif>]
  <xml />
  <xml />
[<?else>]
  <xml />
  <xml />
<?endif?>
```

Expressions

The expressions used with `<?if?>` and `<?elseif?>` is a Boolean expression evaluated according to the following rules:

- The expression is evaluated left to right
- Expressions are case-sensitive with the following exceptions:
  - Environment variable names
  - The keywords `and`, `or`, and `not`
  - The `~=` operator is case-insensitive.
- All variables must use the `$()` syntax or they will be considered a literal value.
- To use a literal `$(`, escape the `$` with a second one: `$$(`
- Variables can be compared to a literal or another variable:
  - Comparisons with `=`, `!=`, and `~=` are string comparisons.
  - Comparisons with relational operators (`<`, `<=`, `>`, `>=`) can only be performed on integer values.

    - If the variable doesn't exist, evaluation will fail and an error will be raised.
- The operator precedence is as follows:
    1. `""`
    2. `()`, `$()`
    3. `<`, `>`, `<=`, `>=`, `=`, `!=`, `~=`
    4. `Not`
    5. `And`, `Or`
- Parentheses can be nested.
- Literals can be surrounded by quotes, although quotes are not required.
- Quotes and leading and trailing whitespace are stripped off literal values.
- Invalid expressions result in a preprocessor failure.

Checking for variable extistence

Use `<?ifdef?>` to check that a variable is defined. Use `<?ifndef?>` to check that a variable is *not* defined. Note that a variable with an empty value is still considered to be defined.

Examples

```
<?define myValue = "3"?>
<?define system32 = $(env.windir)\system32  ?>
<?define B = "good var" ?>
<?define C = 3 ?>
<?define IExist ?>
```

```
<?if $(Iexist) ?>                           <?endif?> <!-- true -->
<?if $(myValue) = 6 ?>                      <?endif?> <!-- false -->
<?if $(myValue) != 3 ?>                     <?endif?> <!-- false -->
<?if not "x" = "y" ?>                       <?endif?> <!-- true -->
<?if $(env.systemdrive) = "a" ?>            <?endif?> <!-- false -->
<?if 3 < $(myValue) ?>                      <?endif?> <!-- false -->
<?if $(B) = "good VAR" ?>                   <?endif?> <!-- false -->
<?if $(B) ~= "good VAR" ?>                  <?endif?> <!-- true -->
<?if $(A) and not $(env.MyEnvVariable) ?>  <?endif?> <!-- false -->
<?if $(A) Or ($(B) And $(myValue) >= 3) ?> <?endif?> <!-- true -->

<?ifdef IExist ?> <!-- true -->
<?else?>          <!-- false -->
<?endif?>
```

Errors and warnings

The preprocessor can show meaningful error and warning messages using `<?error message ?>` and `<?warning message ?>` processing instructions. When one of these preprocessor instructions is encountered the preprocessor will either display an error and stop the compile or display a warning and continue.

For example:

```
<?ifndef RequiredVariable ?>
    <?error RequiredVariable must be defined ?>
<?endif?>
```

Iteration

There is a single iteration processing instruction:

```
<?foreach variable-name in semi-colon-delimited-list ?>
  ...
<?endforeach?>
```

When this occurs the preprocessor:

- creates a private copy of the variable context
- sets the variable in the foreach statement to an iteration on the semicolon-delimited list
- generates an XML fragment with the variable substituted

For example:

```
<?foreach LCID in 1033;1041;1055 ?>
    <Fragment>
        <DirectoryRef Id='TARGETDIR'>
            <Component Id='MyComponent.$(LCID)' />
        </DirectoryRef>
    </Fragment>
<?endforeach?>
```

Escaping

The preprocessor treats the `$` character in a special way if it is followed by a `$` or `(`. If you want to use a literal `$$`, use `$$$$` instead. Every two `$` characters will be replaced with one. For example, `$$$$$` will be replaced with `$$$`.

Functions

The preprocessor supports the following functions:

```
$(fun.AutoVersion(x.y))
```

Gets an auto generated version number using the same scheme as .NET AssemblyVersion attribute. The parameters x.y specify the major and minor verion number. The build number is set to the number of days since 1/1/2000 and revision to the number of seconds since midnight divided by 2. Both values are calculated using UTC.

Edit this page